CSDA 1040: Assignment 3 - Group 4

Jose German, Anjana Pradeep Kumar, Anupama Radhakrishnan Kowsalya, and Xenel Nazar

07/18/2020

1.0 Abstract

Human Interaction Proof (HIP) systems are currently deployed to restrict spam bots. Deep Learning implementations, like Convolutional Neural Networks, on images used on HIP systems are effective at classifying images based on the data provided to it. HIP systems would need to continually evolve to further improve security measures.

2.0 Introduction

Various web services deploy security measures, like Human Interaction Proof (HIP) systems that are designed to be challenges easily solved by Humans, but difficult for computers or bots to solve (Chellapilla, Larson, Simard, & Czerwinski, 2018).

One popular HIP implementation is the CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) program, which was was coined in 2000 by Luis von Ahn, Manuel Blum, Nicholas Hopper, and John Langford of Carnegie Mellon University. CAPTCHAs can be applied in multiple cases, including preventing comments spam in blogs, protecting website registration, protecting email addresses from scrapers, online polls, preventing dictionary attacks in password systems, blocking search engine bots, as well blocking email worms and spam. The ESP-PIX Captcha script implementation is based on image recognition, where users are asked to review and recognize similar or common set of images (Carnegie Mellon University, 2010).

The recent advancement and popularity of deep learning algorithms point to a question, if models can be developed that would be able to circumvent HIP systems like the ESP-PIX Captcha implementation.

3.0 Objective

The objective of this project is to develop a model based on deep learning fundamentals and evaluate its effectiveness on images used on CAPTCHA or HIP systems.

4.0 Data Understanding

4.1 About the Data

The ASIRRA (Animal Species Image Recognition for Restricting Access) dataset currently hosted on Kaggle and by Microsoft is a collection of cat and dog images that was intended for use as a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) or a HIP (Human Interactive Proof) (Cukierski, 2014; Microsoft, 2017).

The data loaded on Kaggle is a subset of the original dataset provided by Petfinder.com to the Microsoft Research team, that consisted of three million images of cats and dogs (Cukierski, 2014). Petfinder.com works with nearly 11,000 animal shelters and adoption organizations in the United States, Canada, and Mexico to provide an online database of adoptable pets (Petfinder.com, 2018).

4.2 Import Libraries

```
# Install EBImage package if needed
#install.packages("BiocManager")
#BiocManager::install("EBImage")
```

```
# Install Libraries
library(keras)
library(EBImage)
library(stringr)
library(pbapply)
library(tensorflow)
```

4.3 Import Data

Get working directory

```
getwd()
```

```
## [1] "/Users/xen/Documents/CSDA 1040/ASSIGNMENT3"
```

Get directory for training and test images

```
# Get working directory
original_dataset_dir_train <- "/Users/xen/Documents/CSDA 1040/ASSIGNMENT3/train"
original_dataset_dir_test <- "/Users/xen/Documents/CSDA 1040/ASSIGNMENT3/test1"
```

5.0 Data Exploration and Preparation

5.1 Data Overview

Verify number of images

```
# Verify number of images
cat("Total Training Images:", length(list.files(original_dataset_dir_train)), "\n")
```

```
## Total Training Images: 25000
```

```
cat("Total Test Images:", length(list.files(original_dataset_dir_test)), "\n")
```

```
## Total Test Images: 12500
```

With over 25,000 training and 12,500 test images, we may have to subset the data to an easily working number of images.

View sample cat image

```
# View sample cat image
example_cat_image <- readImage(file.path(original_dataset_dir_train, "cat.1.jpg"))
display(example_cat_image)
```

View sample dog image

```
# View sample dog image
example_dog_image <- readImage(file.path(original_dataset_dir_train, "dog.1.jpg"))
display(example_dog_image)
```

```
# View sample test image
example_test_image <- readImage(file.path(original_dataset_dir_test, "1.jpg"))
display(example_test_image)
```

The training images have been labeled the images as either “cat” or “dog”, while the test images have not.

5.2 Create Working Directory for Subset of Data

Note, code from the Deep Learning with R book was referenced for this project (Chollet & Allaire, 2018).

```
# Create Working Directory
working_dir <- "/Users/xen/Documents/CSDA 1040/ASSIGNMENT3/working_dir"
dir.create(working_dir)
```

```
# Create Training Folder
train_dir <- file.path(working_dir, "train")
dir.create(train_dir)
# Create Validation Folder
validation_dir <- file.path(working_dir, "validation")
dir.create(validation_dir)
# Create Test Folder
test_dir <- file.path(working_dir, "test")
dir.create(test_dir)
```

```
# Create Cats and Dogs Subfolders for Training
train_cats_dir <- file.path(train_dir, "cats")
dir.create(train_cats_dir)
train_dogs_dir <- file.path(train_dir, "dogs")
dir.create(train_dogs_dir)
# Create Cats and Dogs Subfolders for Validation
validation_cats_dir <- file.path(validation_dir, "cats")
dir.create(validation_cats_dir)
validation_dogs_dir <- file.path(validation_dir, "dogs")
dir.create(validation_dogs_dir)
```

Place images in folders

```
# Set 1000 Cat images for Training
fnames <- paste0("cat.", 1:1000, ".jpg")
file.copy(file.path(original_dataset_dir_train, fnames),
          file.path(train_cats_dir))
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [43] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [57] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [71] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [85] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [99] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [113] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [127] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [141] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [155] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [169] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [183] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [197] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [211] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [225] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [239] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [253] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [267] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [281] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [295] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [309] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [323] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [337] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [351] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [365] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [379] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [393] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [407] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [421] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [435] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [449] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [463] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [477] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [491] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [505] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [519] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [533] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [547] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [561] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [575] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [589] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [603] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [617] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [631] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [645] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [659] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [673] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [687] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [701] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [715] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [729] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
##  [743] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [757] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [771] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [785] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [799] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [813] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [827] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [841] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [855] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [869] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [883] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [897] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [911] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [925] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [939] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [953] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [967] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [981] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [995] TRUE TRUE TRUE TRUE TRUE TRUE
```

```
# Set 500 Cat images for Validation
fnames <- paste0("cat.", 1001:1500, ".jpg")
invisible(file.copy(file.path(original_dataset_dir_train, fnames),
          file.path(validation_cats_dir)))

# Set 1000 Dog images for Training
fnames <- paste0("dog.", 1:1000, ".jpg")
invisible(file.copy(file.path(original_dataset_dir_train, fnames),
          file.path(train_dogs_dir)))

# Set 500 Dog images for Validation
fnames <- paste0("dog.", 1001:1500, ".jpg")
invisible(file.copy(file.path(original_dataset_dir_train, fnames),
          file.path(validation_dogs_dir)))

# Set 500 images for Testing
fnames <- paste0(1501:2000, ".jpg")
invisible(file.copy(file.path(original_dataset_dir_test, fnames),
          file.path(test_dir)))
```

```
# Verify number of Cat images
cat("Total Training Cat Images:", length(list.files(train_cats_dir)), "\n")
```

```
## Total Training Cat Images: 1000
```

```
cat("Total Validation Cat Images:", length(list.files(validation_cats_dir)), "\n")
```

```
## Total Validation Cat Images: 500
```

```
# Verify number of Dog images
cat("Total Training Dog Images:", length(list.files(train_dogs_dir)), "\n")
```

```
## Total Training Dog Images: 1000
```

```
cat("Total Validation Dog Images:", length(list.files(validation_dogs_dir )), "\n")
```

```
## Total Validation Dog Images: 500
```

```
# Verify number of Test images
cat("Total Test Images:", length(list.files(test_dir)), "\n")
```

```
## Total Test Images: 500
```

6.0 Modeling and Evaluation

6.1 Model Setup

```
# Model Setup
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(150, 150, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

The following are additional steps that need to be defined before the model is trained. The loss function measures how accurate the model is during training, the optimizer is how the model is updated based on the data it receives, and lastly is the accuracy metrics which monitors the training steps (Allaire, 2020).

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

```
# Model Summary
summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## conv2d (Conv2D)              (None, 148, 148, 32)      896
## _________________________________________________________________________
## max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
## _________________________________________________________________________
## conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
## _________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D) (None, 36, 36, 64)      0
## _________________________________________________________________________
## conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
## _________________________________________________________________________
## max_pooling2d_2 (MaxPooling2D) (None, 17, 17, 128)     0
## _________________________________________________________________________
## conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
## _________________________________________________________________________
## max_pooling2d_3 (MaxPooling2D) (None, 7, 7, 128)       0
## _________________________________________________________________________
## flatten (Flatten)            (None, 6272)              0
## _________________________________________________________________________
## dense (Dense)                (None, 512)               3211776
## _________________________________________________________________________
## dense_1 (Dense)              (None, 1)                 513
## =========================================================================
## Total params: 3,453,121
## Trainable params: 3,453,121
## Non-trainable params: 0
## _________________________________________________________________________
```

6.2 Data Preprocessing

```
# Rescale Images by 1/255
train_datagen <- image_data_generator(rescale = 1/255)
validation_datagen <- image_data_generator(rescale = 1/255)
```

```
train_generator <- flow_images_from_directory(
  # Target Directory
  train_dir,
  train_datagen,
  # Resize images to 150 X 150
  target_size = c(150, 150),
  # 20 images per batch
  batch_size = 20,
  class_mode = "binary"
)

validation_generator <- flow_images_from_directory(
  # Target Directory
  validation_dir,
  validation_datagen,
  # Resize images to 150 X 150
  target_size = c(150, 150),
  # 20 images per batch
  batch_size = 20,
  class_mode = "binary"
)
```

6.3 Model Fitting

Fitting the model using the batch generator.On the generator we need to define the steps per epoch for the training and validation images. For the training images we will set the steps as 100 (# of Training Images / Batch Size) and 50 for validation (# of Validation Images / Batch Size)

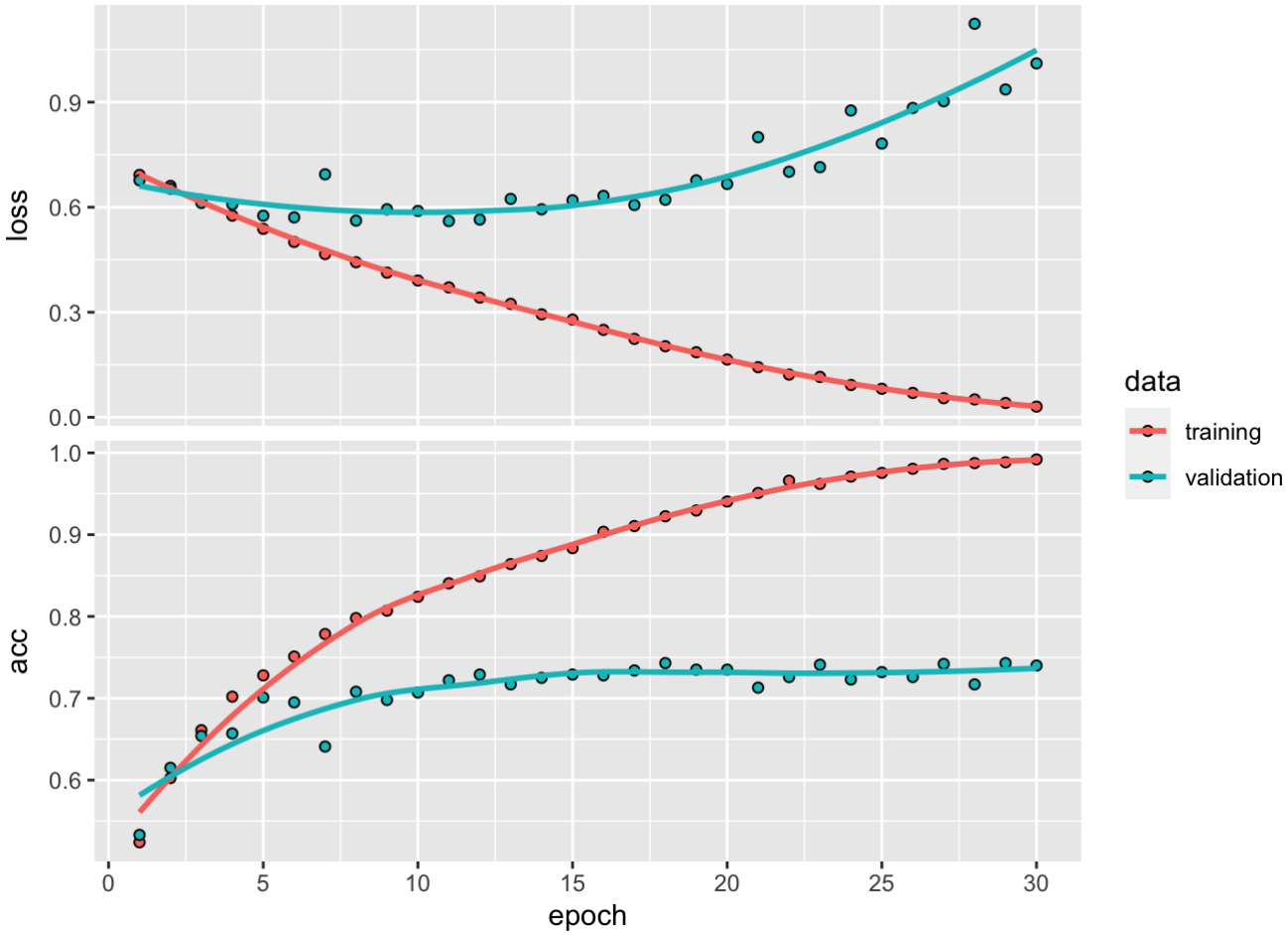
```
history <- model %>% fit_generator(
  train_generator,
  # steps_per_epoch = # of Training Images / Batch Size -> 2,000/20
  steps_per_epoch = 100,
  epochs = 30,
  validation_data = validation_generator,
  # validation_steps = # of Validation Images / Batch Size -> 1,000/20
  validation_steps = 50
)
```

6.4 Initial Model Results / Evaluation

```
# Save model
model %>% save_model_hdf5("cats_and_dogs_small_1.h5")
```

```
# Plot Model Results
plot(history)
```

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



Based on the plot and accuracy listed on the last epoch of 0.9935, there is likely a case of over-fitting due to the small sample size when training the data. We can use tools like Data Augmentation and Dropout Regularization in Keras to improve the model overall.

6.5 Data Augmentation

Creating new training data from existing training data is used in a technique called Data Augmentation. This is done in image data augmentation by creating transformed versions of images in the training data set, expanding the training data into more possible examples (Brownlee, *How to Configure Image Data Augmentation in Keras* 2019).

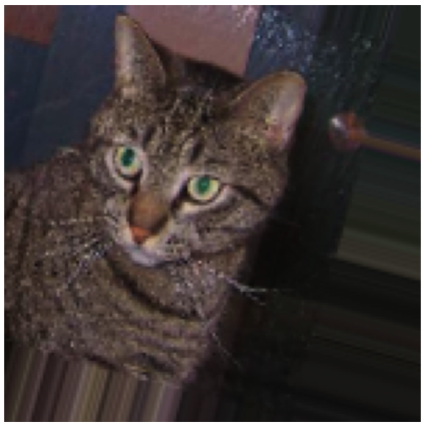
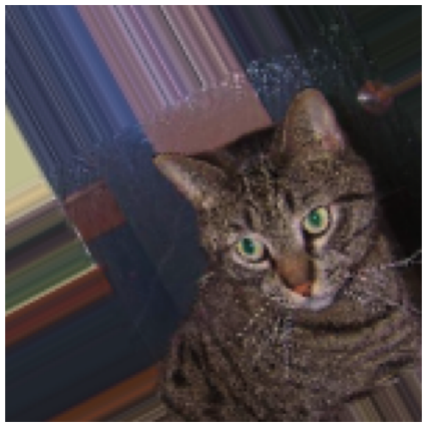
```
datagen <- image_data_generator(
  # Rescale Images by 1/255
  rescale = 1/255,
  # Rotate picture by 40 degrees
  rotation_range = 40,
  # Randomly translate picture vertically and horizontally
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  # Shear transformation
  shear_range = 0.2,
  # Randomly zoom inside a picture
  zoom_range = 0.2,
  # Flip half of the image horizontally
  horizontal_flip = TRUE,
)
```

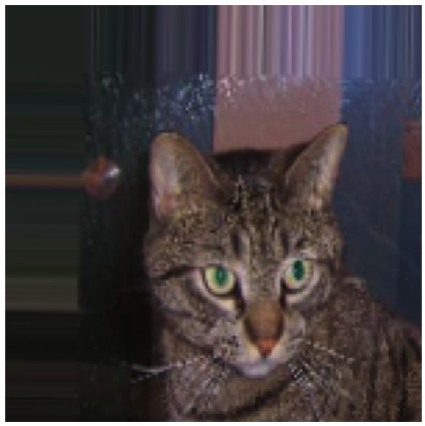
Sample augmented images

```
fnames <- list.files(train_cats_dir, full.names = TRUE)
# Choose one image to augment
img_path <- fnames[[1]]
img <- image_load(img_path, target_size = c(150, 150))
img_array <- image_to_array(img)
img_array <- array_reshape(img_array, c(1, 150, 150, 3))

augmentation_generator <- flow_images_from_data(img_array,generator = datagen,batch_size
= 1)

x <- par(mfrow = c(2, 2), pty = "s", mar = c(1, 0, 1, 0))
for (i in 1:4) {
  batch <- generator_next(augmentation_generator)
  plot(as.raster(batch[1,,,]))
}
```



```
par(x)
```

6.6 Set New Data-Augmentation Generators

```
train_generator <- flow_images_from_directory(
  # Target Directory
  train_dir,
  # Use revised datagen
  datagen,
  # Resize images to 150 X 150
  target_size = c(150, 150),
  # 20 images per batch
  batch_size = 20,
  class_mode = "binary"
)

validation_generator <- flow_images_from_directory(
  # Target Directory
  validation_dir,
  # Use previous datagen
  validation_datagen,
  # Resize images to 150 X 150
  target_size = c(150, 150),
  # 20 images per batch
  batch_size = 20,
  class_mode = "binary"
)
```

6.7 Revised Model

For the revised model, we need to implement Dropout Regularization which will help eliminate any generalization errors due to overfitting.Dropout Regularization during model training some of the layer outputs are randomly ignored or dropped, and each update to a layer during training is performed with a different view of the configured layer (Brownlee, *Dropout Regularization in Deep Learning Models With Keras* 2019).

```
# Revised model with dropout
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(150, 150, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
```

Compile Model

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

```
# Model Summary
summary(model)
```

```
## Model: "sequential_1"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## conv2d_4 (Conv2D)            (None, 148, 148, 32)      896
## _________________________________________________________________________
## max_pooling2d_4 (MaxPooling2D)  (None, 74, 74, 32)     0
## _________________________________________________________________________
## conv2d_5 (Conv2D)            (None, 72, 72, 64)        18496
## _________________________________________________________________________
## max_pooling2d_5 (MaxPooling2D)  (None, 36, 36, 64)     0
## _________________________________________________________________________
## conv2d_6 (Conv2D)            (None, 34, 34, 128)       73856
## _________________________________________________________________________
## max_pooling2d_6 (MaxPooling2D)  (None, 17, 17, 128)    0
## _________________________________________________________________________
## conv2d_7 (Conv2D)            (None, 15, 15, 128)       147584
## _________________________________________________________________________
## max_pooling2d_7 (MaxPooling2D)  (None, 7, 7, 128)      0
## _________________________________________________________________________
## flatten_1 (Flatten)          (None, 6272)              0
## _________________________________________________________________________
## dropout (Dropout)            (None, 6272)              0
## _________________________________________________________________________
## dense_2 (Dense)              (None, 512)               3211776
## _________________________________________________________________________
## dense_3 (Dense)              (None, 1)                 513
## =========================================================================
## Total params: 3,453,121
## Trainable params: 3,453,121
## Non-trainable params: 0
## _________________________________________________________________________
```

6.8 Train with revised model

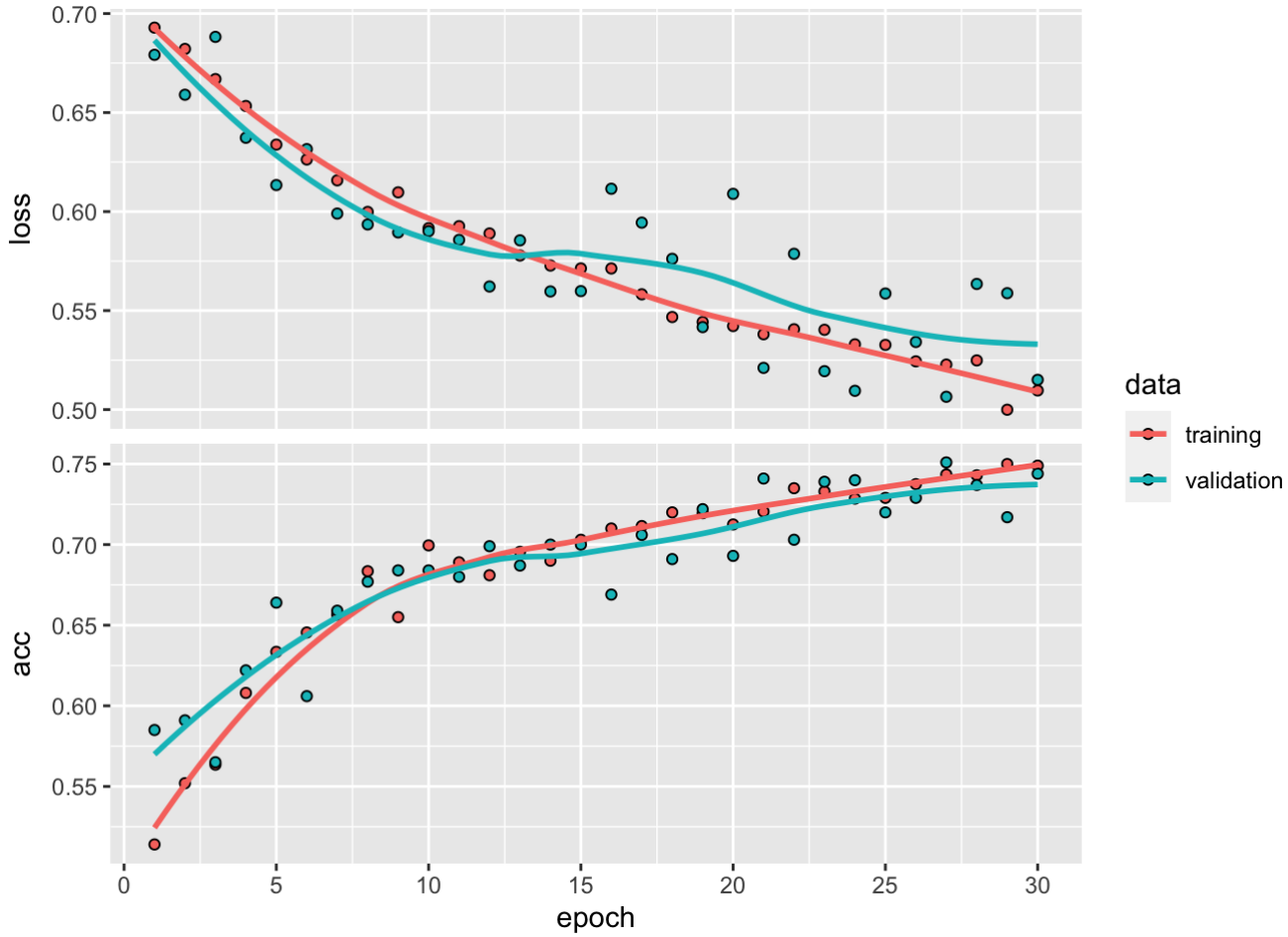
```
history <- model %>% fit_generator(
  train_generator,
  # steps_per_epoch = # of Training Images / Batch Size -> 2,000/20
  steps_per_epoch = 100,
  epochs = 30,
  validation_data = validation_generator,
  # validation_steps = # of Validation Images / Batch Size -> 1,000/20
  validation_steps = 50
)
```

6.9 Revised Model Evaluation

```
# Save revised model
model %>% save_model_hdf5("cats_and_dogs_small_2.h5")
```

```
# Plot Revised Model Results
plot(history)
```

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



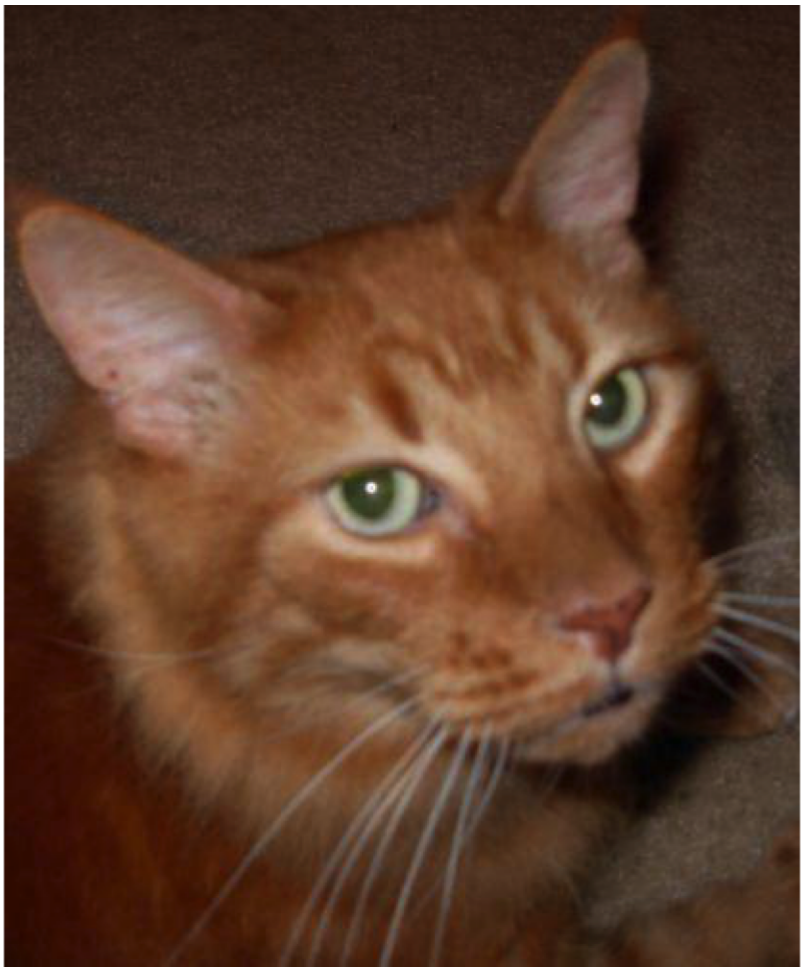
The model now has an accuracy of 0.7490, and as the plot shows is now tracking more in-line with the validation data.

7.0 Model Predictions

7.1 Select 1 Cat and Dog Picture From Test Data

```
# View sample test dog image
test_image_dog <- readImage(file.path(test_dir, "1501.jpg"))
display(test_image_dog)
```

```
# View sample test cat image
test_image_cat <- readImage(file.path(test_dir, "1502.jpg"))
display(test_image_cat)
```



7.2 Load and Augment Images

```
# Load and Augment Dog Image
img_path1 <- "/Users/xen/Documents/CSDA 1040/ASSIGNMENT3/test1/1501.jpg"
img <- image_load(img_path1, target_size = c(150, 150))
img_dog <- image_to_array(img)
img_dog <- array_reshape(img_dog, c(1, 150, 150, 3))
img_dog <- img_dog / 255

dim(img_dog)
```

```
## [1]   1 150 150   3
```

```
# Load and Augment Cat Image
img_path2 <- "/Users/xen/Documents/CSDA 1040/ASSIGNMENT3/test1/1502.jpg"
img <- image_load(img_path1, target_size = c(150, 150))
img_cat <- image_to_array(img)
img_cat <- array_reshape(img_cat, c(1, 150, 150, 3))
img_cat <- img_dog / 255

dim(img_cat)
```

```
## [1]   1 150 150   3
```

7.3 Generate Predictions

The values generated by the model for prediction, will be shown as 0 for cats and 1 for dogs.

```
# Actual Prediction for Dog Image
class_pred <- model %>% predict_classes(img_dog)
class_pred[1]
```

```
## [1] 1
```

```
# Prediction Score for Dog Image
preds <- model %>% predict(img_dog)
preds[1,]
```

```
## [1] 0.7625087
```

```
# Actual Prediction
class_pred <- model %>% predict_classes(img_cat)
class_pred[1]
```

```
## [1] 0
```

```
# Prediction Score
preds <- model %>% predict(img_cat)
preds[1,]
```

```
## [1] 0.009058389
```

The model is able to effectively identify a cat or a dog based on an image provided.

8.0 Conclusions and Recommendations

CAPTCHA or HIP systems using images are no longer effective as a security solution to stop spam or computer bots. Deep Learning implementations such as Convolutional Neural Networks can effectively classify images that it has been trained on.

Those that deploy image classifying HIP or CAPTCHA systems, can take certain steps to improve the overall security of end users. First, HIP systems can use a diverse set of images. Images of various subjects like various other animals or objects would decrease the overall accuracy of convolutional neural networks if the subject was not included in the training data. HIP systems can also include other mediums, such as video and audio clips, as this may be more of a challenge to program model in spam bots. Another option is to implement interactive elements to HIP or CAPTCHA systems that require users to interact with objects, like moving segments of a picture to complete a puzzle.

Lastly, the underlying data used in image HIP or CAPTCHA systems would need to have frequent or recurring refreshes and reviews to determine overall effectiveness at ensuring the security of the end users.

9.0 Deployment

The underlying code of this markdown report, can be found on Github (https://github.com/xnazar/CSDA1040Assignment3) and on the Shiny web application About page as listed on shinyapps.io (https://jose-g.shinyapps.io/catsdogs)

10.0 Bibliography

Allaire, J. (2020). TensorFlow for R - Basic Image Classification. Retrieved July 10, 2020, from https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_classification/ (https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_classification/)

Brownlee, J. (2019, July 05). How to Configure Image Data Augmentation in Keras. Retrieved July 10, 2020, from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/ (https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/)

Brownlee, J. (2019, September 13). Dropout Regularization in Deep Learning Models With Keras. Retrieved July 10, 2020, from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/ (https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/)

Carnegie Mellon University. (2010). CAPTCHA: Telling Humans and Computers Apart Automatically. Retrieved July 10, 2020, from http://www.captcha.net/ (http://www.captcha.net/)

Chellapilla, K., Larson, K., Simard, P., & Czerwinski, M. (2018, October 17). Designing Human Friendly Human Interaction Proofs (HIPS). Retrieved July 10, 2020, from https://www.microsoft.com/en-us/research/publication/designing-human-friendly-human-interaction-proofs-hips/ (https://www.microsoft.com/en-us/research/publication/designing-human-friendly-human-interaction-proofs-hips/)

Chollet, F., & Allaire, J. J. (2018). Deep learning for computer vision. In Deep Learning with R. Shelter Island, NY: Manning Publications. Retrieved July 10, 2020, from https://livebook.manning.com/book/deep-learning-with-r/chapter-5 (https://livebook.manning.com/book/deep-learning-with-r/chapter-5)

Cukierski, W. (2014). Dogs vs. Cats. Retrieved July 10, 2020, from https://www.kaggle.com/c/dogs-vs-cats/data (https://www.kaggle.com/c/dogs-vs-cats/data)

Microsoft. (2017, October 2). Kaggle Cats and Dogs Dataset. Retrieved July 10, 2020, from https://www.microsoft.com/en-us/download/details.aspx?id=54765 (https://www.microsoft.com/en-us/download/details.aspx?id=54765)

Petfinder.com. (2018, June 22). About Petfinder. Retrieved July 10, 2020, from https://www.petfinder.com/about/ (https://www.petfinder.com/about/)