Understanding the data

To understand the data I have first created two functions and prepared all my imports from the relevant R libraries:

```
library(tensorflow)
library(keras)
library(plyr)
library(dplyr)
library(ggplot2)
library(magrittr)
library(tidyr)
library(sys)
library(caret)
library(magick)
library(fs)
library(abind)
library(imager)
library(purrr)

# FUNCTIONS

show_image
```

Explaining the functions. The first function uses imager to load the image and prepare a plot. I have wrapped this in a function for convenience and so other people can make use of it. The second function **get_image_info()** extracts the dimensions from an image and stores each dimension in an R list.

Creating a link to where the images are stored

The next step was to setup the relevant directories to understand where the images are stored:

```
dataset_dir
```

This takes some explaining:

- The **dataset_dir** variable links to a string path of where the cell images are stored
- The **train_dir** uses this path and then appends the next level down, which is the train directory
- The **test_dir** does the same thing, but for the test directory
- The **train_parasite** and **train_uninfected** variables use the **dir_ls** function to search the relevant parasite and uninfected folders for all the matching .png files in the directory. If these were other formats, then the postfix would need to be changed to suit

Getting the file path

The next step is to index one of the file paths to uses as a test image to view. I select the second index from both the train_parasite and train_uninfected images, which have been accessed by using the relevant **dir_ls** path:

```
parasite_image
```

Running the code will produce a cell with a parasite infection for malaria and the second will run an uninfected image and the dim commands will print out the image size and dimensions:

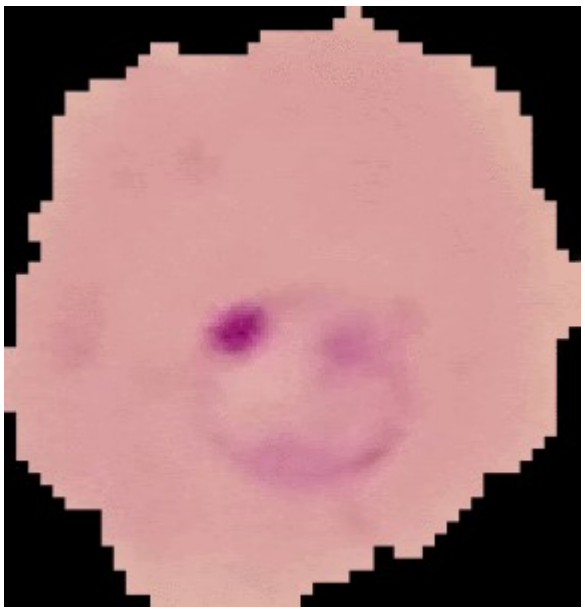
```
> dim(parasite_image)
[1] 148 208   1   3
> dim(uninfected_image)
[1] 145 136   1   3
```

Animating the images

The following codes uses Magick to animate the images for the parasitised cells and those that are uninfected:

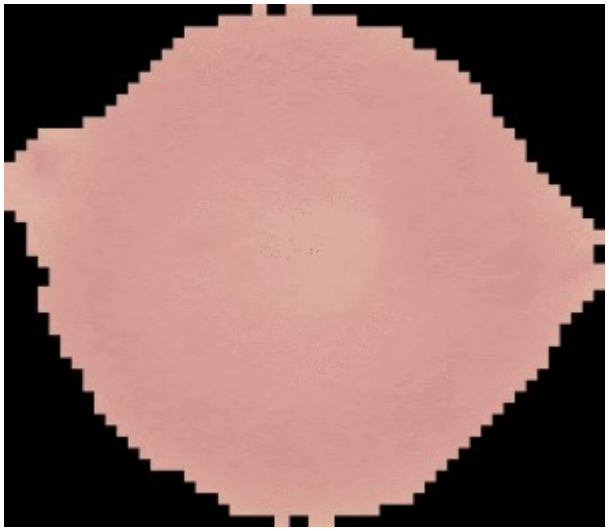
```
# Build infected cell animation
system.time(train_parasite[1:100] %>%
  map(image_read) %>%
  image_join() %>%
  image_scale("300x300") %>%
  image_animate(fps = .5) %>%
  image_write("Data/Parasite_Cells.gif"))

# Build Uninfected Cells animation
system.time(train_uninfected[1:100] %>%
              map(image_read) %>%
              image_join() %>%
              image_scale("300x300") %>%
              image_animate(fps = .5) %>%
              image_write("Data/Uninfected_Cells.gif"))
```

Running these two lines you get the resultant gifs of the first 100 images:



Malaria Parasite Infected Cells

The below shows the cells that are not infected:

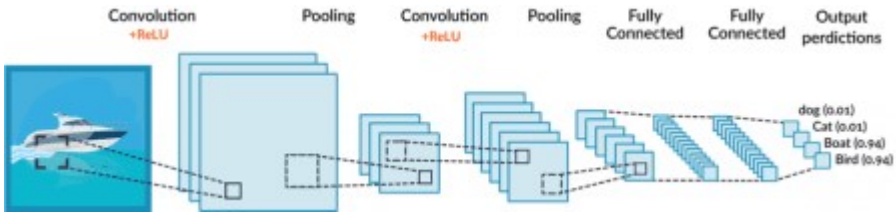


Uninfected cells

Building the CNN model skeleton

A bit about CNNs

The general process for a CNN is shown graphically here:



I will design these layers in our model baseline skeleton hereunder.

The model skeleton

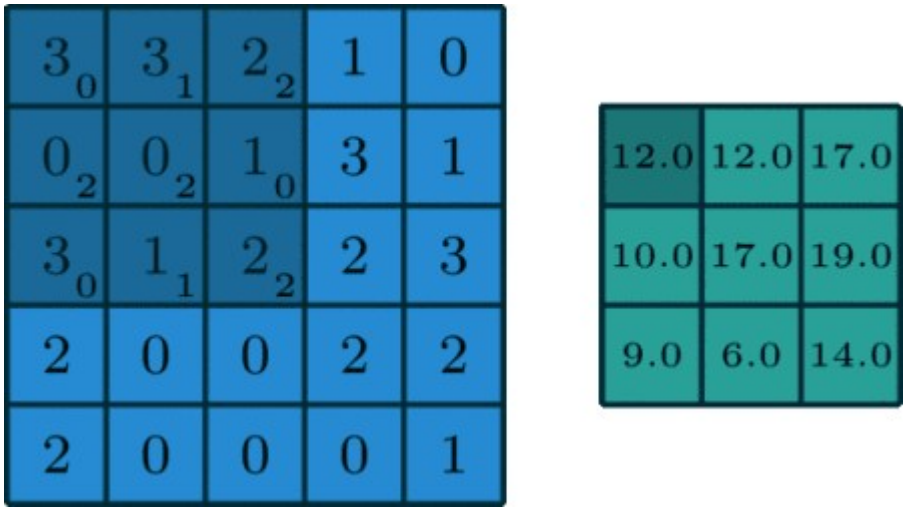
The first thing I will save is the target image shape and then I will start to build the model skeleton:

```
#Build Keras Baseline Model
model %
  layer_conv_2d(filters=32, kernel_size=c(3,3), activation = "relu",
                input_shape = image_shape) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_conv_2d(filters=64, kernel_size = c(3,3),
                input_shape = image_shape, activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%


  layer_conv_2d(filters=64, kernel_size = c(3,3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%


  layer_conv_2d(filters=32, kernel_size=c(3,3), activation = "relu",
                input_shape = image_shape) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
```

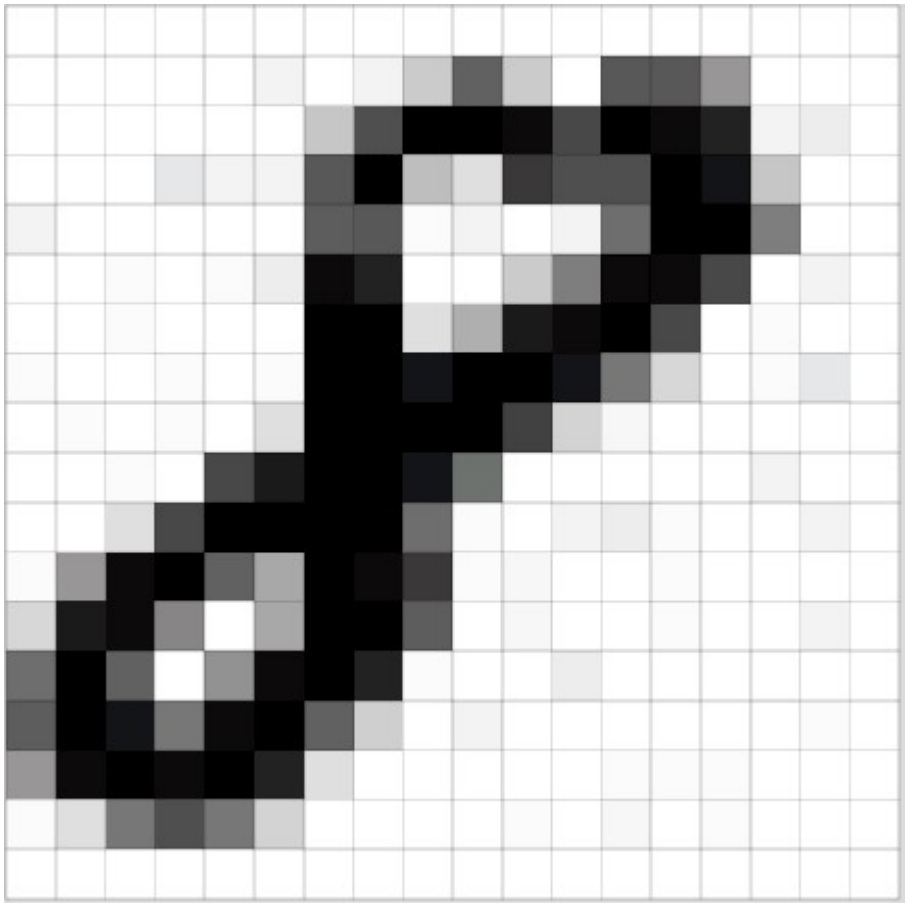
```
  layer_flatten() %>%
  layer_dense(1, activation = "sigmoid") %>%
  layer_dropout(0.5)
```

To explain this I will explain how a CNN is structured:

- The model is a **sequential** model so this needs to be the first statement, as each of the hidden layers connect together to be a fully connected network
- I create a 2 dimensional convolution over the image, set the filters to 32 to begin with. In the context of **CNN**, a **filter** is a set of learnable weights which are learned using the backpropagation algorithm. You can think of each **filter** as storing a single template/pattern and this is called a feature map. These filters allow for different maps to be created for each image – a little analogous to when you go to an optician for new lens and they test different filters out. The activation function is set to **relu** (rectified linear units), and this is now the defacto activation function for CNNs – along with Leaky-ReLu.
- This data then gets reduced or pooled by a process called max, min or average pooling:



The graphic shows the process of what this type of pooling reduction achieves:



- This process is repeated three times, one layer adding more feature maps , the next holding 64 constant and the final reducing back down to 32.
- The next part is to flatten the number of layers back down to one layer – implemented by the **layer_dense** function which will then produce the binary outcome I need. This time the activation is changed to sigmoid – as this is a binary classification task, the same as that used in logistic regression
- The final step is to use a method called dropout, invented by Geoffrey Hinton, the godfather of Neural Networks. Hear what he has to say.

Compiling the model

The next step is to compile the model. Here I use the loss equal to binary crossentropy (for multiclass classification – the best choice is categorical crossentropy) and you would need to change the activation function in the final dense layer to **softmax**.

The optimizer I used here is **rmsprop**, this takes the root mean square proportion across the classes, then I set the metrics equal to accuray:

```
model %>%
  compile(
    loss='binary_crossentropy',
    optimizer=optimizer_rmsprop(),
    metrics = c("acc")
  )

print(model)
```

Printing the compiled model prints the structure of the model for me:

```
print(model)
Model
Model: "sequential"
_________________________________________________________________
_________________________________
Layer (type)                                 Output Shape
Param #
=================================================================
=============================
conv2d (Conv2D)                              (None, 128, 128, 32)
896
_________________________________________________________________
_________________________________
max_pooling2d (MaxPooling2D)                 (None, 64, 64, 32)
0
_________________________________________________________________
_________________________________
conv2d_1 (Conv2D)                            (None, 62, 62, 64)
18496
_________________________________________________________________
_________________________________
max_pooling2d_1 (MaxPooling2D)               (None, 31, 31, 64)
0
_________________________________________________________________
_________________________________
conv2d_2 (Conv2D)                            (None, 29, 29, 64)
36928
_________________________________________________________________
_________________________________
max_pooling2d_2 (MaxPooling2D)               (None, 14, 14, 64)
0
_________________________________________________________________
_________________________________
conv2d_3 (Conv2D)                            (None, 12, 12, 32)
18464
_________________________________________________________________
_________________________________
max_pooling2d_3 (MaxPooling2D)               (None, 6, 6, 32)
0
_________________________________________________________________
_________________________________
flatten (Flatten)                            (None, 1152)
0
_________________________________________________________________
_________________________________
dense (Dense)                                (None, 1)
1153
_________________________________________________________________
_________________________________
dropout (Dropout)                            (None, 1)
0
=================================================================
```

```
=================================
Total params: 75,937
Trainable params: 75,937
Non-trainable params: 0
_________________________________________________________________
_________________________________
```

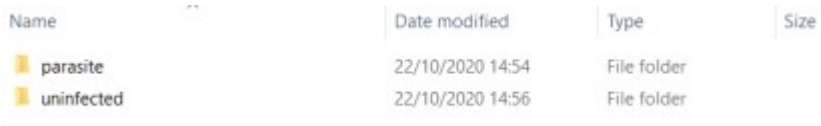
Rescaling and the awesome Keras flow_from_directory function

The first part is to rescale my train_datagen and test/validatioon data gen variables using the **image_data_generator** object from Keras:

```
train_datagen
```

Here I also set the batch size. This will be the size of the total training set / batch size equals the images to process in each batch.

Next, I use the **flow_images_from_directory** Keras method to look in my folder structure and use the training folder and the subfolders inside will be the classification labels:



Structure of cell images storage

The flow images from directory function only works when the images are organised in this structure, and it is worth it, as it saves tons of manual image labelling i.e. image 1 is a horse; image 2 is a cat; image 3 is a parasite infected malaria cell, you get the point.

Once this setup is in place you can run the functions and you will get this output:

```
> train_generator
```

The same message will be printed for the test_generator:

```
> test_generator
```

The target size here only expects the height and width dimensions – so this is why I have index sliced the image_shape vector.

Training the baseline model

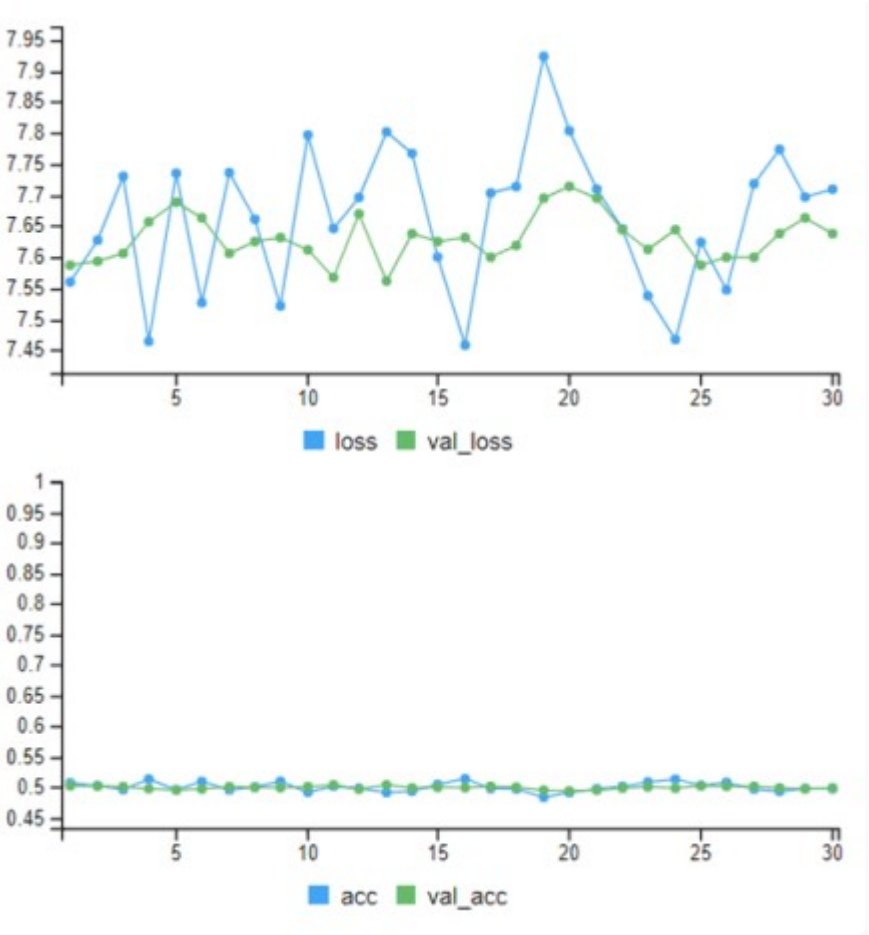
The last step is to use the the **fit_generator** function to train the model:

```
history % fit_generator(
  train_generator,
  steps_per_epoch = 150,
  epochs = 50,
  validation_data = test_generator,
  validation_steps = 75
)

model %>% save_model_hdf5("Data/parasite_cells_classification.h5")
```

Here I create a variable history – this is an old Python thing – and use the fit generator on the train generator object, we state we want 150 steps per epoch, with 50 epochs and we want to test the validation (test) data against the training set – with approx half the steps of the training steps.

Once this has completed when then save the model as **.h5** format, as this is accessed via reticulate (Python interface) and can only use the models in that format.

The output of the *baseline model is shown here:*



This has not done very well at all – it is failing to pick up on anything useful and has plateaued at about 50% accuracy. I am sure we can improve this…

Improving the baseline model

To improve the baseline model I will use this approach.

Fun with images and image augmentation

I will use image augmentation functions in Keras to enhance, add images as copies and do some rotations, shifting and scaling:

```
image_gen
```

From looking at the images in the directory, I think we need to focus in more on the parasite images. I cranked the zoom in Windows Picture Viewer to 80% enlargement and the parasite cells became more clear, so guess what I will apply this to every image and the augmented images using the zoom_range of 0.8 (80%).

Adding more flesh to our skeleton

This time I will add filters of 128 to the model and collapse to a dense layer of 512, before dropping down to 1 layer with a sigmoid and applying dropout:

```
model %
  layer_conv_2d(filters=32, kernel_size=c(3,3), activation = "relu",
                input_shape = image_shape) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_conv_2d(filters=64, kernel_size = c(3,3),
                input_shape = image_shape, activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%


  layer_conv_2d(filters=128, kernel_size = c(3,3),
                input_shape = image_shape, activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters=128, kernel_size = c(3,3),
                input_shape = image_shape, activation="relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_flatten() %>%
  layer_dense(512, activation = "relu") %>%
  layer_dense(1, activation = "sigmoid") %>%
  layer_dropout(0.5)

# Compile the model and add a learning rate

model %>%  compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(lr=1e-4),
  metrics = c("acc")
)
```

This shows the model summary of:

```
summary(model)

## Model output
Model: "sequential_1"
_____________________________________________________________________
_________________________________
```

```
Layer (type)                                 Output Shape
Param #
=================================================================
=============================
conv2d_4 (Conv2D)                            (None, 128, 128, 32)
896
_________________________________________________________________
_____________________________
max_pooling2d_4 (MaxPooling2D)               (None, 64, 64, 32)
0
_________________________________________________________________
_____________________________
conv2d_5 (Conv2D)                            (None, 62, 62, 64)
18496
_________________________________________________________________
_____________________________
max_pooling2d_5 (MaxPooling2D)               (None, 31, 31, 64)
0
_________________________________________________________________
_____________________________
conv2d_6 (Conv2D)                            (None, 29, 29, 128)
73856
_________________________________________________________________
_____________________________
max_pooling2d_6 (MaxPooling2D)               (None, 14, 14, 128)
0
_________________________________________________________________
_____________________________
conv2d_7 (Conv2D)                            (None, 12, 12, 128)
147584
_________________________________________________________________
_____________________________
max_pooling2d_7 (MaxPooling2D)               (None, 6, 6, 128)
0
_________________________________________________________________
_____________________________
flatten_1 (Flatten)                          (None, 4608)
0
_________________________________________________________________
_____________________________
dense_1 (Dense)                              (None, 512)
2359808
_________________________________________________________________
_____________________________
dense_2 (Dense)                              (None, 1)
513
_________________________________________________________________
_____________________________
dropout_1 (Dropout)                          (None, 1)
0
=================================================================
=============================
```

```
Total params: 2,601,153
Trainable params: 2,601,153
Non-trainable params: 0
_________________________________________________________________
_____________________________
```

Flowing images from directories again

This time we use the augmented model, with the adjustments e.g. increased zoom and other shifting adjustments. I have also increased the batch sizes from 16 to 32.

The code below implements this:

```
train_gen_augment
```

The same image sizes will be printed, as but this will add augmented images on top of these to bolster model sample size.

Fitting the model

The same code is used, but because this is a deeper network and takes about 50 mins to train, I will take down the steps per epoch and the number of epochs across the training set.

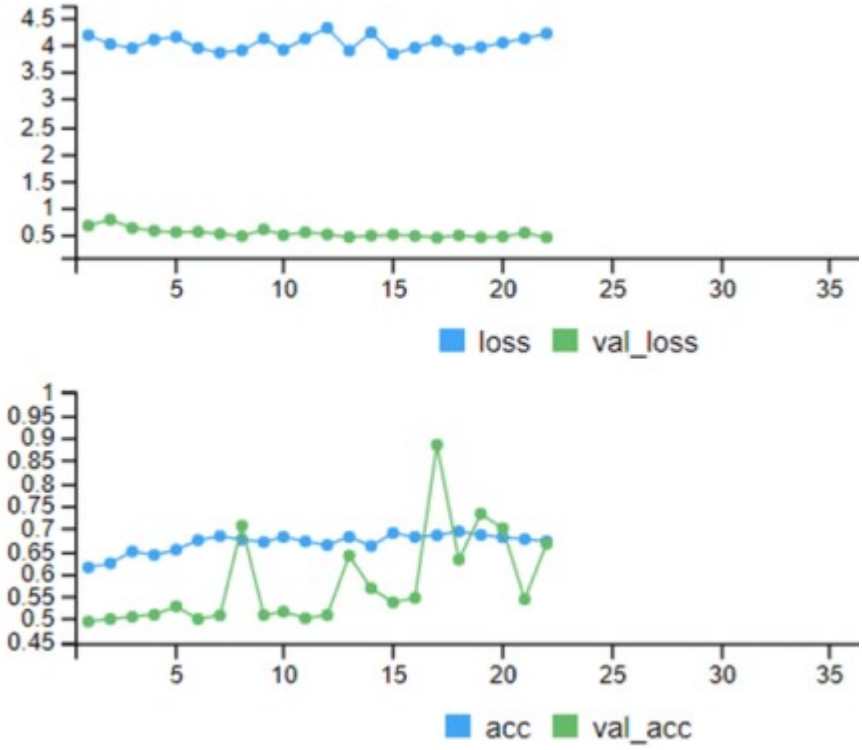
One other important function I have added here is something called early stopping. What this does is it evaluates the loss in the values in the model and if the model starts to overfit, it will hault at that point and output the model accuracy.

```
history_augment %
  fit_generator(
    train_gen_augment,
    steps_per_epoch = 100,
    epochs = 50,
    validation_data = test_gen_augment,
    validation_steps = as.integer(100 / 2),
    callbacks = callback_early_stopping(monitor = "val_loss",
                                        patience=5)
  )
```

The patience parameter shows how long after early stopping has been detected should it wait. This says if it haults at epoch 5 go to 10 and then terminate.

Evaluating the model

This model reported these results:



Checking the model summary the below summary statistics indicate the accuracy of the model across the 23 epochs it completed:

```
summary(history_augment$metrics$acc)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.6135  0.6612  0.6741  0.6680  0.6809  0.6932
```

This is a good improvement and it might be useful. You could spend a couple of weeks finalising and improving this, but for the sake of example, this accuracy is good enough. I will go with the median.

Saving the model

To save the model – I will use the same script in R:

```
model %>% save_model_hdf5("Data/parasite_cells_classification_
augmented.h5")
```

As this took 50+ mins to train and this would only increase with increasing the sample sizes. I will only retrain every now and then. What I will do is load the model from my directory and use this pretrained model to make predictions.

Making predictions

This section will show how to make predictions against our trained model

Loading pretrained model

To load a pretrained model you use the below syntax:

```
model
```

Predicting with new cell scan

The following steps will show what we need to do to convert the image into a format that the model can work with.

```
pred_img
```

There is a bit to this, so I will bullet it out for you:

- pred_img uses index number 100 of the train_parasite list of files. This gives the full path, so we can work with the image
- img_new – this uses Keras's image_load function to take the file path of the pred_img and then adjusts the image to the image shape defined by the variable set at the start of this, but only selects the height and width, not the colour channel
- We then convert the image_to_array by using the numpy array function call from Keras. Essentially, changing it to a array structure
- img_tensor then uses the array_reshape command to take the image we have just converted to an array and adjusts by the full dimensions of the tensor – so 1 image, 130 height, 130 width and 3 colour channels i.e. RGB.
- The last part is to standardise the image between 0 and 1 using the tensor size divided by the max pixels of 255 in an image.
- This is then plotted as a raster image using tensor slicing i.e. img_tensor[1,,,].

The image is now prepared and a plot of the image appears:

Predicting class label and class membership

The last step of the process is to predict the class membership.

The steps of how to achieve are highlighted below:

```
predval %
  dplyr::mutate(Class=case_when(
    V1 == 0 ~ "Parasite Class",
    TRUE ~ "Uninfected"
  )) %>%
  dplyr::rename(ClassPred=V1) %>%
  cbind(predval)

print(pred)
```

The steps are:

- https://keras.rstudio.com/reference/predict_proba.htmlThe **predval** variable holds the probablistic prediction of belonging to that class.
- The **pred** variable uses the keras::predict_classes function to pass in the trained model and make class predictions. Please note – that the class predictions from the flow_from_directory function are assigned 0 – n (n being the number of classes, in this case 1, as it is a binary classification task)
- We then use piping to convert the object to a data.frame. Next, we use mutate to add a

new column and use a case_when statement to set the class label. As indicated in the previous bullet, we assign 0 to the parasite class, as in the flow_images_from_directory parasite comes before the folder uninfected.
- The final step is to rename the class prediction and bind the predval with the pred data frame.

The resultant output shows:

```
ClassPred       Class                          predval
0                       Parasite Class         1.030897e-05
```

Conclusion

There are a number of different ways this network could be improved, some of which have already been highlighted. For a good introductory text to CNNs and Deepe Learning I would recommend *Deep Learning with R*.

As part of my previous role at Draper and Dash I would run three days courses on deep learning to tackle vision, supervised, unsupervised ML and time series forecasting with LSTM networks. I would be happy to be contacted about training any NHS trust or other agency that might need this sort of training.