bshild6\_Assignment\_2

Brandon Lloyd Shields

10/24/2021

#Creating Directories for training, Validation and Testing.  
  
original\_dataset\_dir <- "~/Downloads/dogs-vs-cats/train"  
  
base\_dir\_1000 <- "~/Downloads/cats\_and\_dogs\_1000"  
dir.create(base\_dir\_1000)  
  
train\_dir\_1000 <- file.path(base\_dir\_1000, "train")  
dir.create(train\_dir\_1000)  
validation\_dir <- file.path(base\_dir\_1000, "validation")  
dir.create(validation\_dir)  
test\_dir <- file.path(base\_dir\_1000, "test")  
dir.create(test\_dir)  
  
train\_cats\_dir\_1000 <- file.path(train\_dir\_1000, "cats")  
dir.create(train\_cats\_dir\_1000)  
  
train\_dogs\_dir\_1000 <- file.path(train\_dir\_1000, "dogs")  
dir.create(train\_dogs\_dir\_1000)  
  
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)  
  
test\_cats\_dir <- file.path(test\_dir, "cats")  
dir.create(test\_cats\_dir)  
  
test\_dogs\_dir <- file.path(test\_dir, "dogs")  
dir.create(test\_dogs\_dir)  
  
#For 1000 training sample  
fnames <- paste0("cat.", 1:500, ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),   
 file.path(train\_cats\_dir\_1000))   
  
#For 500 Validation Set  
fnames <- paste0("cat.", 501:750, ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),   
 file.path(validation\_cats\_dir))  
  
#For 500 Test set  
fnames <- paste0("cat.", 751:1000, ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),  
 file.path(test\_cats\_dir))  
  
#For 1000 training set  
fnames <- paste0("dog.", 1:500, ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),  
 file.path(train\_dogs\_dir\_1000))  
  
#For 500 validation set  
fnames <- paste0("dog.", 501:750, ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),  
 file.path(validation\_dogs\_dir))   
  
#For 500 Test set  
fnames <- paste0("dog.", 751:1000, ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),  
 file.path(test\_dogs\_dir))

#Building Initial Model for Classification using 1000 training samples - 500 cat and 500 Dog  
model\_1000 <- 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")

summary(model\_1000)

model\_1000 %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(learning\_rate = 1e-4),  
 metrics = c("acc")  
)

train\_datagen <- image\_data\_generator(rescale = 1/255)  
validation\_datagen <- image\_data\_generator(rescale = 1/255)  
  
train\_generator\_1000 <- flow\_images\_from\_directory(  
 # This is the target directory  
 train\_dir\_1000,  
 # This is the data generator  
 train\_datagen,  
 # All images will be resized to 150x150  
 target\_size = c(150, 150),  
 batch\_size = 20,  
 # Since we use binary\_crossentropy loss, we need binary labels  
 class\_mode = "binary"  
)  
  
validation\_generator <- flow\_images\_from\_directory(  
 validation\_dir,  
 validation\_datagen,  
 target\_size = c(150, 150),  
 batch\_size = 20,  
 class\_mode = "binary"  
)

history\_1000 <- model\_1000 %>% fit\_generator(  
 train\_generator\_1000,  
 steps\_per\_epoch = 50,  
 epochs = 25,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)

model\_1000 %>% save\_model\_hdf5("cats\_and\_dogs\_small\_1.1000")  
  
plot(history\_1000)

#There was significant evidence of overfitting due to a small sample size. I will now employ data augmentation and regularization.

#Introducing a dropout layer  
  
model\_1000\_reg <- 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")   
   
model\_1000\_reg %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-4),  
 metrics = c("acc")  
)

# Setting Up data augmentation to reduce overfitting  
  
datagen <- image\_data\_generator(  
 rescale = 1/255,  
 rotation\_range = 40,  
 width\_shift\_range = 0.2,  
 height\_shift\_range = 0.2,  
 shear\_range = 0.2,  
 zoom\_range = 0.2,  
 horizontal\_flip = TRUE  
)  
  
test\_datagen <- image\_data\_generator(rescale = 1/255)  
  
train\_generator\_1000 <- flow\_images\_from\_directory(  
 train\_dir\_1000,  
 datagen,  
 target\_size = c(150, 150),  
 batch\_size = 20,  
 class\_mode = "binary"  
)  
  
validation\_generator <- flow\_images\_from\_directory(  
 validation\_dir,  
 test\_datagen,  
 target\_size = c(150, 150),  
 batch\_size = 20,  
 class\_mode = "binary"  
)  
  
history\_1000\_reg <- model\_1000\_reg %>% fit\_generator(  
 train\_generator\_1000,  
 steps\_per\_epoch = 50,  
 epochs = 50,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)  
  
plot(history\_1000\_reg)

plot(history\_1000\_reg)

#Setting up 2000 training sample directory  
  
base\_dir\_2000 <- "~/Downloads/cats\_and\_dogs\_2000"  
dir.create(base\_dir\_2000)  
  
train\_dir\_2000 <- file.path(base\_dir\_2000, "train")  
dir.create(train\_dir\_2000)  
  
  
train\_cats\_dir\_2000 <- file.path(train\_dir\_2000, "cats")  
dir.create(train\_cats\_dir\_2000)  
  
train\_dogs\_dir\_2000 <- file.path(train\_dir\_2000, "dogs")  
dir.create(train\_dogs\_dir\_2000)  
  
  
fnames <- paste0("cat.", c(1:500,1001:1500), ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),   
 file.path(train\_cats\_dir\_2000))   
  
fnames <- paste0("dog.", c(1:500,1001:1500), ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),  
 file.path(train\_dogs\_dir\_2000))

model\_2000\_reg <- 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")   
   
model\_2000\_reg %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-4),  
 metrics = c("acc")  
)  
  
  
train\_generator\_2000 <- flow\_images\_from\_directory(  
 train\_dir\_2000,  
 datagen,  
 target\_size = c(150, 150),  
 batch\_size = 20,  
 class\_mode = "binary")  
  
  
  
history\_2000\_reg <- model\_2000\_reg %>% fit\_generator(  
 train\_generator\_2000,  
 steps\_per\_epoch = 50,  
 epochs = 50,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)  
  
plot(history\_2000\_reg)

#Setup training directory for 4000 training samples  
  
base\_dir\_4000 <- "~/Downloads/cats\_and\_dogs\_4000"  
dir.create(base\_dir\_4000)  
  
train\_dir\_4000 <- file.path(base\_dir\_4000, "train")  
dir.create(train\_dir\_4000)  
  
  
train\_cats\_dir\_4000 <- file.path(train\_dir\_4000, "cats")  
dir.create(train\_cats\_dir\_4000)  
  
train\_dogs\_dir\_4000 <- file.path(train\_dir\_4000, "dogs")  
dir.create(train\_dogs\_dir\_4000)  
  
  
fnames <- paste0("cat.", c(1:500,1001:2500), ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),   
 file.path(train\_cats\_dir\_4000))   
  
fnames <- paste0("dog.", c(1:500,1001:2500), ".jpg")  
file.copy(file.path(original\_dataset\_dir, fnames),  
 file.path(train\_dogs\_dir\_4000))

model\_4000\_reg <- 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")   
   
model\_4000\_reg %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-4),  
 metrics = c("acc")  
)  
  
  
train\_generator\_4000 <- flow\_images\_from\_directory(  
 train\_dir\_4000,  
 datagen,  
 target\_size = c(150, 150),  
 batch\_size = 40,  
 class\_mode = "binary")  
  
  
  
history\_4000\_reg <- model\_4000\_reg %>% fit\_generator(  
 train\_generator\_4000,  
 steps\_per\_epoch = 100,  
 epochs = 50,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)  
  
plot(history\_4000\_reg)

plot(history\_4000\_reg)

#Set-up Pretrained Netowork  
library(keras)  
  
conv\_base <- application\_vgg16(  
 weights = "imagenet",  
 include\_top = FALSE,  
 input\_shape = c(150, 150, 3)  
)

#Creating the base for pretrained convolution base  
model.pt.1000 <- keras\_model\_sequential() %>%   
 conv\_base %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 256, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")

freeze\_weights(conv\_base)

model.pt.1000 %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-5),  
 metrics = c("acc")  
)  
  
history.pt.1000 <- model.pt.1000 %>% fit\_generator(  
 train\_generator\_1000,  
 steps\_per\_epoch = 50,  
 epochs = 25,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)  
  
plot(history.pt.1000)

unfreeze\_weights(conv\_base, from = "block3\_conv1")

model.pt.1000 %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-5),  
 metrics = c("acc")  
)  
  
history.pt.1000 <- model.pt.1000 %>% fit\_generator(  
 train\_generator\_1000,  
 steps\_per\_epoch = 50,  
 epochs = 25,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)

model.pt.2000 <- keras\_model\_sequential() %>%   
 conv\_base %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 256, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
freeze\_weights(conv\_base)  
  
unfreeze\_weights(conv\_base, from = "block3\_conv1")  
  
model.pt.2000 %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-5),  
 metrics = c("acc")  
)  
  
history.pt.2000 <- model.pt.2000 %>% fit(  
 train\_generator\_2000,  
 steps\_per\_epoch = 50,  
 epochs = 25,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)

model.pt.4000 <- keras\_model\_sequential() %>%   
 conv\_base %>%   
 layer\_flatten() %>%   
 layer\_dense(units = 256, activation = "relu") %>%   
 layer\_dense(units = 1, activation = "sigmoid")  
  
freeze\_weights(conv\_base)  
  
unfreeze\_weights(conv\_base, from = "block3\_conv1")  
  
model.pt.4000 %>% compile(  
 loss = "binary\_crossentropy",  
 optimizer = optimizer\_rmsprop(lr = 1e-5),  
 metrics = c("acc")  
)  
  
history.pt.4000 <- model.pt.4000 %>% fit(  
 train\_generator\_4000,  
 steps\_per\_epoch = 50,  
 epochs = 25,  
 validation\_data = validation\_generator,  
 validation\_steps = 25  
)

test\_generator <- flow\_images\_from\_directory(  
 test\_dir,  
 test\_datagen,  
 target\_size = c(150, 150),  
 batch\_size = 10,  
 class\_mode = "binary"  
)  
  
model.pt.4000 %>% evaluate\_generator(test\_generator, steps = 50)

#Model Prediction

model\_1000\_reg%>% evaluate\_generator(test\_generator, steps = 50)

model\_2000\_reg%>% evaluate\_generator(test\_generator, steps = 50)

model\_4000\_reg %>% evaluate\_generator(test\_generator, steps = 50)

model.pt.1000 %>% evaluate\_generator(test\_generator, steps = 50)

model.pt.2000 %>% evaluate\_generator(test\_generator, steps = 50)

model.pt.4000 %>% evaluate\_generator(test\_generator, steps = 50)