Fall 2019, BIOS 740: Computing Assignment 3 (C3)

In this assignment, you will learn how to apply convolutional neural networks in R to distinguish images of cats versus dogs. You will find the data via this microsoft link (hyper-linked) and you can download it manually. Note that the dataset is large, around 900MB in size so it will take time to download. Please read the document about data usage in the folder carefully and follow the terms listed. If you open the data, you will find that the cats versus dog data have already been organized in folders of "Cat" and "Dog". Due to the large input data size, you should expect the training process to take some decent amount of time so please do not wait until the last minute to work on this assignment.

The goal of this assignment is to build a binary classifier using the Cats vs. Dogs images. The classifier should be able to take in an image and make a classification decision of "cat" or "dog".

Here are the guidelines:

- Data preprocessing.

  1. Unzip the file.
  2. The original dataset contains 25,000 images, which will slow down the computation quite a lot. To prevent your computer from burning up, we recommend that you only use the first 12,000 images of the dataset to speed up computation (i.e., 6,000 for the cats folder and 6,000 for the dogs folder) before the random split. This clipping is optional unless you do not have a problem running all 25,000 images.
  3. Split the clipped data randomly into training and testing sets (just once, no need to perform k-fold cross validation this time) separately for cats and dogs. In other words, you will create a new directory containing two subfolders, 'training' and 'testing', each of which has two sub-subfolders, 'cats' and 'dogs'.
  4. Use 9 : 1 as the training and testing ratio and 2019 as the randomization seed.

  You can use the function `split_data` provided at the end of this assignment to do this or write your own function.

- Define a Keras convolutional neural network model with at least 2 convolutional + max pooling layers, 1 flatten layer, and 2 dense layers. Feel free to add more layers! Specify activation functions to be `relu` for all layers except the output layer. The activation function of the output layer could be `sigmoid` or `softmax` depending on how many classes you define. Some hyperameters can only take certain values because of the structure our dataset but more hyperparameter values (such as batch size, filters, kernel sizes, optimizer, etc.) are up to the neural network trainer (which is you!). We recommend that you look up what values are often used and learn from existing DL examples. You need to provide sufficient justification for each hyperparameter value you choose.

- Use `image_data_generator` to rescale the images by 1/255 or perform other kinds data augmentations before training. Use `flow_images_from_directory` to load data from a given directory and generate batches.

- Train the Keras model with the training dataset and test it on the testing dataset with `fit_generator`. Try 5 to 10 epochs. **Note:** This will take a couple of minutes to run one epoch so please plan ahead. If you find yourself running for a long time (e.g., over 12 hours), that means something is wrong with your model or your model is too big. High performance

computing will speed up the computing time so we can afford more epochs. For the purpose of this assignment we do not expect you to get perfect accuracies and you should be able to get adequate results on your local machines.

- Record and plot the training and testing accuracy per epoch. Record and plot the training and testing loss per epoch. Organize the training and testing accuracies and losses in a table. Include plots in the report with sufficient labels and caption. Comment on the results. Do not use too many decimal places.

- Write a paragraph (at least 3 sentences) to comment on how this classification tool can be useful in precision medicine and give a specific example or scenario of your choice.

- *[Bonus points]* Find an image of a cat or a dog online (or take a picture of your own pet!) and have it classified using your trained network without crashing. Check if your model prediction matches with the image. The test image can be easy or challenging. It is up to you. To make it more challenging, try exotic kinds of cats or dogs or an image with both dogs and cats. Note that the image need not predict the image label correctly for you to receive credit. We mainly look for no crashing errors when grading, but it would be nice if your neural network could get it right. Rerport the predicted probability as well as the predicted class of your test image. Include also the image in your report so the grader can see the test image you selected.

This assignment should be submitted as a well-written technical report with sufficient explanation in the same style as Methods and Results sections in peer-reviewed journals. It should be programmed in R and the `keras` (hyper-linked) package. You can find installation instructions here. We recommend that you look up the documentation and tutorials through the link provided, which will help you finish this assignment. For example, MNIST CNN (hyper-linked) is a good starting point for you to get familiar with Keras.

This assignment is **due before class on November 13**. Please turn in the report together with your **code and intermediate program output** (e.g., the progress bar of each epoch, time elapsed, accuracies, and losses) as an appendix as one stapled hard copy. Your code should have an appropriate amount of comments in between. The report needs to be typed up and no longer than 5 pages (including results but not including code) and the code part should be no longer than 3 pages. PDF files generated from LaTeX or RMarkdown are preferred. Email submission is only allowed if notified and approved by the course instructor in advance. The quality of the report is judged by (1) completion, (2) statistical correctness, (3) code presentation, and (4) explanation and report presentation.

Hints

- Here is the `split_data` function for you to clip, randomly split, and organize the original dataset. We do not provide comments for this function because you should be able to figure out what this function is and what the arguments do. Feel free to write you own function to preprocess the data.

```
split_data <- function(source, training, testing, split_size, clip, seed){
set.seed(seed)

files = c()
```

```
filenames <- list.files(source)
filenames <- filenames[1:clip]

for (filename in filenames){
file = paste0(source, filename)
if (file.info(file)$size > 0){
files <- c(files, filename)
} else {
cat(filename, "has zero length and thus removed.")
}
}

training_length = length(files) * split_size
shuffled_set = sample(files)
training_set = shuffled_set[1:training_length]
testing_set = shuffled_set[-(1:training_length)]

for (filename in training_set){
this_file = paste0(source, filename)
destination = paste0(training, filename)
file.copy(this_file, destination)
}

for (filename in testing_set){
this_file = paste0(source, filename)
destination = paste0(testing, filename)
file.copy(this_file, destination)
}
}
```

- After preprocessing, you would need to define and compile the CNN model. Here is some very crude, incomplete example code to help you get started with:

```
model <- keras_models_sequential() %>%
layer_conv_2d(filters, kernel_size, activation, input_shape) %>%
layer_max_pooling_2d(pool_size) %>%
layer_conv_2d(...) %>%
layer_max_pooling_2d(...) %>%
...
layer_flatten() %>%
layer_dense(units, activation) %>%
layer_dense(units, activation)

model %>% compile(optimizer, loss, metrics)

datagen <- image_data_generator(rescale, ...)

train_generator <- flow_images_from_directory(directory = training_dir,
```

```
 generator = datagen, target_size, class_mode, batch_size)

test_generator <- flow_images_from_directory(directory = testing_dir,
 generator = datagen, target_size, class_mode, batch_size)

model %>% fit_generator(generator, steps_per_epoch, epochs, verbose, validation_data)
```