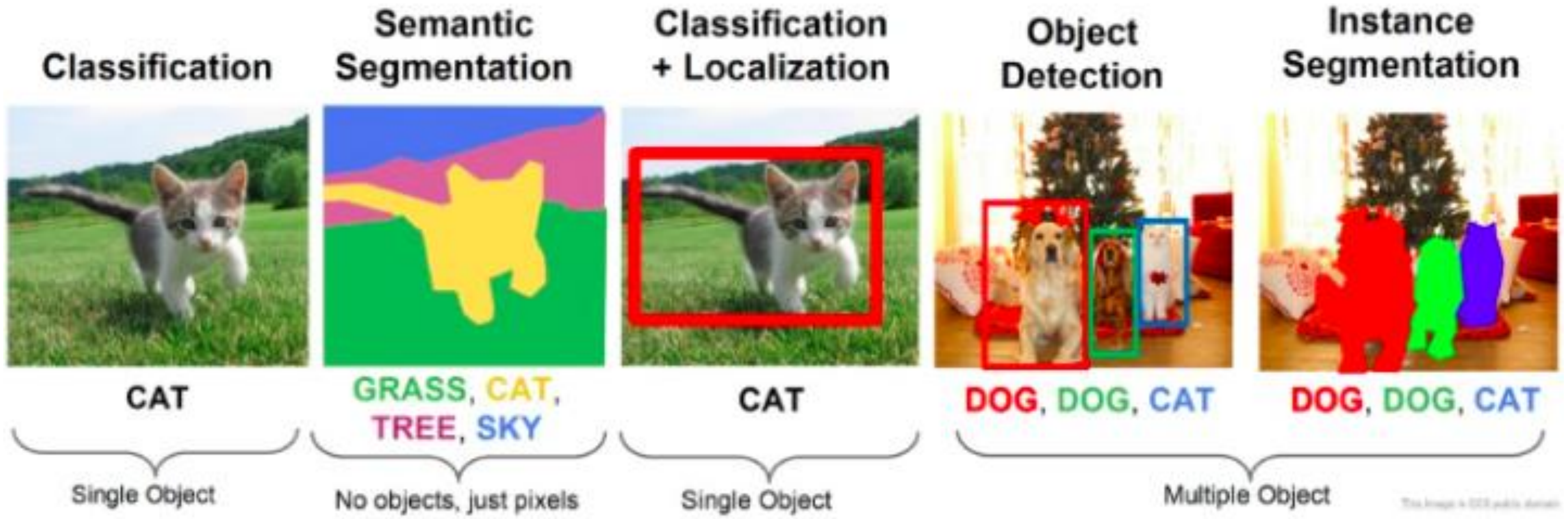
COMPUTER VISION CLASSIFICATION

Algorithms in R to aid clinicians

Gary Hutson
Head of Advanced Analytics
Arden & Gem CSU

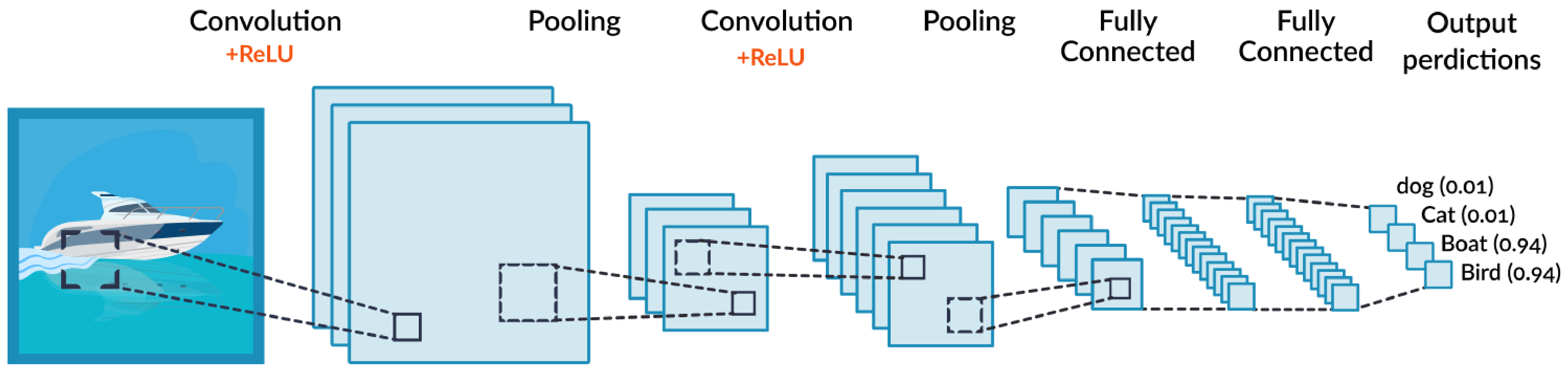
WHAT IS COMPUTER VISION?



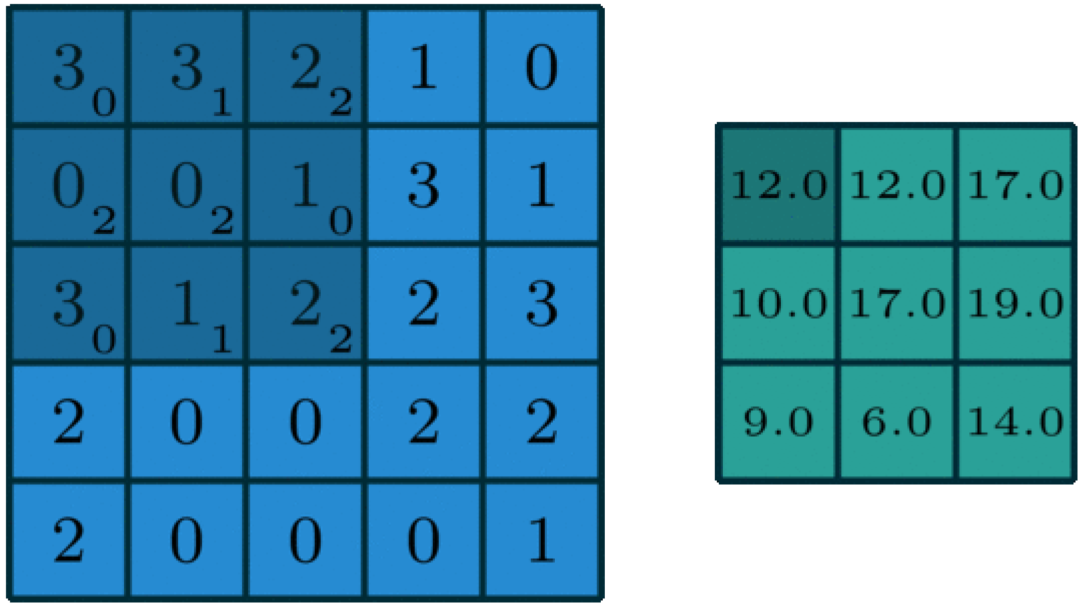
APPLICABILITY TO HEALTHCARE

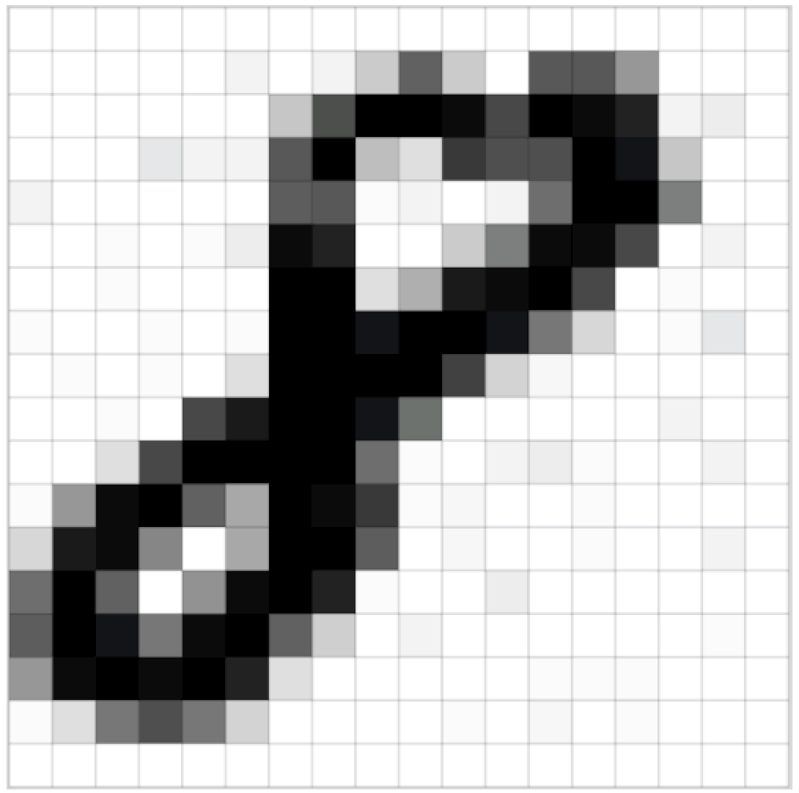
- Aiding clinicians with the identification of problems faster e.g. at Mount Sinai hospital this is being used to detect neurological acute illnesses. This has been aided by utilising 40,000 CT scans from across the health system. This required a "joined up" approach
- More precise diagnosis – CV algorithms can be trained on a massive amount of data that can detect the slightest presence of a condition. A human doctor might easily miss out on, as the algorithm could be trained to detect small anomalies that could be missed. Used alongside clinician skill, this could minimize false positives and improve diagnostic outcomes
- Medical imaging – increases in CV applications have already been an assistance to clinicians and could be used to create binary and multiclass algorithms to check and detect certain conditions
- Image recognition has been used in an example project I worked on as a social distancing detector (profanity here – this was designed in Python). This was a proof of concept idea, more than an application that I would use.

R AND COMPUTER VISION



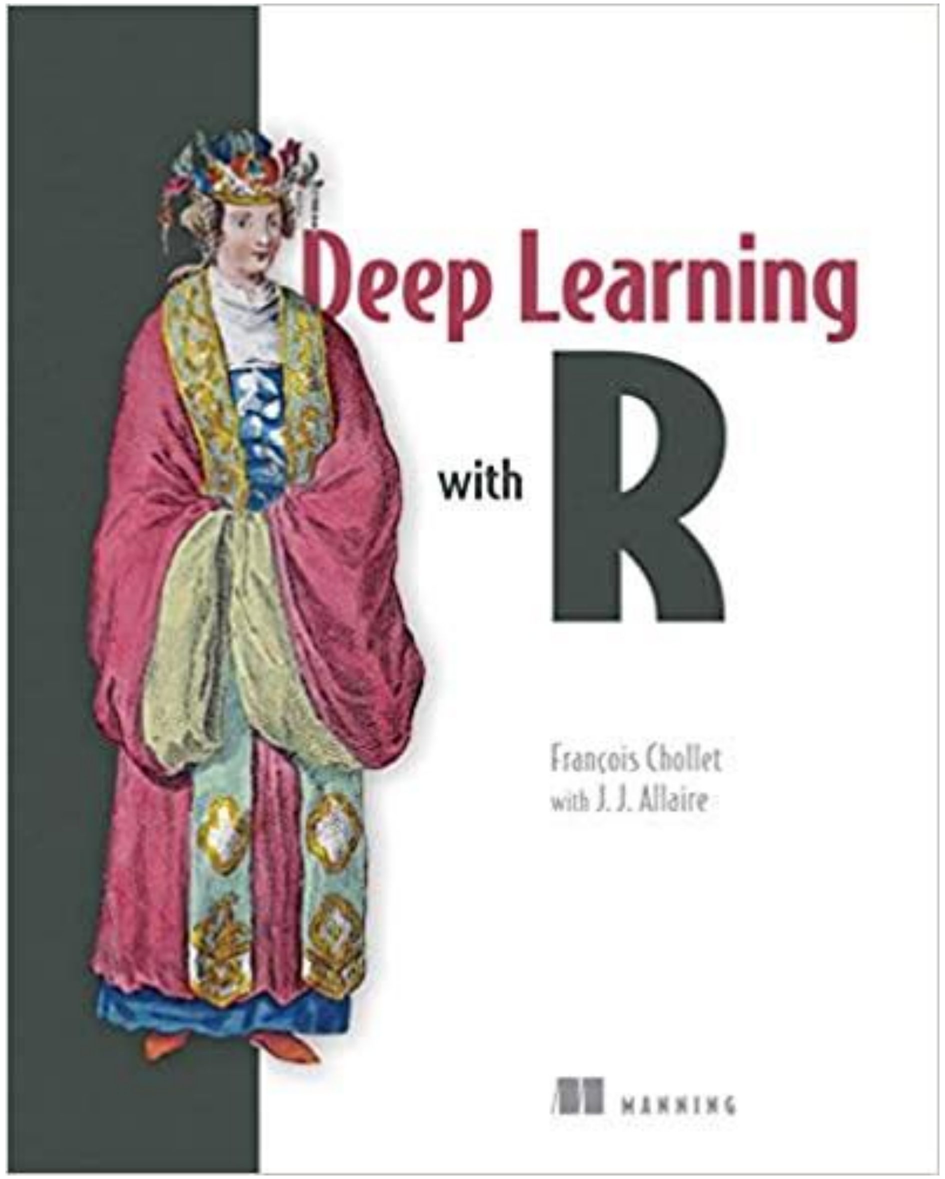
THE POWERHOUSE OF CV – CONVOLUTIONAL NEURAL NETWORKS





GRAPHICAL VERSION

CHECK THIS BOOK OUT


Deep Learning with R

François Chollet with J. J. Allaire

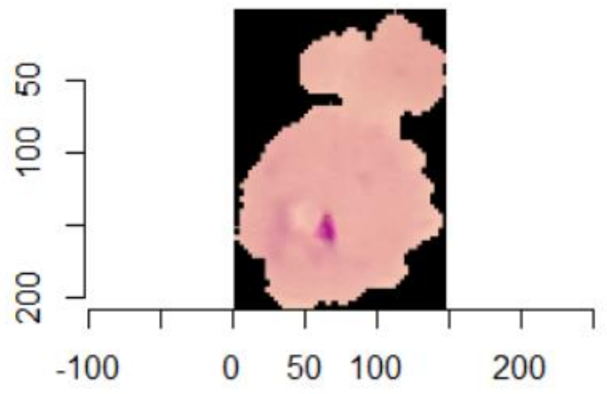
MANNING

BUILDING A CNN IN R

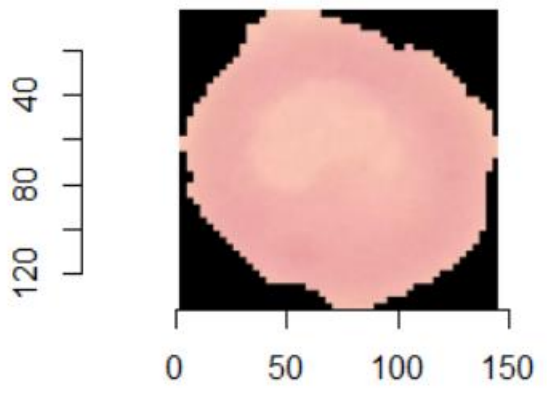
- The article published on my website shows the process of using a Kaggle malaria cell dataset to build a CNN in R for a binary classification task of whether someone's cells have been infected with Malaria or not.
- This type of model could be extended to work with multiple types of x-rays, cancer scans and other types of imaging procedures
- The next few slides will show the process of building the model and the considerations I put in place

EXAMINE THE IMAGES AND BUILD THE KERAS STRUCTURE

- The images need to be stored in a specific way to allow them to flow from a directory. My website gives a full example of the step by step process, and shows the resources used to build the R model in Keras and Tensorflow.
- R functions have been created in the Github to process these images view the show_image() routine



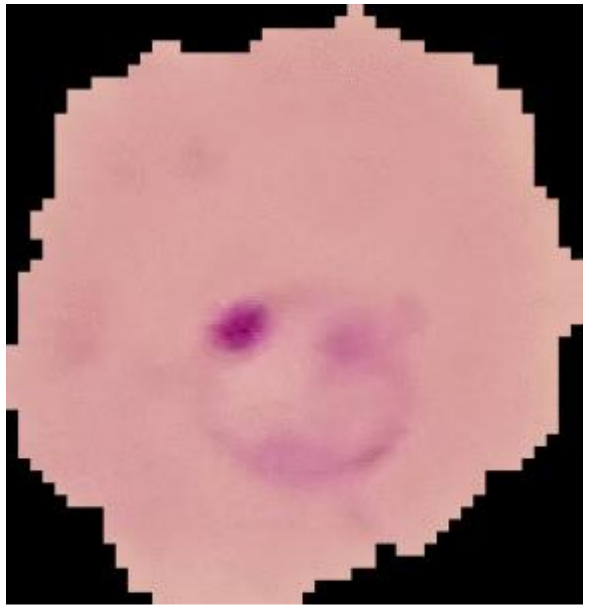
Malaria Parasite Infection

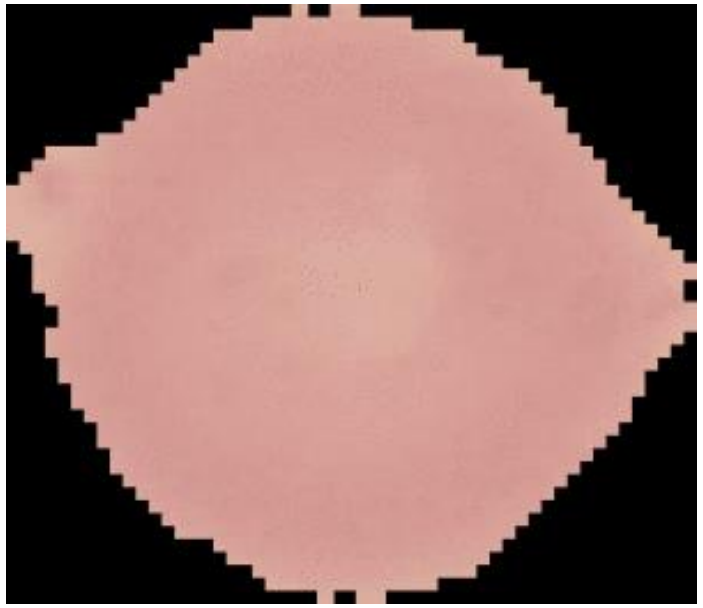


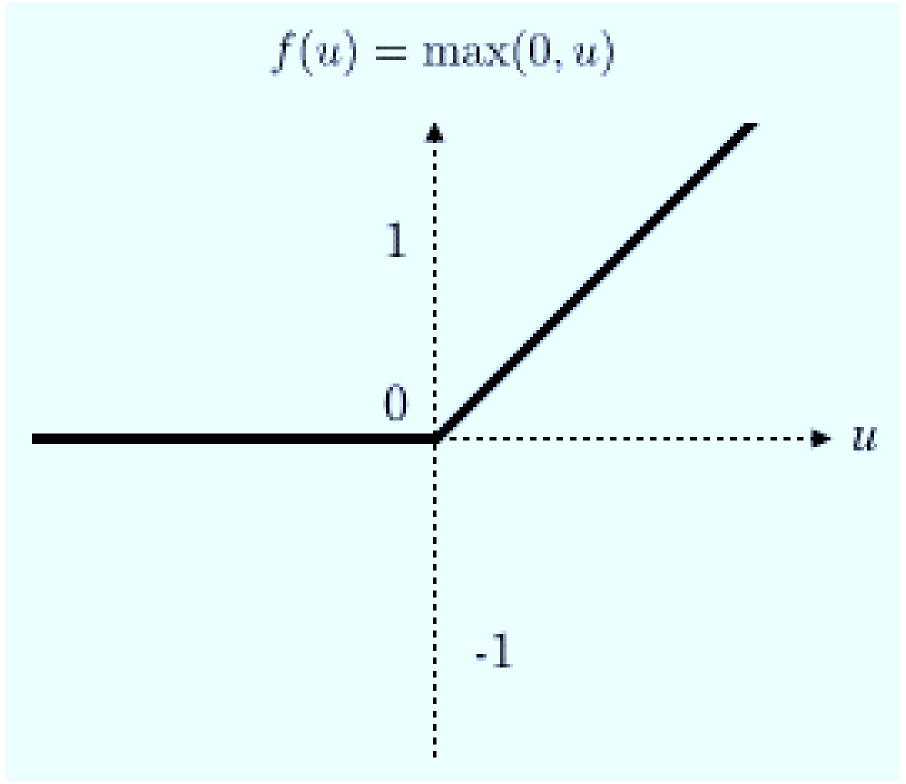
Uninfected

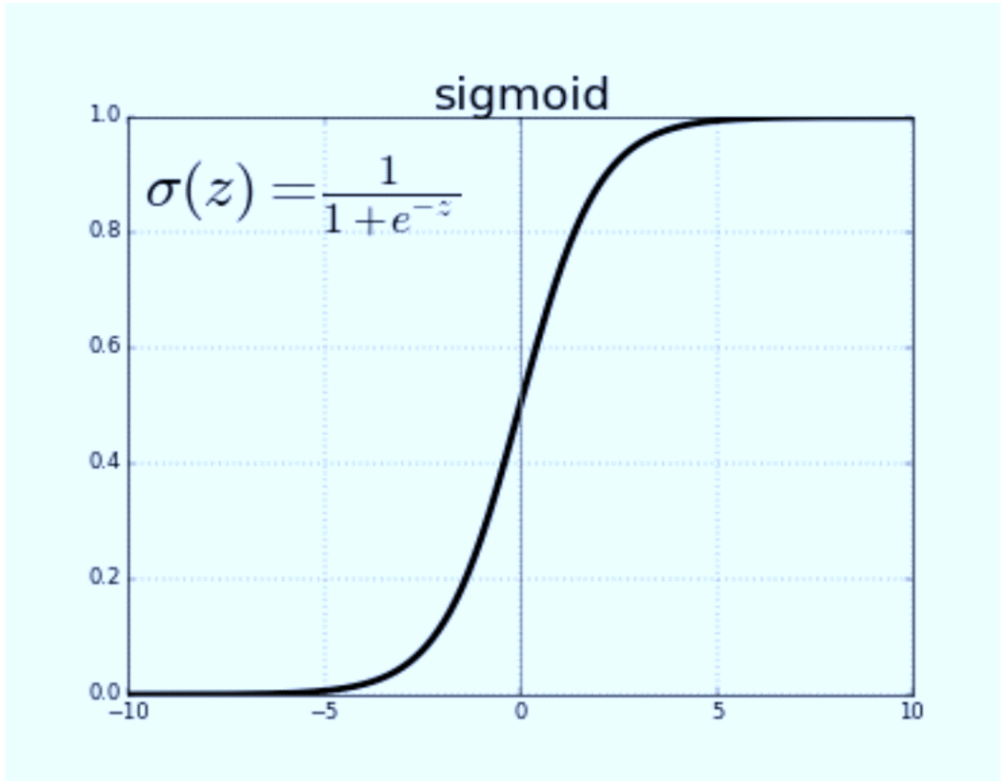
EXAMINE THE IMAGES AND BUILD THE KERAS STRUCTURE

- Inspecting the dimensions of the images was the next step. Here all the images were of a different scale, so needed to be rescaled in Keras.
- Next, I created an animation of the cells seen in the past two slides to check how the first 100 images differed from the 24,000 + images, albeit still a small sample for a CV project.

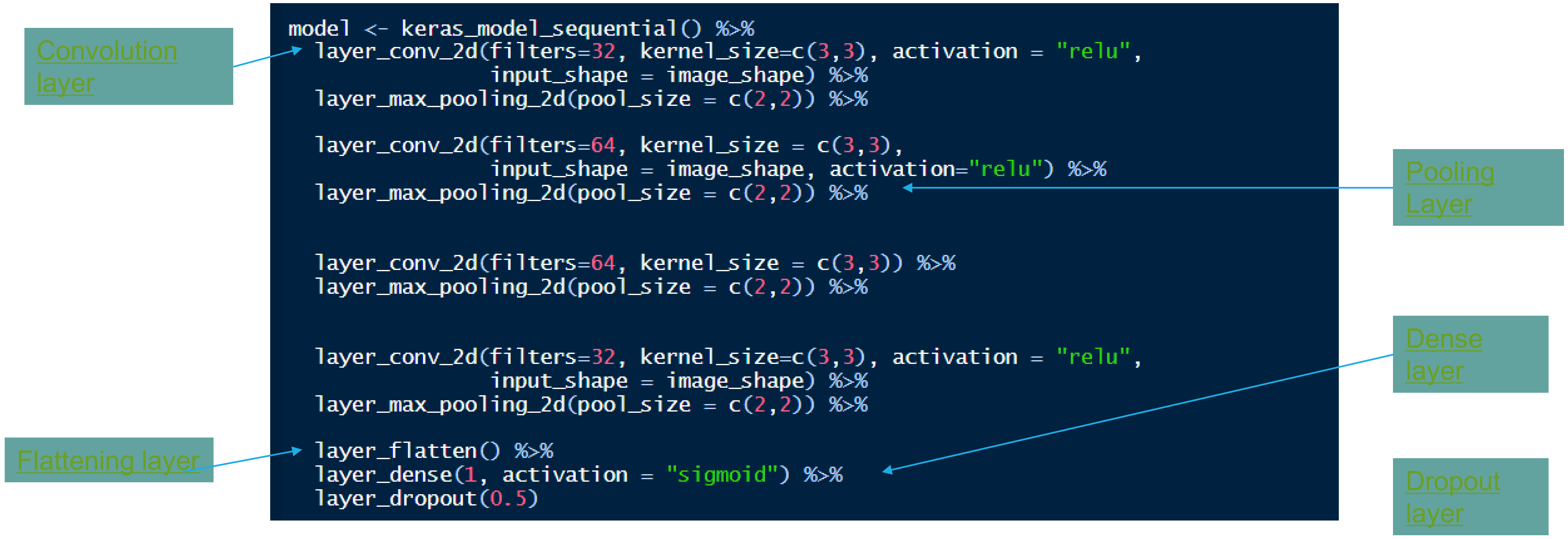

Malaria Parasite Infection


Uninfected





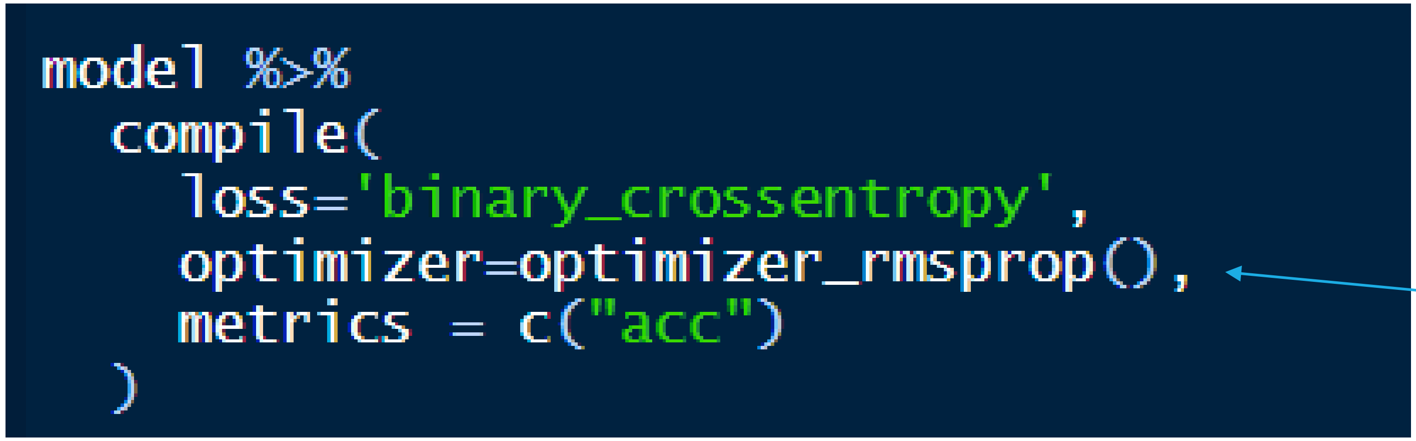
ACTIVATION FUNCTIONS USED



```r
model <- keras_model_sequential() %>%
  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)
```

BUILDING A BASELINE MODEL

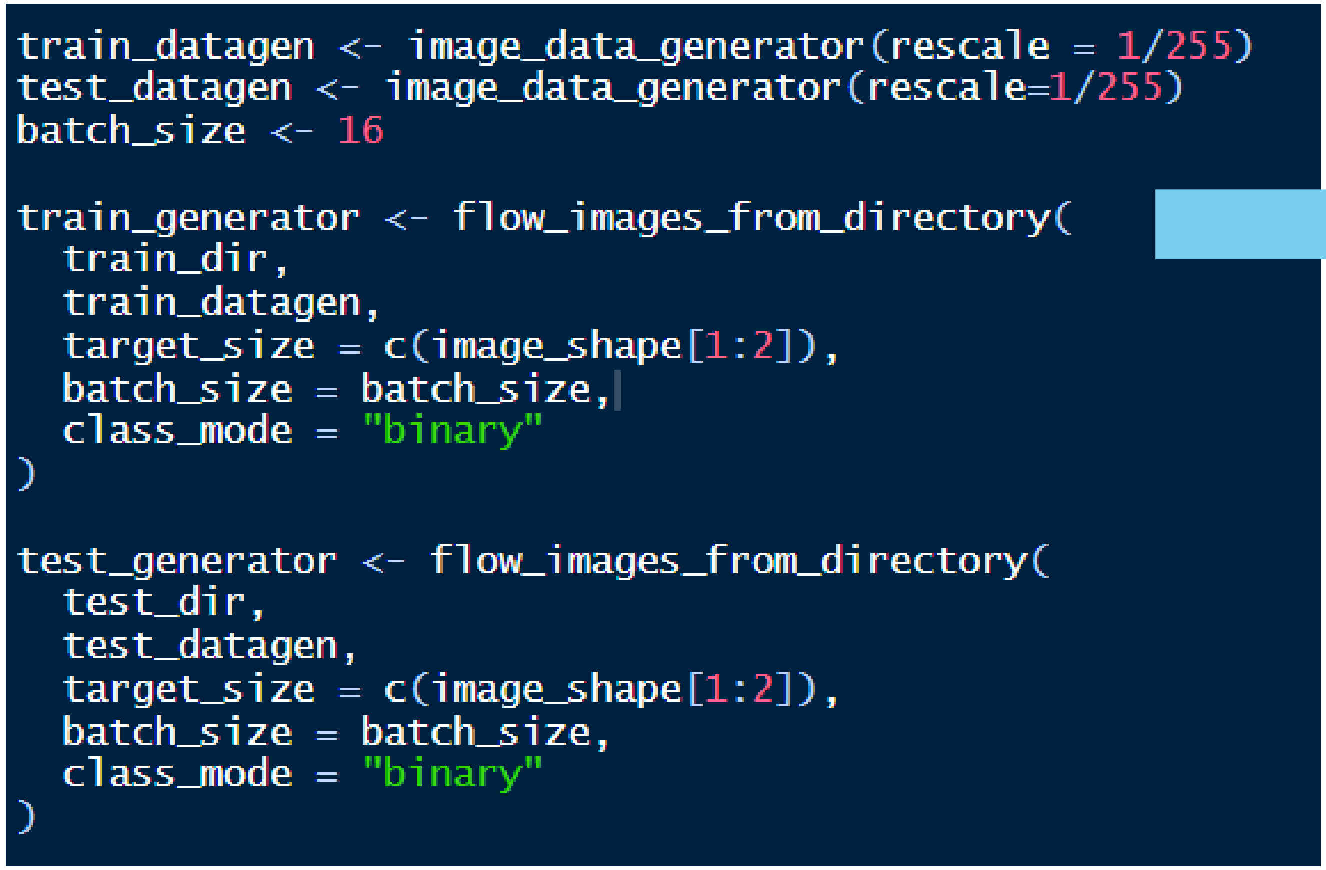
Loss function



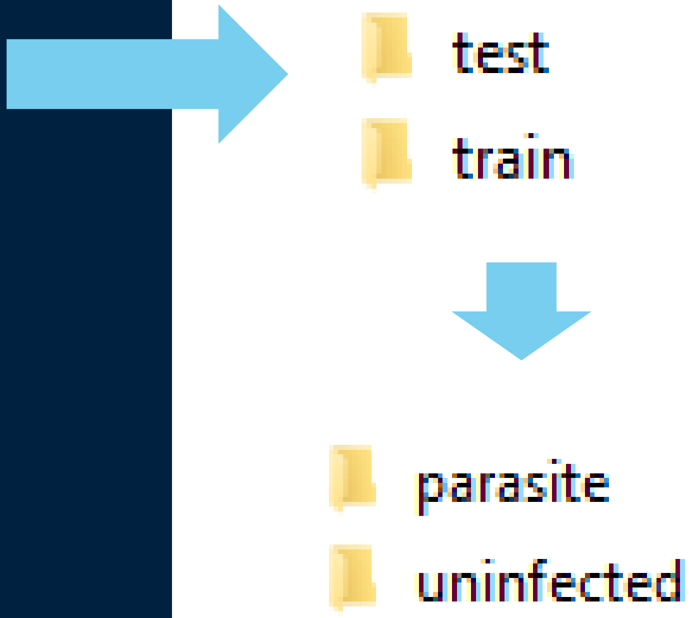
```
model %>%
  compile(
    loss='binary_crossentropy',
    optimizer=optimizer_rmsprop(),
    metrics = c("acc")
  )
```

Optimizer to minimise loss

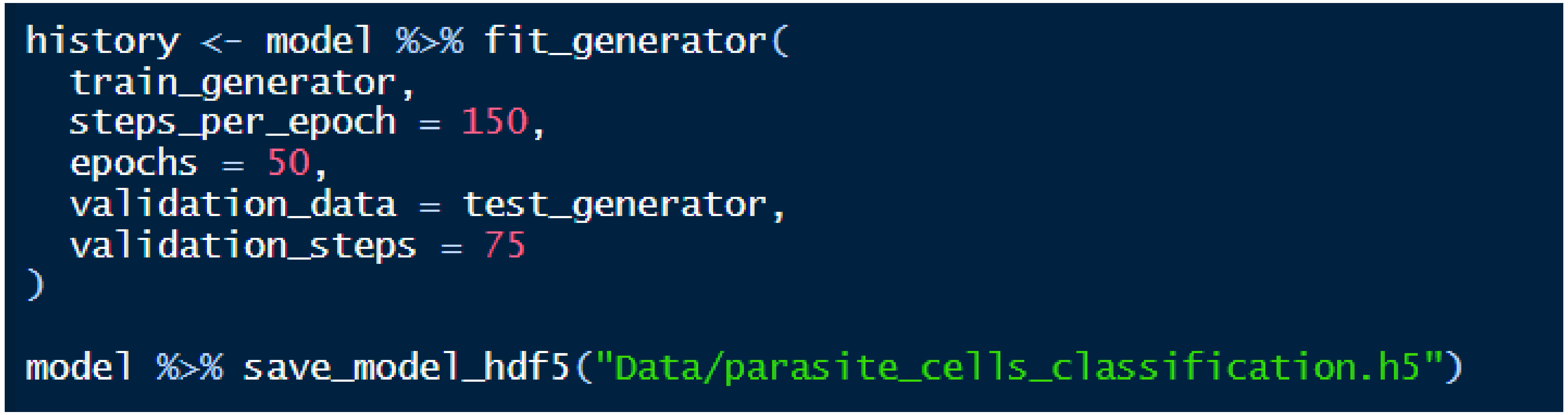
COMPILE MODEL



```
train_datagen <- image_data_generator(rescale = 1/255)
test_datagen <- image_data_generator(rescale=1/255)
batch_size <- 16

train_generator <- flow_images_from_directory(
  train_dir,
  train_datagen,
  target_size = c(image_shape[1:2]),
  batch_size = batch_size,
  class_mode = "binary"
)

test_generator <- flow_images_from_directory(
  test_dir,
  test_datagen,
  target_size = c(image_shape[1:2]),
  batch_size = batch_size,
  class_mode = "binary"
)
```

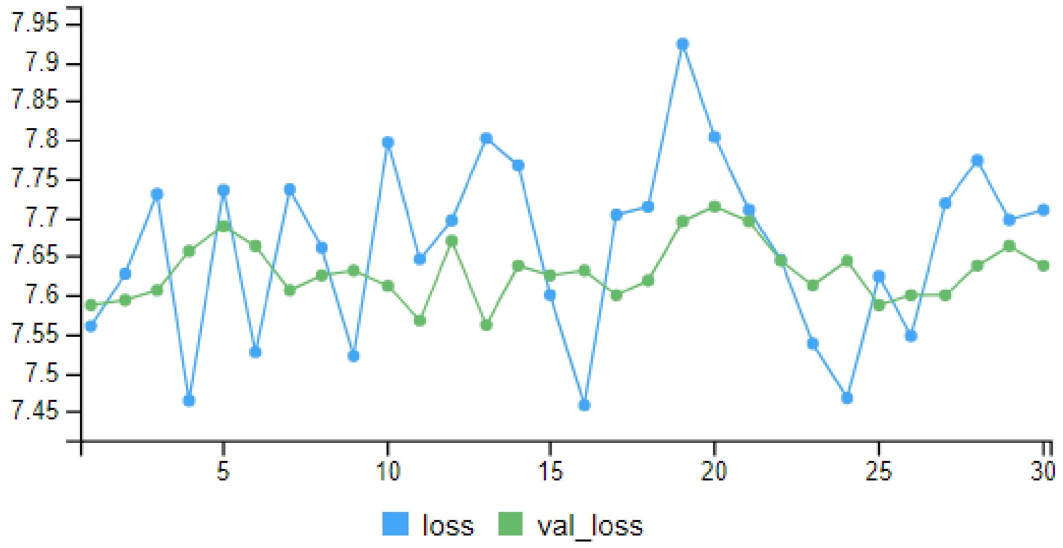


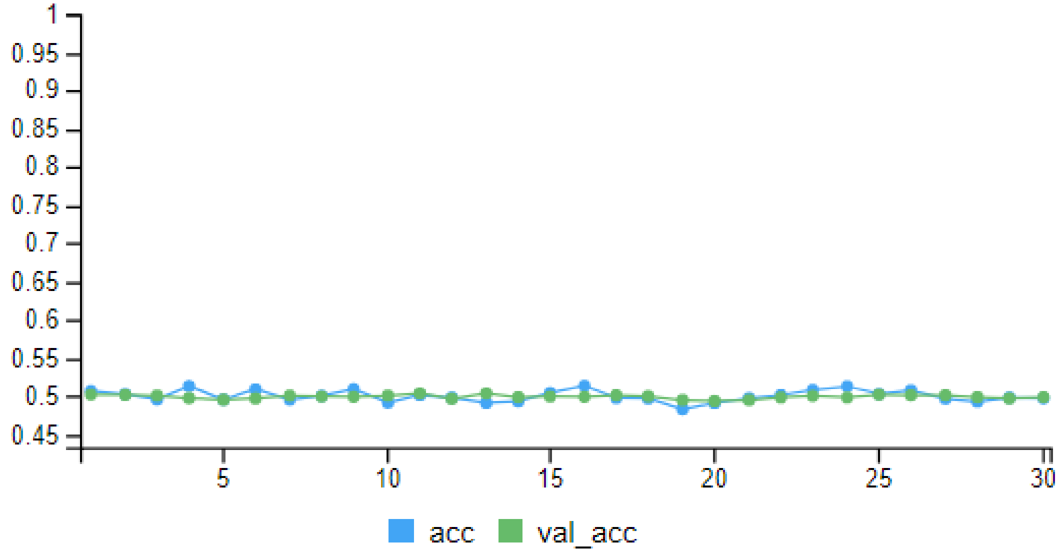
FLOW IMAGES FROM DIRECTORY

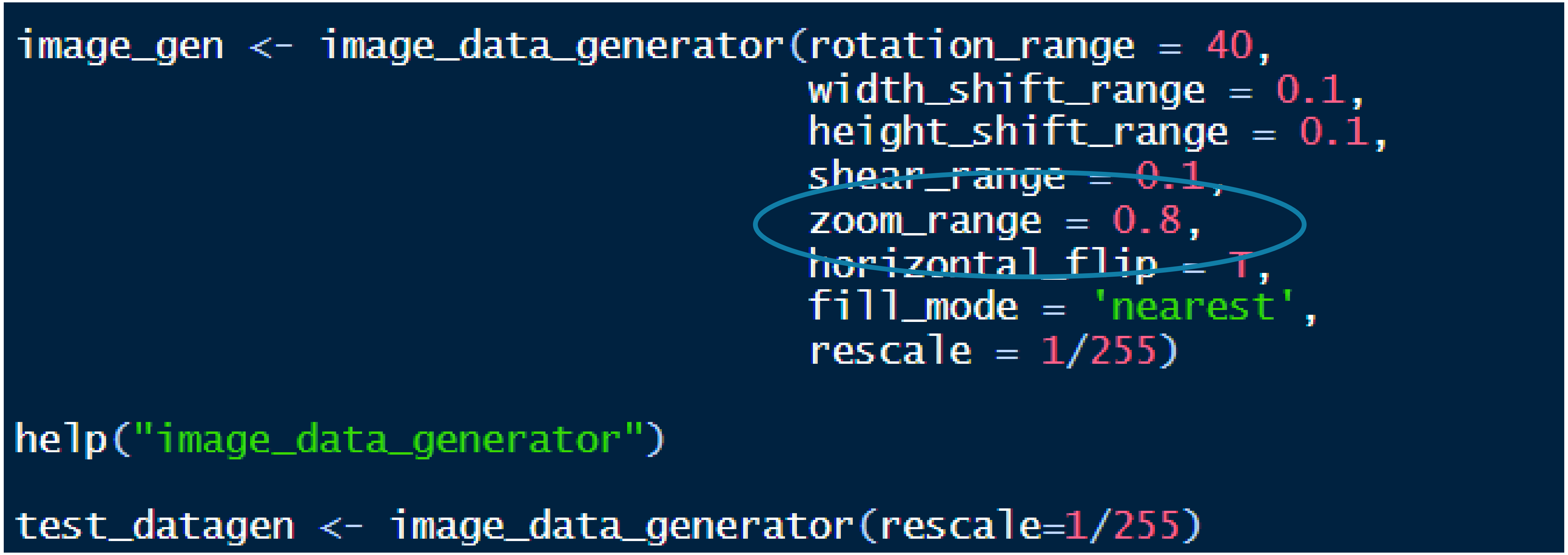

```r
history <- model %>% fit_generator(
  train_generator,
  steps_per_epoch = 150,
  epochs = 50,
  validation_data = test_generator,
  validation_steps = 75
)

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

FIT THE MODEL

KERAS PLOT


7.95
7.9
7.85
7.8
7.75
7.7
7.65
7.6
7.55
7.5
7.45
5
10
15
20
25
30
loss
val_loss

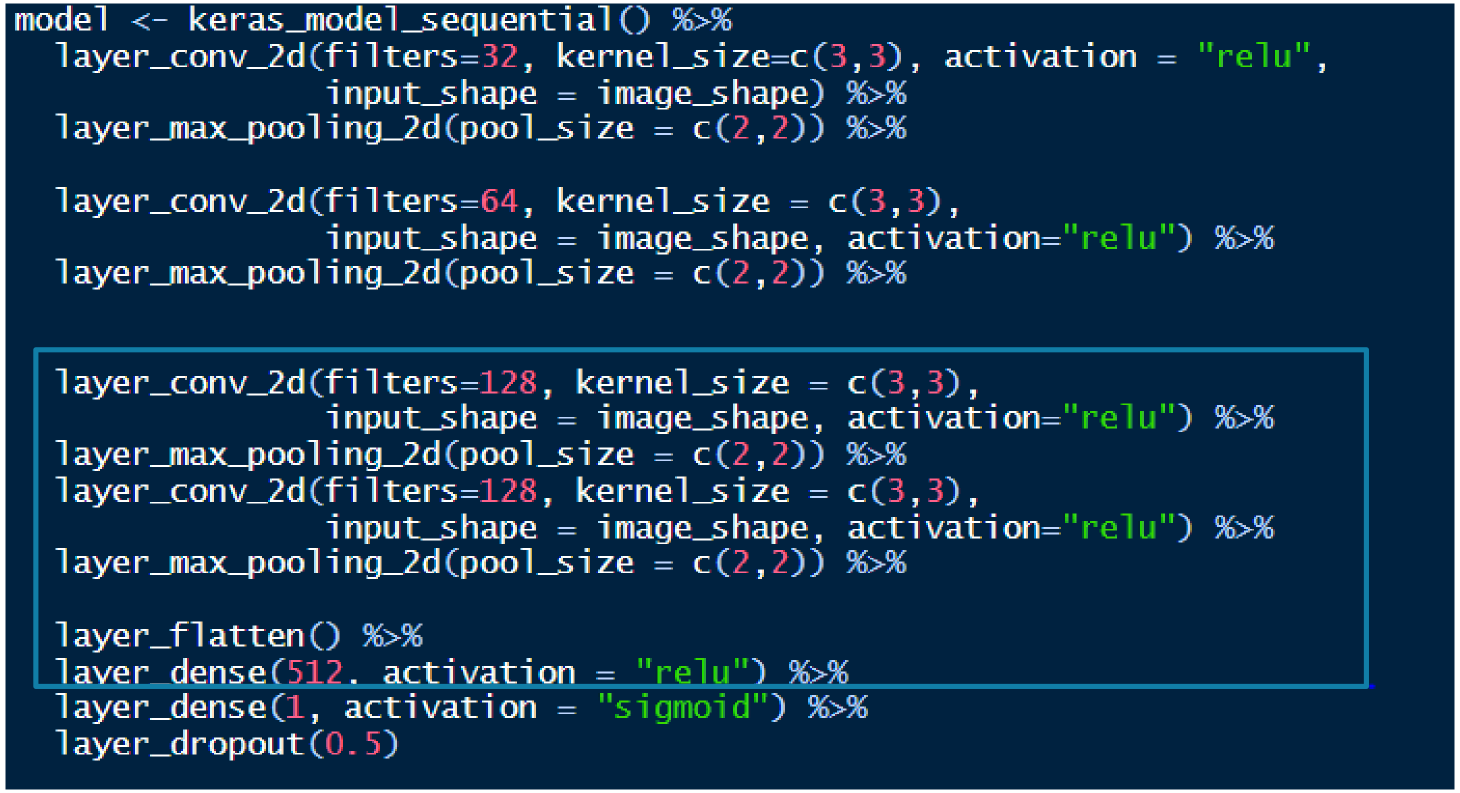

1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
5
10
15
20
25
30
acc
val_acc



```
image_gen <- image_data_generator(rotation_range = 40,
                                  width_shift_range = 0.1,
                                  height_shift_range = 0.1,
                                  shear_range = 0.1,
                                  zoom_range = 0.8,
                                  horizontal_flip = T,
                                  fill_mode = 'nearest',
                                  rescale = 1/255)

help("image_data_generator")

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

DATA AUGMENTATION TO IMPROVE MODEL

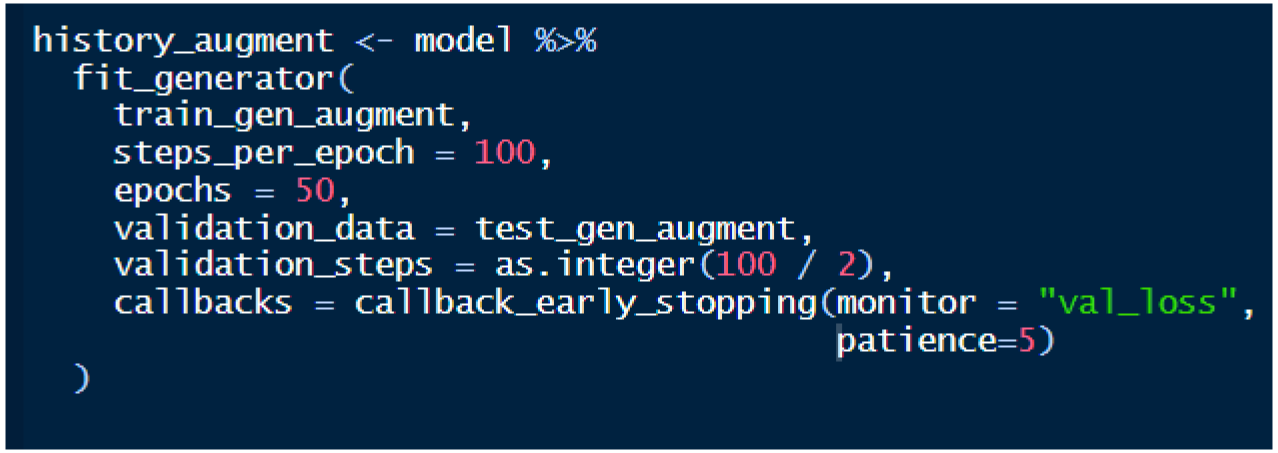
REFORMAT MODEL STRUCTURE


```
model <- keras_model_sequential() %>%
  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)
```

FIT AUGMENTED MODEL

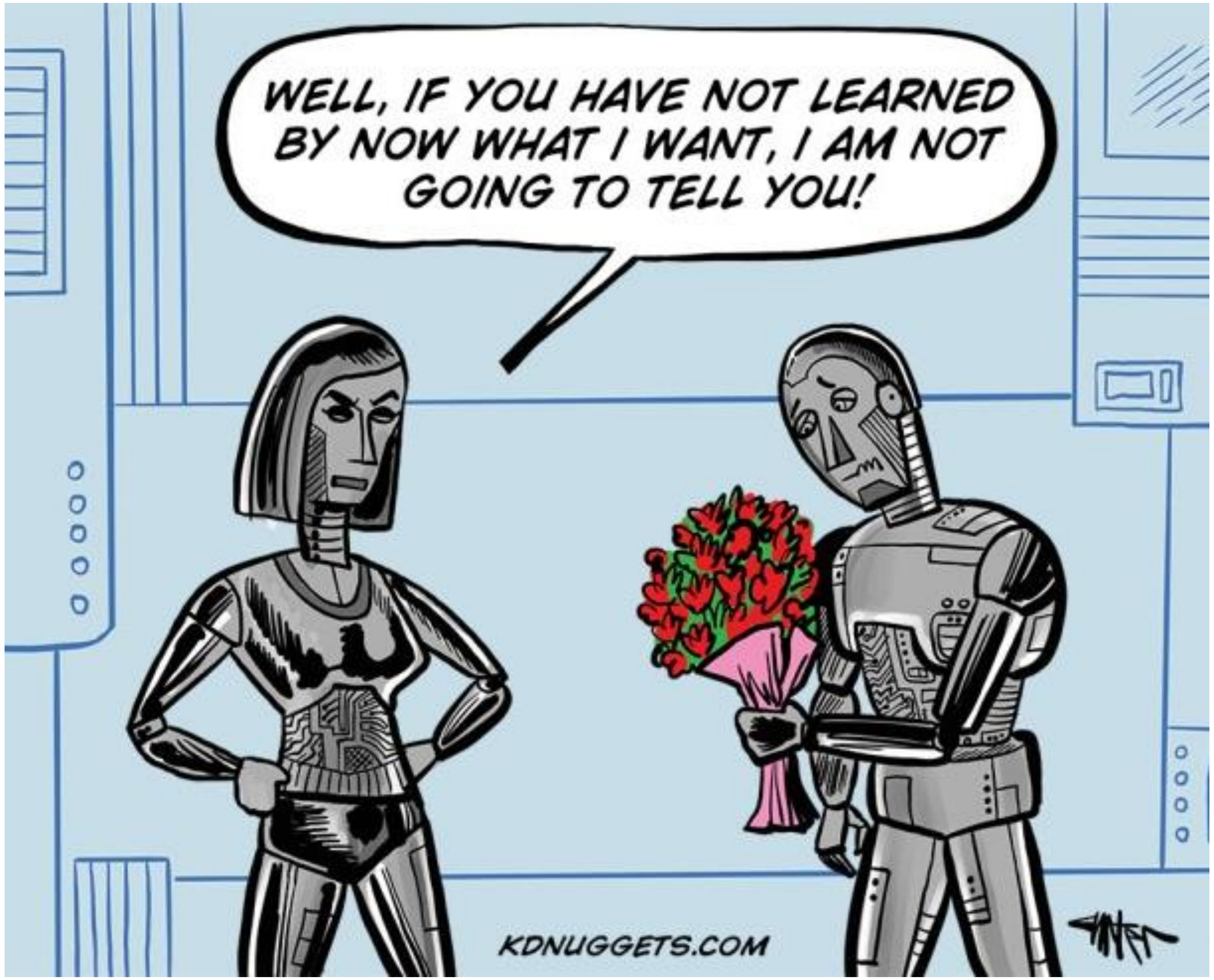
Callbacks were utilized to montor the value loss in the model, as this is the first sign that the model is starting to overfit. Early stopping allows for the model to terminate when it starts to overfit

Model generated from the augmented images – the main addition was the zoom in the previous slide. This allowed for the parasites to be viewed more clearly and thus improved the model performance

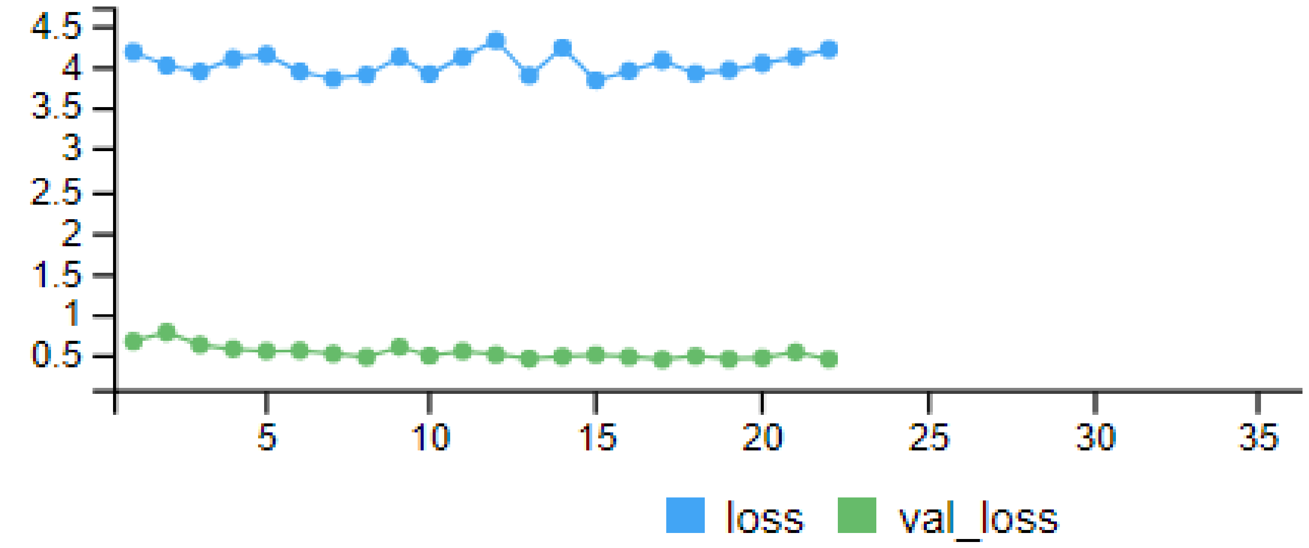


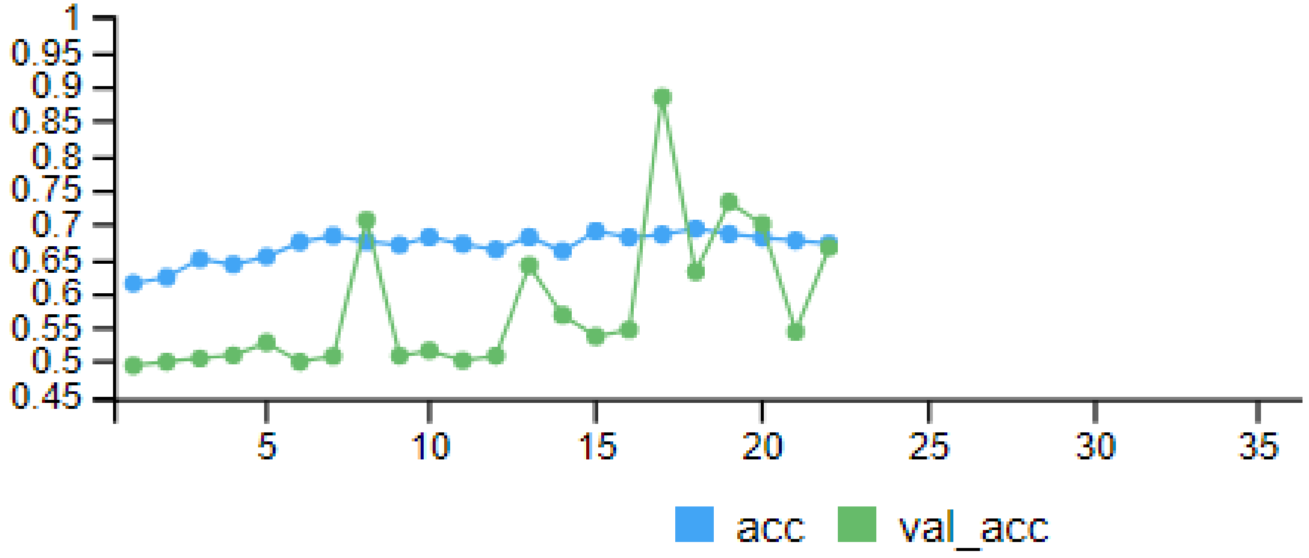
```
history_augment <- model %>%
  fit_generator(
    train_gen_augment,
    steps_per_epoch = 100,
    epochs = 50,
    validation_data = test_gen_augment,
    validation_steps = as.integer(100 / 2),
    callbacks = callback_early_stopping(monitor = "val_loss",
                                     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

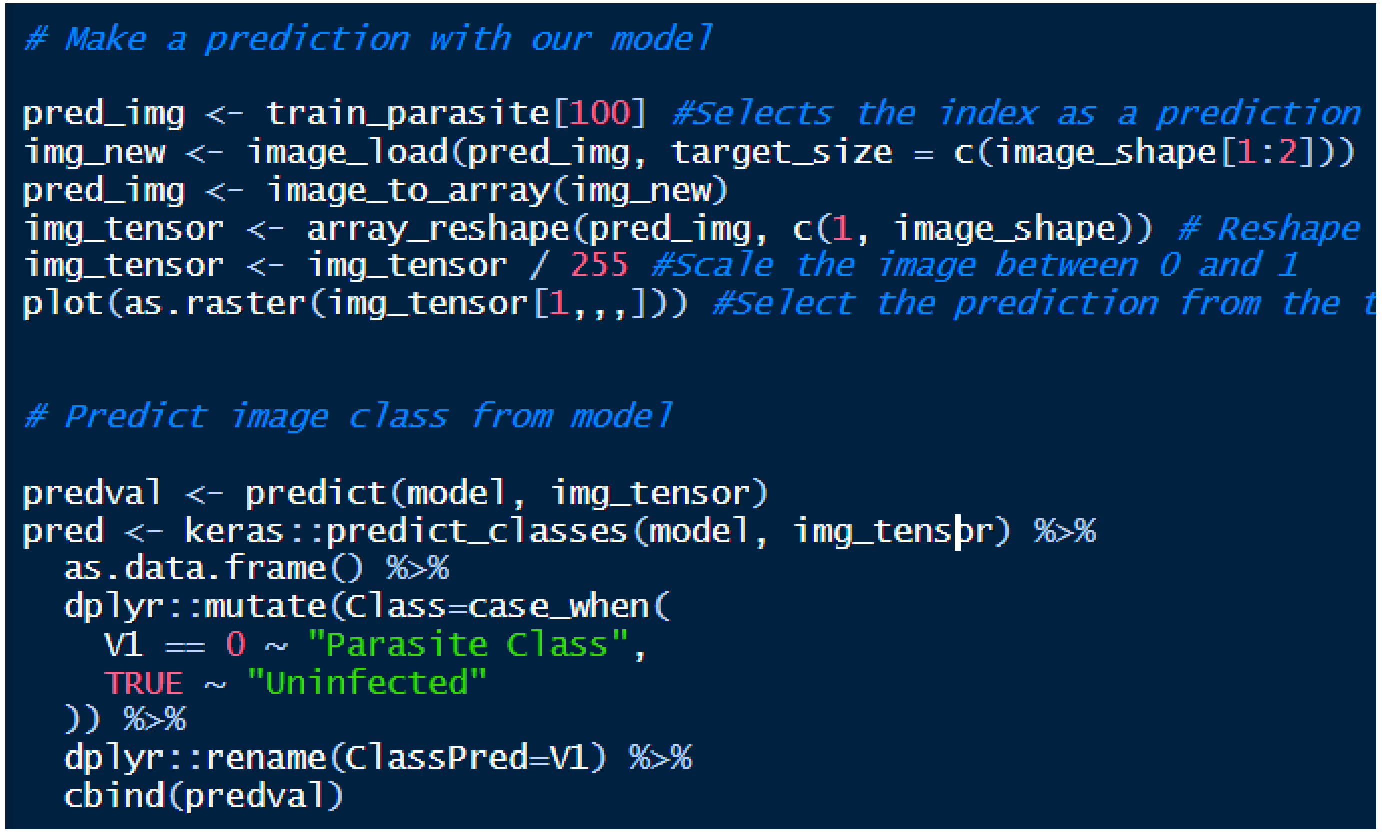
WAIT...WAIT...WAIT...


WELL, IF YOU HAVE NOT LEARNED BY NOW WHAT I WANT, I AM NOT GOING TO TELL YOU!
KDNUGGETS.COM

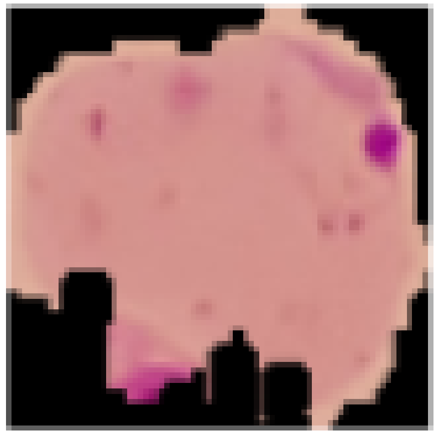
ANALYSE AUGMENTED MODEL RESULTS







```r
# Make a prediction with our model

pred_img <- train_parasite[100] #Selects the index as a prediction
img_new <- image_load(pred_img, target_size = c(image_shape[1:2]))
pred_img <- image_to_array(img_new)
img_tensor <- array_reshape(pred_img, c(1, image_shape)) # Reshape
img_tensor <- img_tensor / 255 #Scale the image between 0 and 1
plot(as.raster(img_tensor[1,,,])) #Select the prediction from the t


# Predict image class from model

predval <- predict(model, img_tensor)
pred <- keras::predict_classes(model, img_tensor) %>%
  as.data.frame() %>%
  dplyr::mutate(Class=case_when(
    V1 == 0 ~ "Parasite Class",
    TRUE ~ "Uninfected"
  )) %>%
  dplyr::rename(ClassPred=V1) %>%
  cbind(predval)
```



ClassPred	Class	predval
0	Parasite Class	1.030897e-05

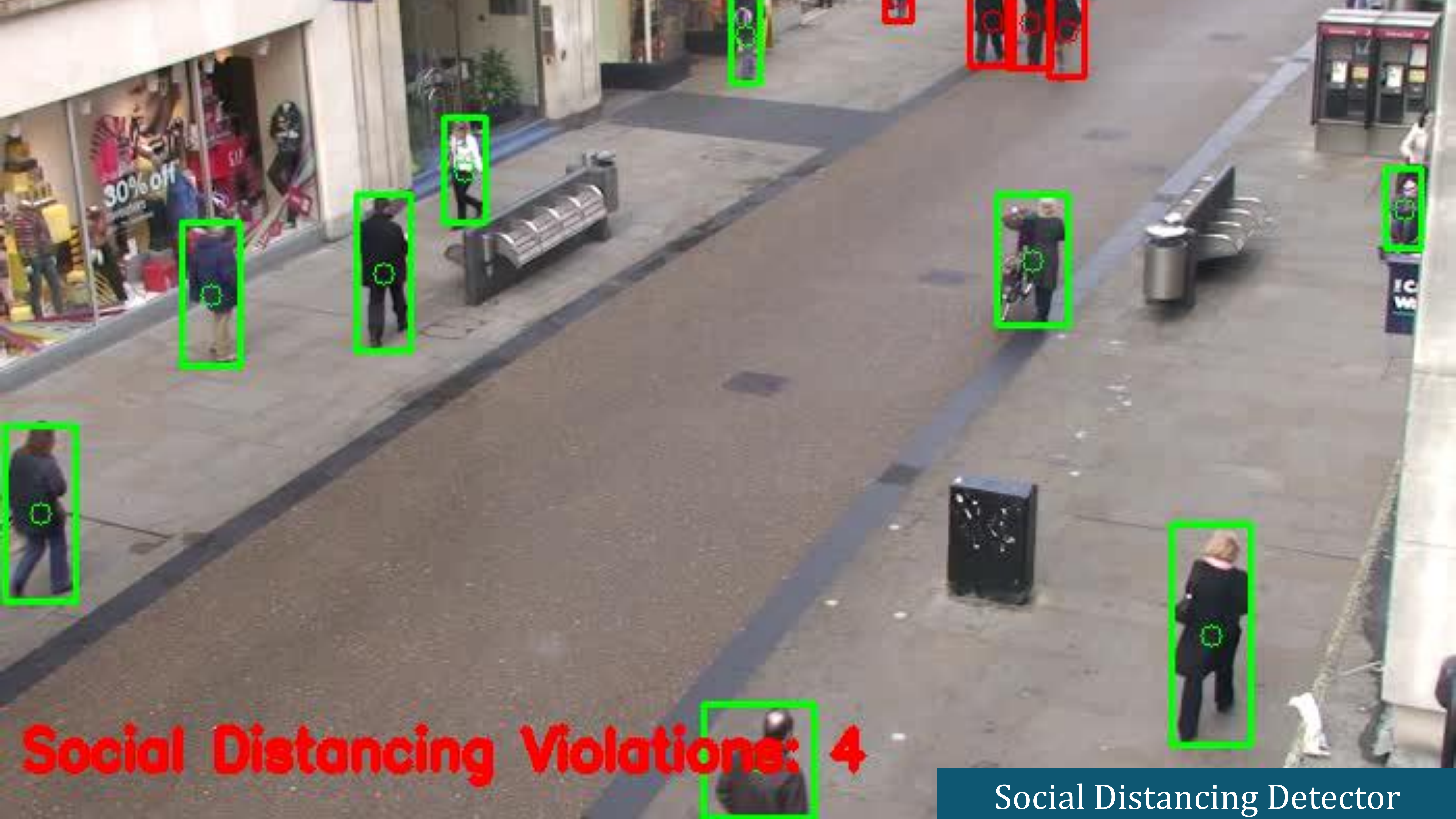
PREDICTING WITH OUR MODEL

FURTHER MODEL IMPROVEMENTS

- There are not any pretrained networks available for this problem, making it different from a lot of the classification tasks. This needs to serve as a “call to arms” to an NHS imaging repository for various scans to improve the model trainability and to rival the **ImageNet** popular pretrained network – containing millions of images
- Different model architectures and layering, such as the Inception network, RESNET, MobileNet, ResNet50, VGG16 and 19. Transformer networks are the new trend in NLP and are being applied to CV as we speak – watch this space: https://openreview.net/forum?id=YicbFdNTTy.
- Use of leaky-Relus might improve performance and accuracy slightly, allowing for some non-zero values
- Image adjustments, scaling, etc. may allow the convolutional nets to detect the images more clearly, as shown with the adjustment of the zoom.

CV – THE R VS PYTHON STRUGGLE

- With the addition of Tensorflow to R – this can easily handle classification tasks, utilising the Convolutional Neural Network architecture, but it does not have the packages to support advanced:
  - Object detection – Open CV works much better in Python and can be integrated to the Raspberry Pi (example on next slide)
  - Facial recognition, again Python wins on this front, as R's VideoPlayR has only some of what Python has at its disposal
  - Video streaming and capture can be done on both platforms but is still more fluid in Python – sorry!



Social Distancing Detector

Questions?