BIOS 740 - Computing HW 3

Andrew Walther

November 17, 2021

1 Introduction

In the field of precision medicine, clinicians seek to implement data-driven decision making to select an optimal treatment regime for patient(s) given observed characteristics about a patient. These decisions are designed to be generalizable and reproducible to the point where a clinician can apply what is known to be an optimal treatment protocol after understanding background information about a patient, or population, in the context of public health. Some methods for exploring the development of optimal treatment regimes in precision medicine include random forests, sequential multiple randomized clinical trials, and reinforcement learning, among many others. In this report, we consider the applications of convolutional neural networks (CNN) [2] for a relatively simple classification task in order to understand the possible benefits of CNN's when applied to precision medicine tasks. A convolutional neural network is a type of deep learning algorithm where the model takes input data and assigns relative values of importance (weights) to different aspects of the data (feature engineering) so that the model is capable of differentiating different features in the data from each other. The CNN is then able to use these weights to eventually make predictions for an outcome of interest based on the input data. In brief, CNN's consist of a series of layers with interconnected nodes between the layers where learned features can be passed through sequential layers to be further refined until a final classification is made on the original data and one of the major advantages of a convolutional neural network is that they are able to manage carry out both feature engineering and classification tasks simultaneously.

Here we consider a large collection of various images of dogs and cats. The task put forth is to implement a convolutional neural network classifier that is capable of distinguishing images of cats from images of dogs. It follows that successful and effective implementation of a simpler task like this could be extended into the field of precision medicine. In those cases, a model might consider an abundance of difficult to understand patient information with the capability of classifying each patient a particular course of treatment based off learning the CNN has performed on prior patient information.

2 Methods

2.1 Kaggle: Dogs and Cats Data

The data used in this task is sourced from kaggle.com and is the collection of images supplied for the 'Dogs vs. Cats' (https://www.kaggle.com/c/dogs-vs-cats) coding challenge where participants are tasked with developing an algorithm to distinguish dogs from cats. The data contains 25,000 total images with 12,500 each for dogs and cats. We randomly separated the images into separate sets that held 90% of the images for model training and 10% of the images for model validation (testing). Therefore, we ended up with approximately 11,250 images of cats and dogs each, for model training and approximately 1250 images of cats and dogs each for validation. These images were separated into appropriate train and test folders to be accessed later on by the neural network model.

2.2 Convolutional Neural Network Setup

For this task, we elected to implement a convolutional neural network in R. This implementation utilizes the machine learning packages Keras[3] and Tensorflow[1]. Our keras convolutional neural network, called with the function 'keras-model-sequential()', consists of three convolution layers, three pooling layers, a flatten layer, and two dense layers. The purpose of a convolution layer is to extract features from the data, like the edges of a dog's face or ears. Convolution layers are alternated with pooling layers which are used to reduce the dimension of the data being considered in order to make it easier for data to be processed. These layers help to continually pull out important features in the data as well. Our model uses "max pooling" layers, which return the largest value for an area of the data or most influential feature to be considered in subsequent layers as the dimension of the data is reduced. A flatten layer simply reforms the remaining data into a single dimension vector and

dense layers are just layers where every node in the layer is connected to every node from the previous layer and every node of the subsequent layer.

In this model, we specified that the image inputs are all 150 pixels by 150 pixels in the convolution layers and the number of filters to be 16, 32, and 64 over the three convolution layers in order to reduce the overall dimension of the data further along in the layers. In intermediate layers, we selected 'relu' as the activation function[3] in order to only consider non-negative feature values. Finally, we selected the final layer activation function to be 'sigmoid'[3] because we were interested in a binary classification between cats and dogs. We also specified the kernel and pool sizes based on conventional methods used with CNN's. Overall, this CNN had 9,494,561 trainable parameters and 0 non-trainable parameters.

2.3 Image Processing

Following the specification of our convolutional neural network model, we processed each of the 25,000 images used in this task. First, the image Red/Green/Blue (RGB) pixel levels were re-scaled to be over a range of 0-255 (256 possible levels). Subsequently, the images were resized to be 150 pixels in width by 150 pixels in height. The images were also grouped into batches of 250 images (1% of all data) to be used as inputs for model training and validation. After grouping the images into batches, we confirmed that each batch contained 250 images with dimensions of 150 pixels by 150 pixels and either a 0 or 1 flag to denote if the image is of a cat or dog.

2.4 CNN Model Training & Validation

We trained the convolutional neural network model by utilizing the previously generated batches as input data. The CNN is then performs convolution with a relu activation function and then pooling to learn features from the data. After processing through all of the convolution and pooling layers, the model utilizes a flatten layer to reorganize the feature information and subsequently makes a classification prediction based off of the learned features in the data.

In this particular model, computing power was scarce so we specified that 5 epochs (times the model sees all of the data) would be performed for model training. Additionally, only 50 steps per epoch were specified so the model would only see approximately half of the entire dataset during each epoch. Typically, we would like the model to see the entire dataset during each epoch, but this was not feasible due to technical issues. Finally, model validation was carried out with batches of validation data that we previously generated and only 5 steps were specified for the sake of efficiency and saving computing space.

3 Results

Following the implementation, training, and validation of the convolutional neural network to classify images of cats and dogs, we obtained the following results below in Table 1. Overall, the epoch with the best validation loss and accuracy was the fifth and final epoch. This epoch has a validation loss value of 0.4631 and a validation accuracy value of 0.7750 meaning that after 5 iterations of training on the entirety of the available data, the model was able to correctly classify validation images of cats and dogs with about 78% accuracy or about 3 correct classifications for every 4 attempts. We can also see in the sequential epoch plot of loss and accuracy for the training and validation data in Figure 1 that over the total number of epochs, the model loss values significantly declined and the accuracy values continually increased until validation accuracy flattened out around the 80% mark. The trends we can see with the loss successively decreasing and accuracy increasing are positive things to note because they confirm that the model is continually improving as it is able to see more of the data, but it is also not over-fitting to the data because the accuracy metrics begin to level off after a couple epochs (and we would expect to see this trend continue if more epochs were carried out).

Table 1: CNN Model Accuracy & Loss for Training and Validation

Epoch	Time Elapsed (s)	Loss	Accuracy	Validation Loss	Validation Accuracy
1	608	0.8641	0.5747	0.6411	0.6040
2	619	0.6185	0.6621	0.5608	0.7064
3	611	0.5986	0.6927	0.5252	0.7536
4	601	0.5350	0.7318	0.4808	0.7736
5	582	0.5064	0.7498	0.4631	0.7750

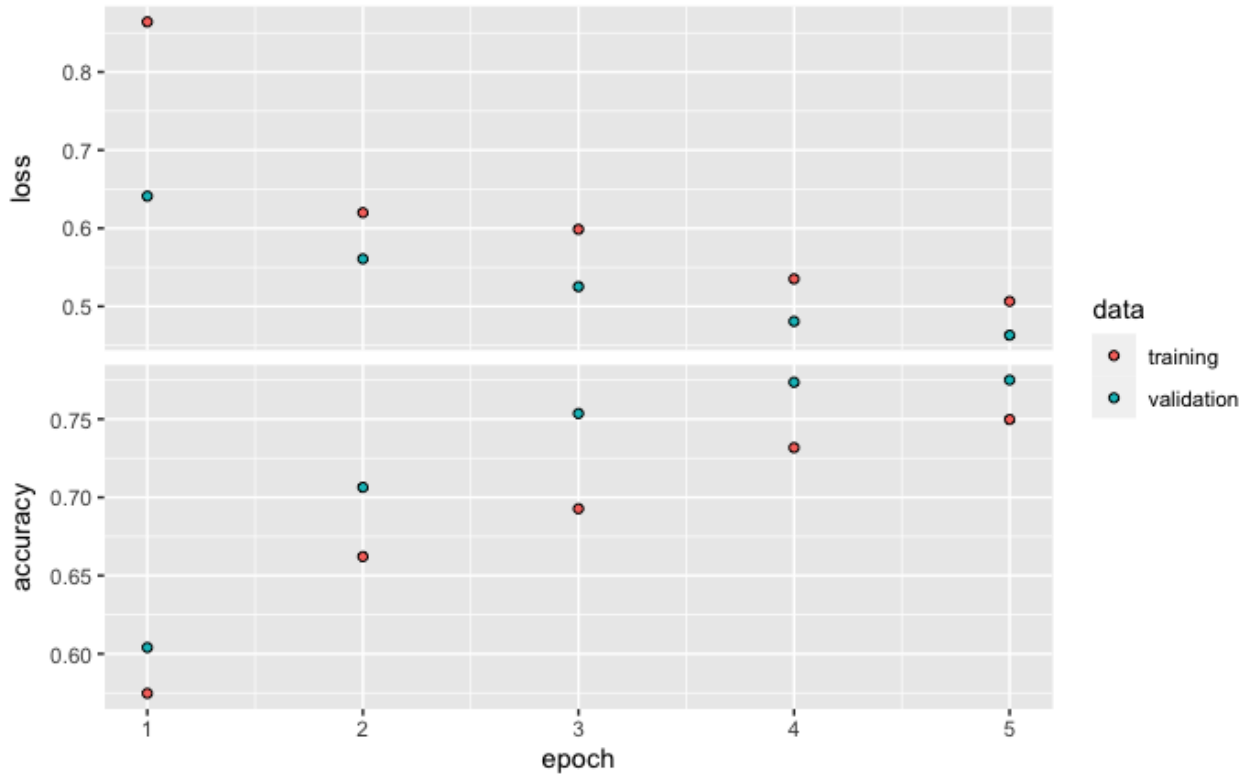

Figure 1: Plot of accuracy and loss for training and validation

4 Discussion

4.1 Model Viability and outlook

We found that the trained convolutional neural network for classifying images of cats and dogs reaches an accuracy level of about 80% success of the binary classification of cat vs. dog validation images. These results show promise in that this method is capable of making accurate predictions more often than not, especially when we are dealing with limited computing power (we were only able to train for 5 epochs on 50% of the data each time without running out of system memory). Additionally, these results show that if additional computing resources were available and utilized, we could train for a greater number of epochs on the full set of data and likely achieve greater overall training and validation accuracy metrics (typically an epoch involves training over the whole set of data, but we limited this to just half of the data). Therefore, further developments on this project would include additionally epochs (10-25) where 100 steps per epoch could be utilized in order to train on the whole set of images for each epoch. Furthermore, we could increase the overall amount of available data by carrying out manipulations to the existing images like clipping sections of the images, rotating the images, reflecting the images, and more to improve the generalizability of the CNN model. We would expect that this additional model training would improve it's adaptability to considering variations in the data which would lead to improved validation accuracy and reduced loss.

4.2 Relevance to precision medicine

Convolutional neural networks can be useful tools in the context of precision medicine because they are designed to take in large amounts of complex data and find connections between different elements of the data. These attributes can be beneficial in precision medicine when we are unsure of which features might be good indicators for how to assign an optimal treatment because the CNN can figure out which features to give stronger weightings to in order to eventually make an optimal treatment prediction. Additionally, a CNN considers feature engineering in the process of model training and predictions so we don't have to depend on selecting appropriate features on our own. Eliminate possibility of human error that can exist in other precision medicine methods such as mis-specifying a propensity model in a regression-based method.

As a relevant example of how neural networks are relevant to precision medicine, we might consider patients who have been diagnosed with colorectal cancer and a physician is attempting to determine which of a few chemotherapy treatments would lead the the best survival for each patient. Additionally, we are fortunate enough to have genomics data for many previous patients who have undergone chemotherapy treatment for

this particular cancer as well as for the new patients where we need to decide which drugs to treat them with. Supposing that we have all of this information along with the drugs each patient received and their subsequent survival times, we could utilize convolutional neural network methods to extract relevant features in the genome from past patients that were indicators of strong or poor survival which may give insight as to which chemotherapy drugs might be optimal to use for current and future patients.

References

[1] Y. Tang D. Eddelbuettel N. Golding D. Falbel, J. Allaire and Maintainer Tomasz Kalinowski T. Kalinowski. Package 'tensorflow'. *R software*, 2021.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[3] F. Chollet Y. Tang W. Van Der Bijl M. Studer T. Kalinowski, D. Falbel and Maintainer Tomasz Kalinowski S. Keydana. Package 'keras'. *R software*, 2021.

```
### BIOS 740 Computing HW 3 Code
### Andrew Walther
### November 17, 2021
################################################################################
#set randomization seed
set.seed(2021)
#load relevant packages
library(tidyverse)
library(keras)
library(tensorflow)
################################################################################
#Extract subfilepaths from source & confirm image counts
PetImages_Dirs <- list.dirs(path = "~/OneDrive - University of North Carolina
at Chapel Hill/UNC - Fall 2021/BIOS 740 - Kosorok/Programming Assignments/
Computing HW3/catdogdata/PetImages",
                            full.names = TRUE, recursive = TRUE)
directory_object_counts <- sapply(PetImages_Dirs, function(dir)
{length(list.files(dir))})
################################################################################
#set up source, training, & testing directories via filepaths
#source (contains all cat & dog images)
root_directory <- "~/OneDrive - University of North Carolina at Chapel Hill/
UNC - Fall 2021/BIOS 740 - Kosorok/Programming Assignments/Computing HW3/
catdogdata/PetImages"
source_dogs_directory <- "~/OneDrive - University of North Carolina at Chapel
Hill/UNC - Fall 2021/BIOS 740 - Kosorok/Programming Assignments/Computing
HW3/catdogdata/PetImages/Dog"
source_cats_directory <- "~/OneDrive - University of North Carolina at Chapel
Hill/UNC - Fall 2021/BIOS 740 - Kosorok/Programming Assignments/Computing
HW3/catdogdata/PetImages/Cat"
#set up training directories (cats & dogs)
training_directory <- file.path(root_directory, "training")
dir.create(training_directory)
training_dogs_directory <- file.path(training_directory, "dogs")
dir.create(training_dogs_directory)
training_cats_directory <- file.path(training_directory, "cats")
dir.create(training_cats_directory)
#set up testing directories (cats & dogs)
testing_directory <- file.path(root_directory, "testing")
dir.create(testing_directory)
testing_dogs_directory <- file.path(testing_directory, "dogs")
dir.create(testing_dogs_directory)
testing_cats_directory <- file.path(testing_directory, "cats")
dir.create(testing_cats_directory)
################################################################################
#Data splitting & randomization, assignment to train/test
split_data <- function(source_dir, training_dest, testing_dest, split_size){
        #file paths for cat/dog images
        files <- list.files(path = source_dir, full.names = T)
        size <- file.size(files)
        #removing images with size zero & randomize remaining images
        shuffled_set <- cbind (files, size) %>% subset(size > 0, select =
c(files)) %>%
                as.character() %>% sample(replace = F)
```

```
        #select X% of images into training & remaining to testing
        training_length <- length(shuffled_set) * split_size
        testing_length <- length(shuffled_set) * (1 - split_size)
        training_set <- shuffled_set[1:training_length]
        testing_set <- shuffled_set[(training_length+1):length(shuffled_set)]
        #move train/test sets to appropriate directory (hide w/invisible)
        invisible(file.copy(from = training_set, to = training_dest))
        invisible(file.copy(from = testing_set, to = testing_dest))}
#Cat image splitting (test/train)
split_data(source_dir = source_cats_directory,
           training_dest = training_cats_directory,
           testing_dest = testing_cats_directory,
           split_size = 0.9)
#Dog image splitting (test/train)
split_data(source_dir = source_dogs_directory,
           training_dest = training_dogs_directory,
           testing_dest = testing_dogs_directory,
           split_size = 0.9)
#Check counts for dogs & cats (training = 11249 (90%) & testing = 1249 (10%))
cat("Cat training images:", length(list.files(training_cats_directory)),
'\n')
cat("Cat testing images:", length(list.files(testing_cats_directory)), '\n')
cat("Dog training images:", length(list.files(training_dogs_directory)),
'\n')
cat("Dog testing images:", length(list.files(testing_dogs_directory)), '\n')
###############################################################################
#set up CNN w/ convolution, pooling, flatten, & dense layers
model <- keras_model_sequential() %>%
        #convolution & pooling layers 1 (150x150 images with RGB color (3))
        layer_conv_2d(input_shape = c(150, 150, 3), filters = 16, kernel_size
= c(3, 3), activation = 'relu') %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        #convolution & pooling layers 2
        layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation =
'relu') %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        #convolution & pooling layers 3
        layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation =
'relu') %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        #flatten layer 1
        layer_flatten() %>%
        #dense layers 1 & 2
        layer_dense(units = 512, activation = "relu") %>%
        #binary classification needs sigmoid activation function
        layer_dense(units = 1, activation = "sigmoid") %>%
        #compile model
        compile(loss = 'binary_crossentropy', optimizer =
optimizer_rmsprop(lr = 0.001), metrics = 'accuracy')
summary(model)
###############################################################################
#Training: Rescale RGB by 1/255
train_datagen <- image_data_generator(rescale = 1/255)
train_generator <- flow_images_from_directory(
```

```
        #target directory
        directory = training_directory,
        #training data generator
        generator = train_datagen,
        #resize images to 150x150 pixels
        target_size = c(150, 150),
        #Input batches of 250 images
        batch_size = 250,
        #binary class for (binary_crossentropy)
        class_mode = 'binary')
#Validation(testing): Rescale RGB by 1/255
validation_datagen <- image_data_generator(rescale = 1/255)
validation_generator <- flow_images_from_directory(
        #target directory
        directory = testing_directory,
        #testing data generator
        generator = validation_datagen,
        #resize images to 150x150 pixels
        target_size = c(150, 150),
        #Input batches of 250 images
        batch_size = 250,
        #binary class for (binary_crossentropy)
        class_mode = 'binary')
#check contents of first training batch (set of 250, 150x150 etc.)
batch_train <- generator_next(train_generator)
str(batch_train)
#check contents of first testing batch (set of 250, 150x150 etc.)
batch_test <- generator_next(validation_generator)
str(batch_test)
###############################################################################
start.time <- Sys.time()
#model fitting object
model_history <- model %>% fit_generator(
        generator = train_generator,
        #number of size 250 batches in each epoch
        steps_per_epoch = 50,
        #iterations over all of data
        epochs = 5,
        validation_data = validation_generator,
        validation_steps = 5)
#print out best loss and its corresponding accuracy
epoch <- which.min(model_history$metrics$val_loss)
loss <- round(model_history$metrics$val_loss[epoch],3)
accuracy <- round(model_history$metrics$val_accuracy[epoch],3)
print(loss)
print(accuracy)
#plot training results
plot(model_history)
summary(model_history)
#function runtime
end.time <- Sys.time()
time.elapsed <- end.time-start.time
print(time.elapsed)
```

BIOS 740 Computing HW 3 – Code/Figures Appendix

CNN Model Parameters

```
Loaded Tensorflow version 2.0.0
2021-11-17 23:19:16.626441: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is optimized with Intel(R)
MKL-DNN to use the following CPU instructions in performance critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2021-11-17 23:19:16.626833: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with default inter op
setting: 4. Tune using inter_op_parallelism_threads for best performance.
the `lr` argument has been renamed to `learning_rate`.Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 148, 148, 16)      448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 74, 74, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 36, 36, 32)      0
_________________________________________________________________
conv2d (Conv2D)              (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 17, 17, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 18496)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               9470464
_________________________________________________________________
dense (Dense)                (None, 1)                 513
=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

Confirmation that generated batches have 250 images with 150x150 pixel dimensions, 3 RGB entries, and a 0 or 1 classifier for cat vs. dog

```
Found 24560 images belonging to 2 classes.
Found 5549 images belonging to 2 classes.
List of 2
 $ : num [1:250, 1:150, 1:150, 1:3] 0.247 0.173 0.137 0.231 0.549 ...
 $ : num [1:250(1d)] 1 1 0 1 0 1 1 1 1 0 ...
List of 2
 $ : num [1:250, 1:150, 1:150, 1:3] 0.475 0.624 0.549 0.678 0.227 ...
 $ : num [1:250(1d)] 0 1 1 0 1 1 0 0 0 0 ...
Time difference of 5.116195 secs
```

Epoch progress bars, time elapsed, training loss, training accuracy, validation loss, & validation accuracy

```
Epoch 1/5
50/50 [==============================] - 608s 12s/step - loss: 0.8641 - accuracy:
0.5747 - val_loss: 0.6411 - val_accuracy: 0.6040
Epoch 2/5
50/50 [==============================] - 619s 12s/step - loss: 0.6185 - accuracy:
0.6621 - val_loss: 0.5608 - val_accuracy: 0.7064
Epoch 3/5
50/50 [==============================] - 611s 12s/step - loss: 0.5986 - accuracy:
0.6927 - val_loss: 0.5252 - val_accuracy: 0.7536
Epoch 4/5
50/50 [==============================] - 601s 12s/step - loss: 0.5350 - accuracy:
0.7318 - val_loss: 0.4808 - val_accuracy: 0.7736
Epoch 5/5
50/50 [==============================] - 582s 12s/step - loss: 0.5064 - accuracy:
0.7498 - val_loss: 0.4631 - val_accuracy: 0.7750
```

5 Epoch overall time elapsed

```
> end.time <- Sys.time()
> time.elapsed <- end.time-start.time
> print(time.elapsed)
Time difference of 50.37726 mins
>
```

Validation loss & accuracy from optimal epoch

```
> #print out best loss and its corresponding accuracy
> epoch <- which.min(model_history$metrics$val_loss)
> loss <- round(model_history$metrics$val_loss[epoch],3)
> accuracy <- round(model_history$metrics$val_accuracy[epoch],3)
> print(loss)
[1] 0.463
> print(accuracy)
[1] 0.775
```

Plot of training & validation loss & accuracy for epochs 1-5