Code By: Xan Lamoreux

Cats and Dogs Classification

Advanced Machine Learning

Building the model

```
model <- keras_model_sequential() %>%
  c(3, 3), activation = "relu",
  150, 3)) %>%
```

```
  layer_conv_2d(filters = 32,                    2, 2)) %>%  c(3, 3), activation = "relu")  %>%
kernel_size =                                            2, 2)) %>% kernel_size = c(3, 3), activation = "relu")  % >
input_shape = c(150,
layer_max_pooling_2d(pool_size                           2, 2)) %>% kernel_size = c(3, 3), activation = "relu")  % >
= c(   layer_conv_2d(filters = 64,
kernel_size =                                  2, 2)) %>%
layer_max_pooling_2d(pool_size                     "relu") %>%
= c(   layer_conv_2d(filters =                  "sigmoid")
128,
%
layer_max_pooling_2d(pool_size
= c(   layer_conv_2d(filters = 128,
%

layer_max_pooling_2d(pool_size                 ## _________________________________________________________________
= c(   layer_flatten() %>%
  layer_dense(units = 512,                     ## Layer (type)                 Output Shape              Param
activation =    layer_dense(units =            ##
1, activation = ## Loaded                      ================================================================
Tensorflow version 2.6.0
                                               ## conv2d_3 (Conv2D)             (None, 148, 148, 32)      896
summary(model)                                 _________________________________________________________________

## Model: "sequential"                         ## max_pooling2d_3 (MaxPooling2D)  (None, 74, 74, 32)       0
                                               _________________________________________________________________
_____
                                               ## conv2d_2 (Conv2D)             (None, 72, 72, 64)        18496
#                                              _________________________________________________________________

======                                         ## max_pooling2d_2 (MaxPooling2D)  (None, 36, 36, 64)       0
                                               _________________________________________________________________

_____                                          ## conv2d_1 (Conv2D)             (None, 34, 34, 128)       73856
                                               ## _________________________________________________________________

_____


_____


_____


_____
## max_pooling2d_1 (MaxPooling2D)  (None, 17, 17, 128)       0
```

```
## _________________________________________________________________ _____
## conv2d (Conv2D)                 (None, 15, 15, 128)           147584
## _________________________________________________________________ _____
## max_pooling2d (MaxPooling2D)    (None, 7, 7, 128)             0
## _________________________________________________________________ _____
## flatten (Flatten)               (None, 6272)                  0
## _____________________________________________________________________
_____
## dense_1 (Dense)                 (None, 512)                   321177 6
## _________________________________________________________________ _____
## dense (Dense)                   (None, 1)                     513
## ================================================================= ======
## Total params: 3,453,121
## Trainable params: 3,453,121
## Non-trainable params: 0
## _____________________________________________________________________
_____
```

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

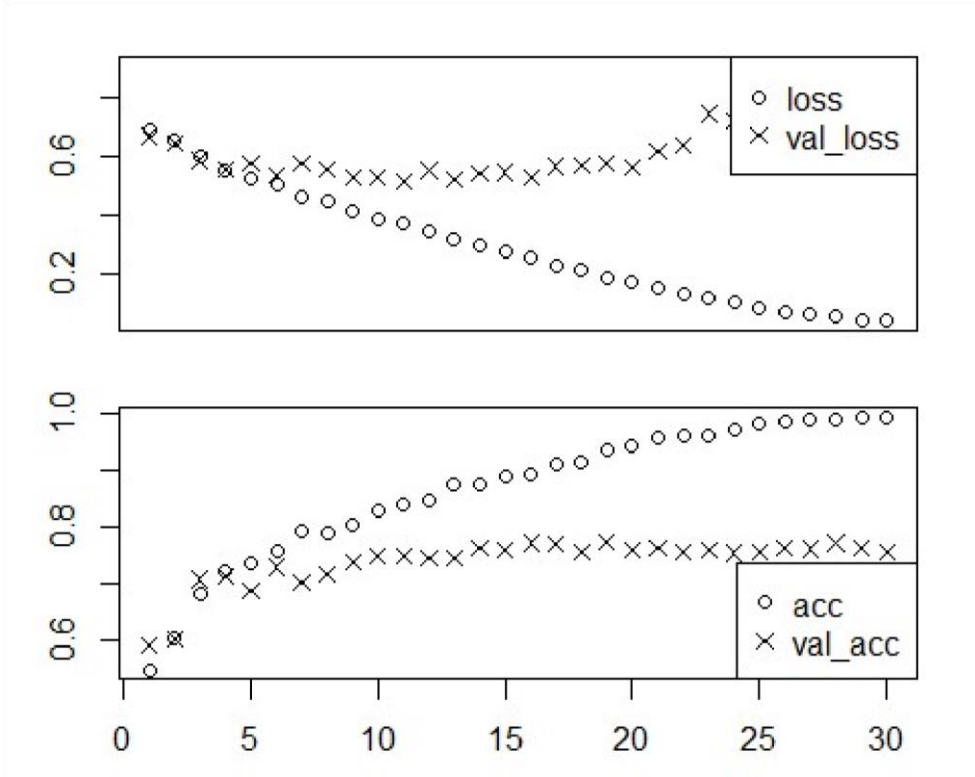
```
## Warning in backcompat_fix_rename_lr_to_learning_rate(...): the `lr` argume nt has
## been renamed to `learning_rate`.
```

Train the model

```
history <- model %>% fit_generator(
  train_generator,   steps_per_epoch =
100,   epochs = 30,
  validation_data    =       validation_generator,
validation_steps = 50 )
```

```
## generators.
```

```
## Warning in fit_generator(., train_generator,
steps_per_epoch = 100, epochs
## = 30, : `fit_generator` is deprecated. Use `fit`
instead, it now accept
```



Save the Model

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

Building the model (Using DropOut to reduce overfitting)

```
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")
## Loaded Tensorflow version 2.6.0

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

## Warning in backcompat_fix_rename_lr_to_learning_rate(...): the `lr` argume
nt has
## been renamed to `learning_rate`.
summary(model)

## Model: "sequential"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param
##
=================================================================
## conv2d_3 (Conv2D)            (None, 148, 148, 32)      896       ##
_________________________________________________________________
## max_pooling2d_3 (MaxPooling2D)  (None, 74, 74, 32)        0
```

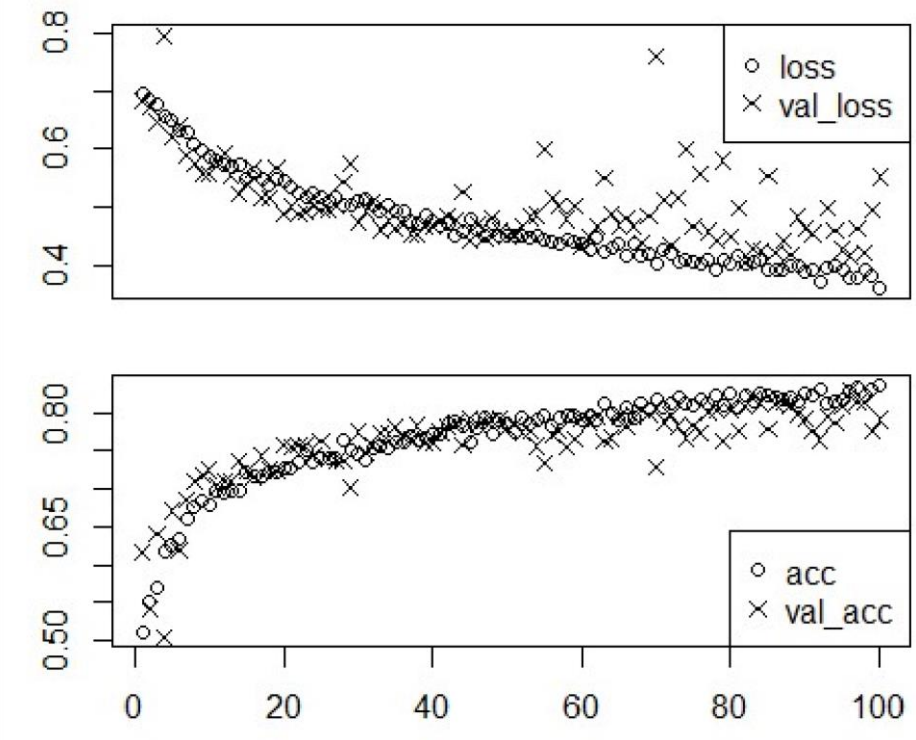
```
## _________________________________________________________________
_____
## conv2d_2 (Conv2D)                (None, 72, 72, 64)            18496
## _________________________________________________________________ _____
## max_pooling2d_2 (MaxPooling2D)      (None, 36, 36, 64)            0
## _________________________________________________________________ _____
## conv2d_1 (Conv2D)                (None, 34, 34, 128)           73856
## _________________________________________________________________ _____
## max_pooling2d_1 (MaxPooling2D)      (None, 17, 17, 128)           0
## _________________________________________________________________ _____
## conv2d (Conv2D)                  (None, 15, 15, 128)           147584
## _________________________________________________________________ _____
## max_pooling2d (MaxPooling2D)        (None, 7, 7, 128)             0
## _________________________________________________________________ _____
## flatten (Flatten)                (None, 6272)                  0
## _________________________________________________________________ _____
## dropout (Dropout)                (None, 6272)                  0
## _________________________________________________________________ _____
## dense_1 (Dense)                  (None, 512)                   321177
6
## _________________________________________________________________ _____
## dense (Dense)                    (None, 1)                     513
## ================================================================= ======
## Total params: 3,453,121
## Trainable params: 3,453,121
## Non-trainable params: 0
## _________________________________________________________________ _____
```

Train the model

```
history <- model %>% fit_generator(
```

```
  train_generator,   steps_per_epoch =
100,   epochs = 100,
  validation_data   =        validation_generator,
validation_steps = 50 )



## generators.
```

```
## Warning in fit_generator(., train_generator,
steps_per_epoch = 100, epochs
## = 100, : `fit_generator` is deprecated. Use `fit`
instead, it now accept
```




```
results <- model %>% evaluate(validation_generator)
 results

##      loss       acc
## 0.5520354 0.7910000
```

Save the Model

```
 model %>% save_model_hdf5("cats_and_dogs_small_2.h5")
```

USING MORE TRAINING DATA (2000 images for training for each category)

```
cat("total training cat images:", length(list.files(train_cats_dir)), "\n")

## total training cat images: 2000 cat("total training dog images:", length(list.files(train_dogs_dir)),
"\n")

## total training dog images: 2000
cat("total validation cat images:" , length(list.files(validation_cats_dir)), "\n")
## total validation cat images: 500
cat("total validation dog images:" , length(list.files(validation_dogs_dir)), "\n")
## total validation dog images: 500 cat("total test cat images:", length(list.files(test_cats_dir)),
"\n")

## total test cat images: 500 cat("total test dog images:", length(list.files(test_dogs_dir)), "\n")

## total test dog images: 500
```

Building the model (Using DropOut to reduce overfitting)

```
model <- keras_model_sequential() %>%                                    c(3, 3), activation = "relu",
          layer_conv_2d(filters = 32, kernel_size =          input_shape = c(150,150, 3)) %>%
 layer_max_pooling_2d(pool_size = c(   layer_conv_2d(filters = 64, kernel_size =2, 2)) %>%  c(3, 3), activation =
                                                                                              "relu")  %>%

 layer_max_pooling_2d(pool_size = c(2, 2)) %>%
```

```
 layer_conv_2d(filters = 128, kernel_size = c(
%   layer_max_pooling_2d(pool_size = c(2, 2))  %>%   layer_conv_2d(filters = 128, kernel_size = c(
%
 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"

## Loaded Tensorflow version 2.6.0

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

## Warning in backcompat_fix_rename_lr_to_learning_rate(...): the `lr` argume nt has
## been renamed to `learning_rate`.

summary(model)

## Model: "sequential"
## _________________________________________________________________ _______
## Layer (type)                 Output Shape              Param
#
##
=================================================================
======
## conv2d_3 (Conv2D)            (None, 148, 148, 32)      896
## _________________________________________________________________ _______
## max_pooling2d_3 (MaxPooling2D)  (None, 74, 74, 32)        0
## _________________________________________________________________ _______
## conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
## _________________________________________________________________ _______
## max_pooling2d_2 (MaxPooling2D)  (None, 36, 36, 64)        0
## _________________________________________________________________ _______
## conv2d_1 (Conv2D)            (None, 34, 34, 128)       73856
## _________________________________________________________________ _______
## max_pooling2d_1 (MaxPooling2D)  (None, 17, 17, 128)       0
## _________________________________________________________________ _______
```

```
3,              3),
activation    =
"relu")  % >
3,              3),
activation    =
"relu")  % >
 %>%
)
```

```
## conv2d (Conv2D)              (None, 15, 15, 128)       147584
```

```
## _________________________________________________________________ _____
## max_pooling2d (MaxPooling2D)       (None, 7, 7, 128)             0
## _________________________________________________________________ _____
## flatten (Flatten)                  (None, 6272)                  0
## _________________________________________________________________ _____
## dropout (Dropout)                  (None, 6272)                  0
## _____________________________________________________________________
_____
## dense_1 (Dense)                    (None, 512)                   321177 6
## _________________________________________________________________ _____
## dense (Dense)                      (None, 1)                     513
## ================================================================= ======
## Total params: 3,453,121
## Trainable params: 3,453,121
## Non-trainable params: 0
## _____________________________________________________________________
_____
```

Creating Data Generator

using Data Augmentation Technique 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(
train_generator <  train_dir,   datagen,
  target_size = c(150, 150),   batch_size
= 20,   class_mode = "binary"
)
validation_generator <  validation_dir,
test_datagen,
```

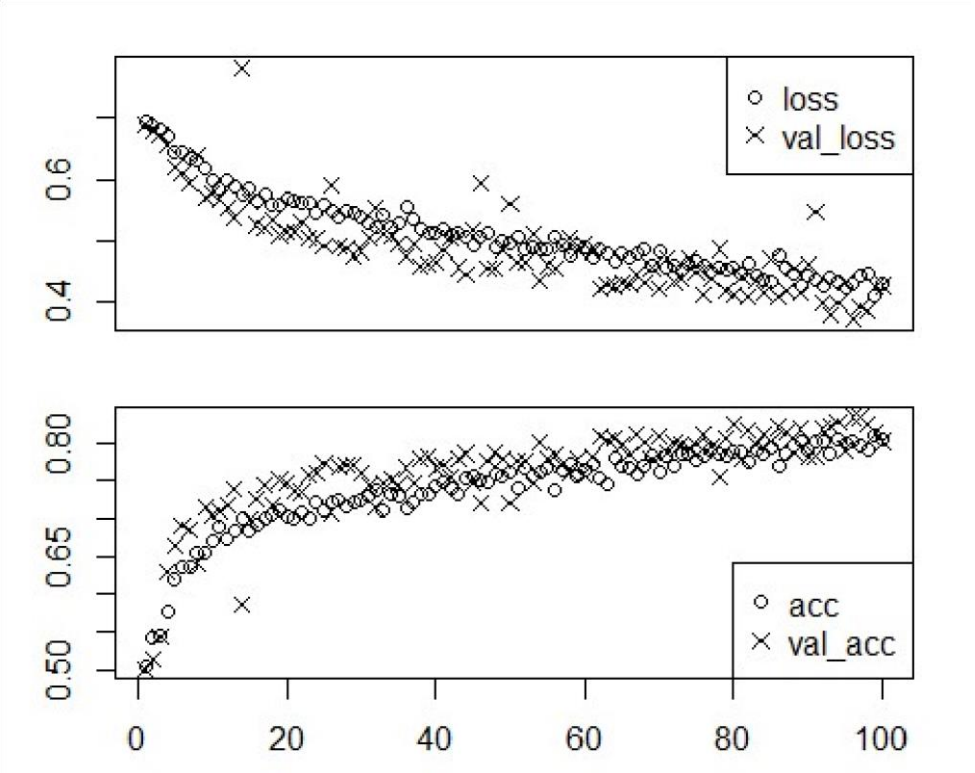
```
        rescale = 1/255)
flow_images_from_directory(
```

```
flow_images_from_directory(
  target_size = c(150, 150),
```

```
  batch_size = 20,   class_mode =
"binary"
)



batch <
generator_next(train_generator)
str(batch)

## List of 2

##  $ : num [1:20(1d)] 1 0 1 0 1 1 1
1 0 1 ...
##  $ : num [1:20, 1:150, 1:150, 1:3] 0.953 0.56 0.504 0.466 0.145
...
```

Train the model

```
history <- model %>% fit_generator(
  train_generator,   steps_per_epoch =
100,   epochs = 100,
  validation_data    =        validation_generator,
validation_steps = 50 )



## generators.
## Warning in fit_generator(., train_generator,
steps_per_epoch = 100, epochs
## = 100, : `fit_generator` is deprecated. Use `fit`
instead, it now accept
```




```
results <- model %>% evaluate(validation_generator)
 results

##      loss       acc
## 0.4275039 0.8020000
```

Save the Model

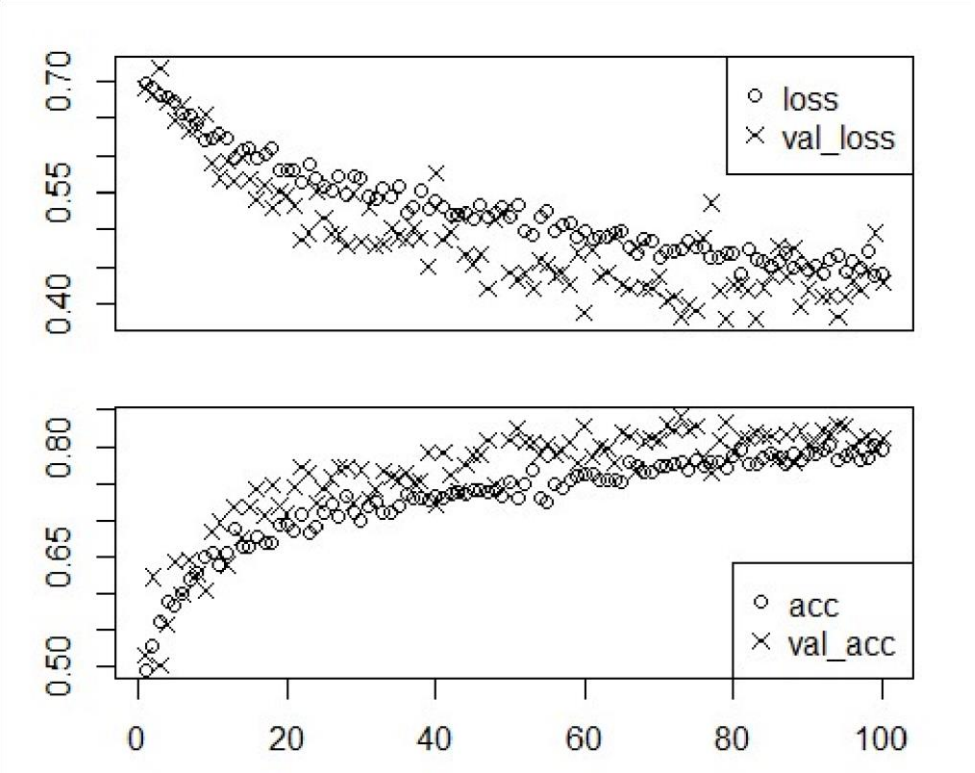
```
 model %>% save_model_hdf5("cats_and_dogs_medium_01.h5")
```

USING MORE TRAINING DATA (2000 images for training for each category)

```
cat("total training cat images:", length(list.files(train_cats_dir)), "\n")
```

```
## total training cat images: 5000 cat("total training dog images:", length(list.files(train_dogs_dir)),
"\n")

## total training dog images: 5000

cat("total validation cat images:" , length(list.files(validation_cats_dir)), "\n")

## total validation cat images: 1000

cat("total validation dog images:" , length(list.files(validation_dogs_dir)), "\n")

## total validation dog images: 1000 cat("total test cat images:", length(list.files(test_cats_dir)),
"\n")

## total test cat images: 1500 cat("total test dog images:", length(list.files(test_dogs_dir)), "\n")

## total test dog images: 1500
```

```
  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(
train_generator <  train_dir,   datagen,
  target_size = c(150, 150),   batch_size
= 20,   class_mode = "binary"
)
validation_generator <  validation_dir,
test_datagen,
  target_size = c(150, 150),   batch_size
= 20,   class_mode = "binary"
)
```



Save the Model

```
model %>% save_model_hdf5("cats_and_dogs_lg_01.h5")
```

Reshape the features

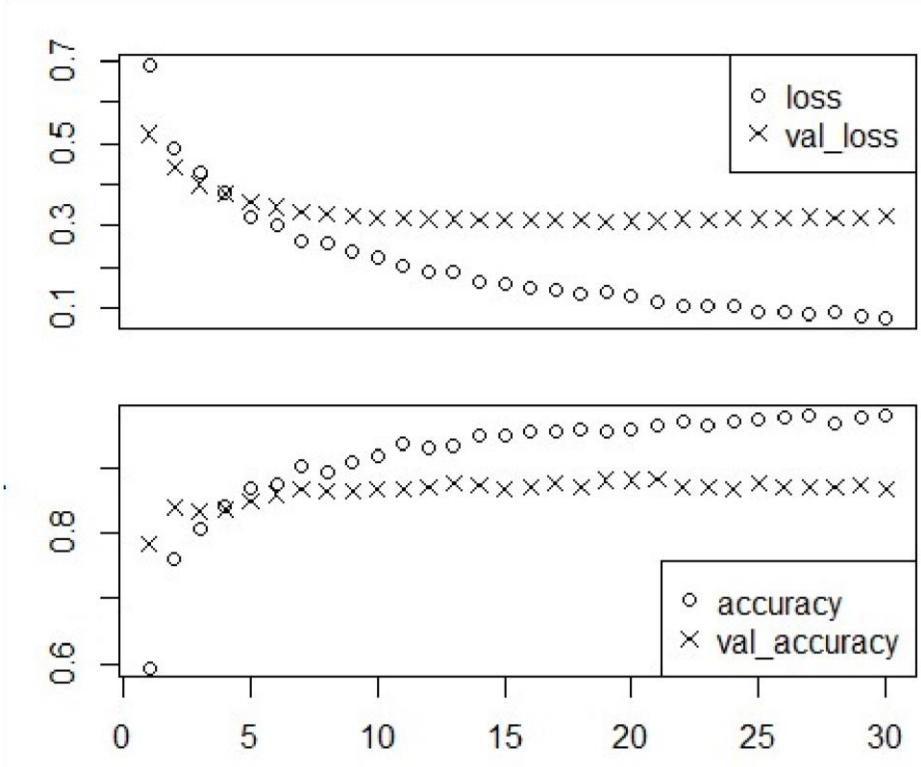
```
reshape_features <- function(features) {

 array_reshape(features, dim =

                                                    c(nrow(features), 4 * 4 * 512))

}

                                                    reshape_features(validation$features)

train$features < reshape_features(train$features)
validation$features <-

test$features < reshape_features(test$features)
```

creating the model

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

 layer_dense(units = 256, activation =    "relu",
input_shape = 4 * 4 *   layer_dropout(rate = 0.5)
%>%   layer_dense(units = 1, activation =

                                          512) %>%

                                           "sigmoid")

model %>% compile(

                                          2e-5),

 optimizer = optimizer_rmsprop(lr =   loss =    ## Warning in
"binary_crossentropy",   metrics = c("accuracy")   backcompat_fix_rename_lr_to_learning_rate(...): the
                                          `lr` argume
)

nt has

## been renamed to `learning_rate`.

history <- model %>% fit                  (

 train$features, train$labels,          list(validation$features, validation$labels)
epochs = 30,   batch_size = 20,
validation_data =

)
```

Plot the history

```
str(history )
## List of 2
## $ params :List of 3
## ..$ verbose: int 1
## $ metrics:List of 4
## ..$ epochs : int 30
## ..$ steps : int 50
## ..$ loss : num [1:30] 0.688 0.488 0.429 0.38 0.32 ...
## ..$ accuracy : num [1:30] 0.594 0.761 0.807 0.842 0.87 ...
## ..$ val_loss : num [1:30] 0.522 0.441 0.399 0.377 0.359 ...
## ..$ val_accuracy: num [1:30] 0.784 0.84 0.834 0.836 0.85 ... ## - attr(*, "class")= chr "keras_training_history"
plot(history)
```



Save the Model

```
model %>% save_model_hdf5("cats_and_dogs_pretrained_04_1.h5")
```

==========

========== ===

————————————-TASK # 04.2————————————

==========

========== ===

Base Model

```
library(keras)

conv_base <- application_vgg16(
weights = "imagenet",
include_top = FALSE,
input_shape = c(150, 150, 3)
)

conv_base

## Loaded Tensorflow version 2.6.0

## Model
## Model: "vgg16"
## ________________________________________________________________________________
## Layer (type)                  Output Shape                 Param
## #
## ================================================================================
## input_1 (InputLayer)          [(None, 150, 150, 3)]        0
## ________________________________________________________________________________
## block1_conv1 (Conv2D)         (None, 150, 150, 64)         1792
## ________________________________________________________________________________
## block1_conv2 (Conv2D)         (None, 150, 150, 64)         36928
## ________________________________________________________________________________
```

_

block1_pool (MaxPooling2D) (None, 75, 75, 64) 0

_____

block2_conv1 (Conv2D) (None, 75, 75, 128) 73856

_____ _____

block2_conv2 (Conv2D) (None, 75, 75, 128) 147584

_____ _____

block2_pool (MaxPooling2D) (None, 37, 37, 128) 0

_____ _____

block3_conv1 (Conv2D) (None, 37, 37, 256) 295168

_____ _____

block3_conv2 (Conv2D) (None, 37, 37, 256) 590080

_____ _____

block3_conv3 (Conv2D) (None, 37, 37, 256) 590080

0 _____

block3_pool (MaxPooling2D) (None, 18, 18, 256) 0

_____

8 ## _____

block4_conv1 (Conv2D) (None, 18, 18, 512) 118016

_____

8 ## _____

block4_conv2 (Conv2D) (None, 18, 18, 512) 235980

_____ _____

block4_conv3 (Conv2D) (None, 18, 18, 512) 235980

8 _____

block4_pool (MaxPooling2D) (None, 9, 9, 512) 0

_____

8 ## _____

block5_conv1 (Conv2D) (None, 9, 9, 512) 235980

_____

block5_conv2 (Conv2D) (None, 9, 9, 512) 235980

_____

block5_conv3 (Conv2D) (None, 9, 9, 512) 235980

```
## ____________________________________________________________________________
```

```
_____

## block5_pool (MaxPooling2D)        (None, 4, 4, 512)            0

## ================================================================= ======

## Total params: 14,714,688

## Trainable params: 14,714,688

## Non-trainable params: 0

## _____

_____

base_dir <- "E:/Assignments and Tasks/Xan - Advance Machine Learning/Module 0

4/cat vs dog - dataset/md/"

train_dir <- file.path(base_dir, "train") validation_dir <- file.path(base_dir, "validation") test_dir <-
file.path(base_dir, "test")



datagen <- image_data_generator(rescale = 1/255) batch_size <- 20



extract_features <- function(directory, sample_count) {   features <- array(0, dim = c(sample_count, 4, 4,
512))   labels <- array(0, dim = c(sample_count))   generator <- flow_images_from_directory(     directory
= directory,     generator = datagen,     target_size = c(150, 150),     batch_size = batch_size,     class_mode
= "binary"

 )   i <- 0   while(TRUE) {

   batch <- generator_next(generator)     inputs_batch <- batch[[1]]     labels_batch <- batch[[2]]

   features_batch <- conv_base %>% predict(inputs_batch)     index_range <- ((i * batch_size)+1):((i + 1) *
batch_size)     features[index_range,,,] <- features_batch     labels[index_range] <- labels_batch     i <- i +
1

   if (i * batch_size >= sample_count)       break
```

```
  }   list(

    features = features,     labels = labels

  )

}
```

```
train <- extract_features(train_dir, 2000)

validation < extract_features(validation_dir, test <- 500)
extract_features(test_dir, 500)
```

Reshape the features

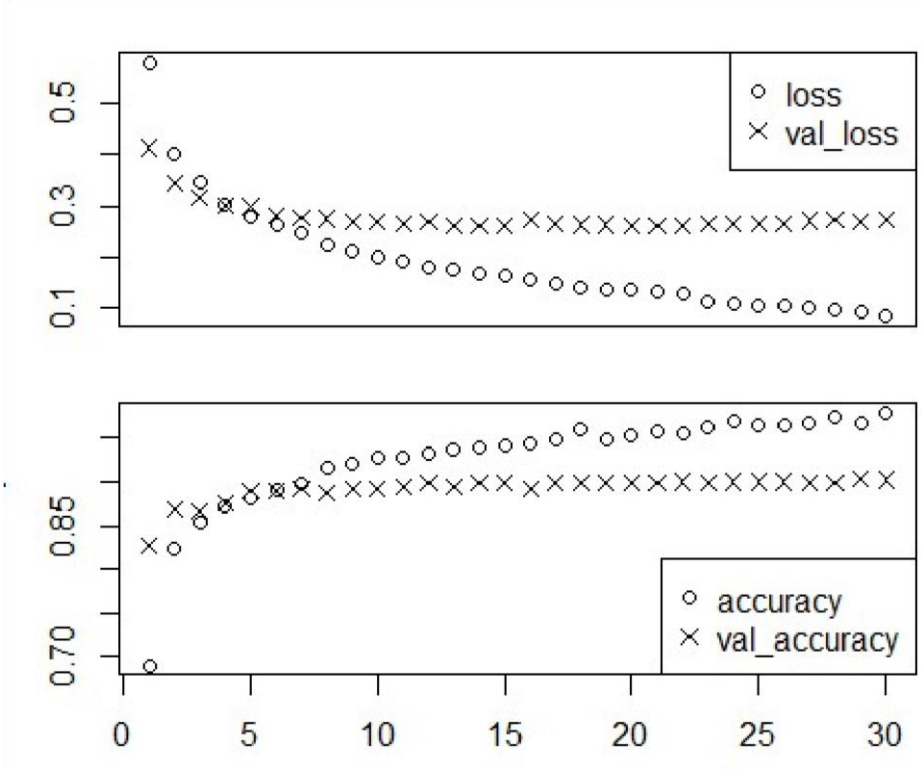
```
reshape_features <- function(features) {

 array_reshape(features, dim =

                                  c(nrow(features), 4 * 4 * 512))

}

                                  reshape_features(validation$features)

train$features < reshape_features(train$features)
validation$features <-

test$features < reshape_features(test$features)
```

creating the model

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

 layer_dense(units = 256, activation =                "relu",
input_shape = 4 * 4 *   layer_dropout(rate = 0.5)    512) %>%
%>%   layer_dense(units = 1, activation =              "sigmoid")



model %>% compile(                                   2e-5),

 optimizer = optimizer_rmsprop(lr =   loss =         ## Warning in
"binary_crossentropy",   metrics = c("accuracy")     backcompat_fix_rename_lr_to_learning_rate(...): the
                                                     `lr` argume
)
                                                      list(validation$features, validation$labels)
nt has

## been renamed to `learning_rate`.

history <- model %>% fit(   train$features,
train$labels,   epochs = 30,   batch_size = 20,
validation_data =

)
```

Plot the history

```
str(history          )

## List of 2          ##   ..$ epochs : int 30

##  $ params          ##   ..$ steps  : int 100
:List of 3
                      ##   ..$ loss       : num [1:30] 0.579 0.401 0.347 0.304 0.279 ... ##   ..$ accuracy    : num
##   ..$ verbose:     [1:30] 0.69 0.822 0.853 0.872 0.881 ...
int 1
                      ##   ..$ val_loss    : num [1:30] 0.413 0.343 0.316 0.299 0.298 ...
##  $
metrics:List of 4

##   ..$ val_accuracy: num [1:30] 0.826 0.868 0.866 0.876 0.89 ...

##  - attr(*, "class")= chr "keras_training_history" plot(history)
```



Save the Model

```
model %>% save_model_hdf5("cats_and_dogs_pretrained_04_2.h5")
```

Tokenizing the text of the raw IMDB data

```
maxlen <- 150
training_samples <- 100
validation_samples <- 10000 10000
max_words <-
tokenizer <- text_tokenizer(num_words = max_words) %>%
  fit_text_tokenizer(texts)
## Loaded Tensorflow
version 2.6.0
sequences
<word_index =
cat("Found"
texts_to_sequences(tokenizer, texts)  tokenizer$word_index
, length(word_index), "unique tokens.\n")
## Found 88584 unique tokens.
```

```
data < pad_sequences(sequences, labels <- as.array(labels) cat("Shape of data tensor:"
## Shape of data tensor: 25000 150
cat('Shape of label tensor:' ## Shape of label tensor: 25000

indices <- sample(1: nrow(data)) training_indices <- indices[1 validation_indices <-

x_train < data[training_indices,] y_train < labels[training_indices] x_val < data[validation_indices,] y_val < labels[validation_indices]
```

```
maxlen = maxlen)

, dim(data), "\n")

, dim(labels), "\n")

:training_samples]
 indices[(training_samples + 1):
                    (training_samples + validation_samples)]
```

Pre-trained Glove Embedding Model

Parsing the GloVe word-embeddings file

```
glove_dir = "E:/Assignments and Tasks/Xan - Advance Machine Learning/Module 05/Data/"
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt")) embeddings_index <- new.env(hash = TRUE, parent = emptyenv()) for (i in 1:length(lines)) {   line <- lines[[i]]
  values <- strsplit(line, " ")[[1]]   word <- values[[1]]
  embeddings_index[[word]] <- as.double(values[-1])
} cat("Found", length(embeddings_index), "word vectors.\n")

## Found 400000 word vectors.
```

Preparing the GloVe word-embeddings matrix

```
embedding_dim <- 100                    array(0, c(max_words,
embedding_matrix <for                embedding_dim))  names(word_index)) {
(word in
  index                    <
word_index[[word]]       if   <- embeddings_index[[word]]
(index  <  max_words)  {   (!is.null(embedding_vector))
embedding_vector    if                  1,] <- embedding_vector

embedding_matrix[index+
  }
}
```

Model definition

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words,
                  output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
summary(model)
```

```
## Model: "sequential"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param
##
## =================================================================
## embedding (Embedding)        (None, 150, 100)          100000
## _________________________________________________________________
## flatten (Flatten)            (None, 15000)             0
## _________________________________________________________________
## dense_1 (Dense)              (None, 32)                480032
## _________________________________________________________________
## dense (Dense)                (None, 1)                 33
##
## =================================================================
## Total params: 1,480,065
## Trainable params: 1,480,065
## Non-trainable params: 0
## _________________________________________________________________
```

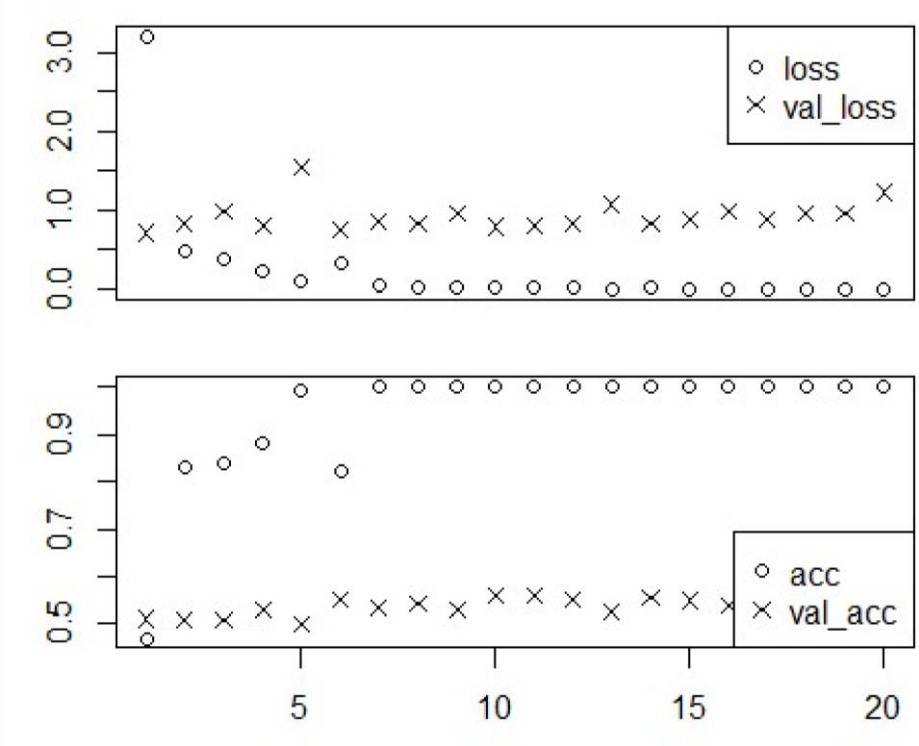
Loading pretrained word embeddings into the embedding layer

```
get_layer(model, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()
```

Training and evaluation

```
model %>% compile(
  "rmsprop",
```

```
  optimizer =   loss "binary_crossentropy", )
=
  metrics = c(          model %>% fit(
"acc"
) history <
x_train, y_train,
epochs = 20,
batch_size = 32 ,
  validation_data = list(x_val, y_val)
)
save_model_weights_hdf5(model,              "pre_trained_glove_model.h5")

plot(history)
```

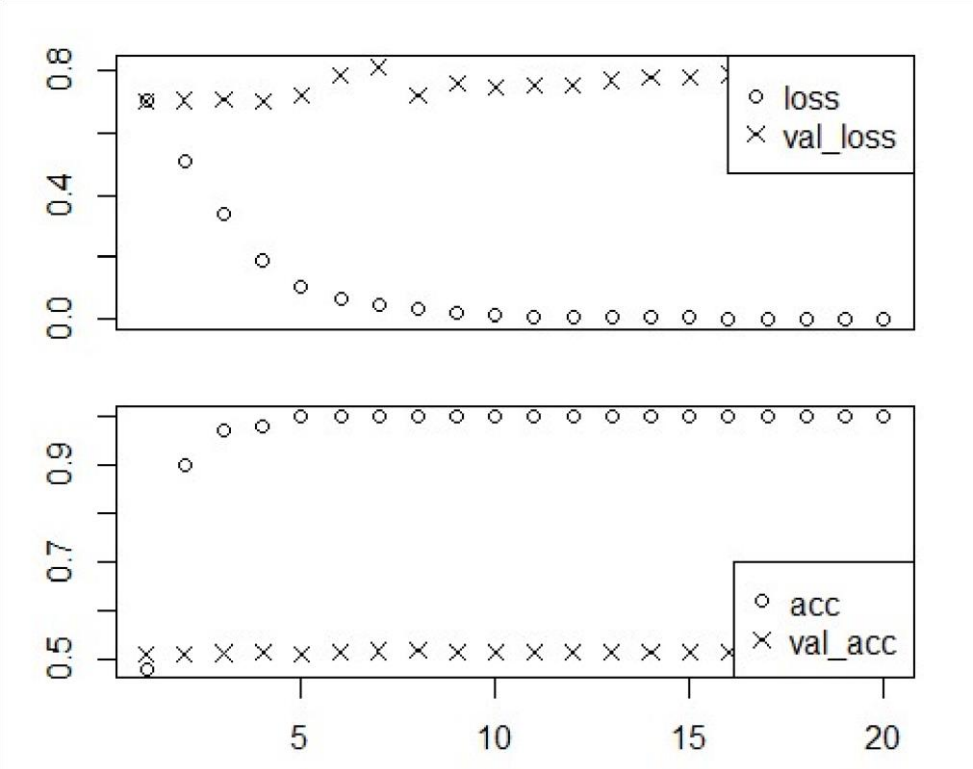


Training the same model without pretrained word embeddings

```
model2 <- keras_model_sequential() %>%              output_dim = embedding_dim,
```

```
  layer_embedding(input_dim =  max_words,          maxlen) %>%
input_length =   layer_flatten() %>%
  layer_dense(units = 32, activation =            "relu") %>%
layer_dense(units = 1, activation =              "sigmoid")


model2 %>% compile(   optimizer =
"rmsprop",   loss = "binary_crossentropy",
metrics = c("acc")
)
history2 <- model2 %>% fit(
x_train, y_train,   epochs = 20,
batch_size = 32,
  validation_data =  list(x_val, y_val) )
```

```
plot(history2)
```



Tokenizing the data of the test set

```
test_dir <- file.path(imdb_dir, "test")
```

```
labels <- c() texts <- c()
        for (label_type in c("neg", "pos")) {   label <-
             switch(label_type, neg = 0,   dir_name <
          file.path(test_dir, label_type)   for (fname in
    list.files(dir_name,                          full.names =
   texts < c(texts, readChar(fname,
file.info(fname)$size))
   labels <- c(labels, label)
 } }
sequences < texts_to_sequences(tokenizer, texts)
x_test <- pad_sequences(sequences, maxlen =
y_test <- as.array(labels)

results2 < model2 %>% evaluate(x_test, y_test)
results2

##      loss       acc
## 0.7763946 0.5173600
```

pos = 1)

pattern = glob2rx("*.txt"), TRUE)) {

maxlen)

Evaluating the model on the test set

```
model        %
%>             load_model_weights_hdf5("pre_trained_glove_model.h5") %>%   evaluate(x_test,
               y_test)

               ##     loss      acc
               ## 1.208684 0.530280
```

(Using RNN & Increase training samples=10000)

Processing the labels of the raw IMDB data

```
library(keras)
```

```
imdb_dir <- 5
/Data/aclImdb/"
train_dir <labels
<- c() texts <- c()
for  (label_type
label <- switch
dir_name <  for
(fname in

   texts <    labels
                  <-
 }
}
```

```
                "E:/Assignments and Tasks/Xan - Advance Machine Learning/Module 0
      file.path(imdb_dir, "train")


  in c("neg", "pos")) {
     (label_type, neg = 0, pos = 1)  file.path(train_dir, label_type)
list.files(dir_name, pattern = glob2rx("*.txt"), full.names = TRUE)) {
c(texts, readChar(fname, file.info(fname)$size))
c(labels, label)
```

Tokenizing the text of the raw IMDB data

```
maxlen <- 500
training_samples      <- 10000
validation_samples      <- 10000 10000
max_words <-

                      text_tokenizer(num_words = max_words) %>%
tokenizer <-           fit_text_tokenizer(texts) ##

                     Loaded Tensorflow version

                     2.6.0
sequences
<word_index =
cat("Found"           texts_to_sequences(tokenizer, texts)  tokenizer$word_index
                     , length(word_index), "unique tokens.\n")

                     ## Found 88584 unique tokens.
data      <labels
<cat(                 pad_sequences(sequences, maxlen = maxlen)  as.array(labels)
                     "Shape of data tensor:", dim(data), "\n")

                     ## Shape of data tensor: 25000 500
cat(
                     'Shape of label tensor:', dim(labels), "\n")

                     ## Shape of label tensor: 25000
               indices
  <training_indices sample(1:nrow(data))
 validation_indices      <- indices[1:training_samples]
                           <- indices[(training_samples + 1):
                    (training_samples + validation_samples)]
```

```
x_train <- data[training_indices,] y_train <-
labels[training_indices] x_val <-
data[validation_indices,] y_val <-
labels[validation_indices]
```

Pre-trained Glove Embedding Model

Parsing the GloVe word-embeddings file

```
glove_dir = "E:/Assignments and Tasks/Xan - Advance Machine Learning/Module 0
5/Data/"
lines <- readLines(file.path(glove_dir, "glove.6B.100d.txt")) embeddings_index <-
new.env(hash = TRUE, parent = emptyenv()) for (i in 1:length(lines)) {   line <-
lines[[i]]
  values <- strsplit(line, " ")[[1]]   word <- values[[1]]
  embeddings_index[[word]] <- as.double(values[-1])
} cat("Found", length(embeddings_index), "word vectors.\n")

## Found 400000 word vectors.
```

Preparing the GloVe word-embeddings matrix

```
embedding_dim <- 100                    array(0, c(max_words,
embedding_matrix <for     embedding_dim))  names(word_index)) {
(word in
  index                    <
word_index[[word]]      if   <- embeddings_index[[word]]
(index  <  max_words)  {   (!is.null(embedding_vector))
embedding_vector    if
                                        1,] <- embedding_vector

embedding_matrix[index+
  }
}
```

Model definition

```
model                        <- output_dim = embedding_dim,
keras_model_sequential()
%>%
```

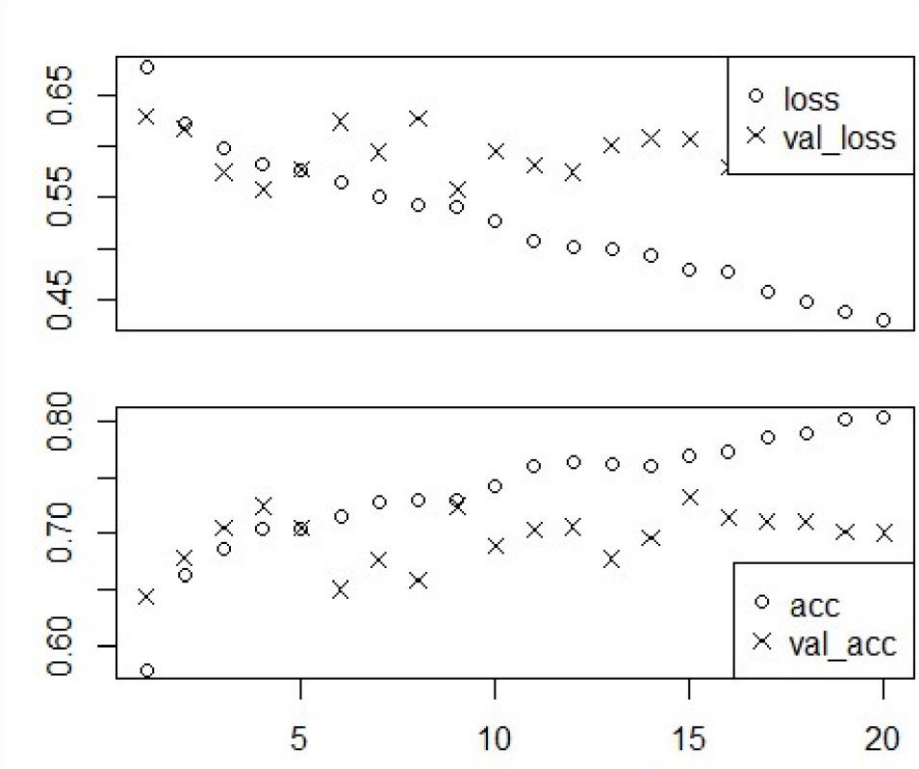
```
                                  maxlen) %>% return_sequences = TRUE) %>% return_sequences =
layer_embedding(input_dim TRUE) %>% return_sequences = TRUE) %>%
=  max_words,
input_length =                    "sigmoid")
layer_simple_rnn(units =
32,   layer_simple_rnn(units
= 32,
layer_simple_rnn(units =                                                                                    ##
32,   layer_simple_rnn(units _____________________________________________________________________________
= 32) %>%
layer_dense(units = 1,                          ## Layer (type)                Output Shape              Param
activation =
summary(model)

## Model: "sequential"


_____
#
0## =================================================================
  ======
  ## embedding (Embedding)            (None, 500, 100)          100000
  ## _____ _____
  ## simple_rnn_3 (SimpleRNN)         (None, 500, 32)           4256
  ## _____ _____
  ## simple_rnn_2 (SimpleRNN)         (None, 500, 32)           2080
  ## _____ _____
  ## simple_rnn_1 (SimpleRNN)         (None, 500, 32)           2080
  ## _____ _____
  ## simple_rnn (SimpleRNN)           (None, 32)                2080
  ## _____ _____
  ## dense (Dense)                    (None, 1)                 33
  ## ================================================================= ======
  ## Total params: 1,010,529
  ## Trainable params: 1,010,529
  ## Non-trainable params: 0
  ## _____ _____
```

Loading pretrained word embeddings into the embedding layer

```
get_layer(model, index = 1) %>              %
  set_weights(list(embedding_matrix))
%>%   freeze_weights()
```

Training and evaluation

```
model %>% compile (                    "rmsprop",
  optimizer =   loss =                 "binary_crossentropy",
  metrics = c( "acc"                                )
) history <  x_train, y_train,          model %>% fit(
epochs = 20,   batch_size = 32 ,
validation_data =
)
save_model_weights_hdf5(model,                        list(x_val, y_val)

plot(history)                                                    "pre_trained_glove_model.h5")
```

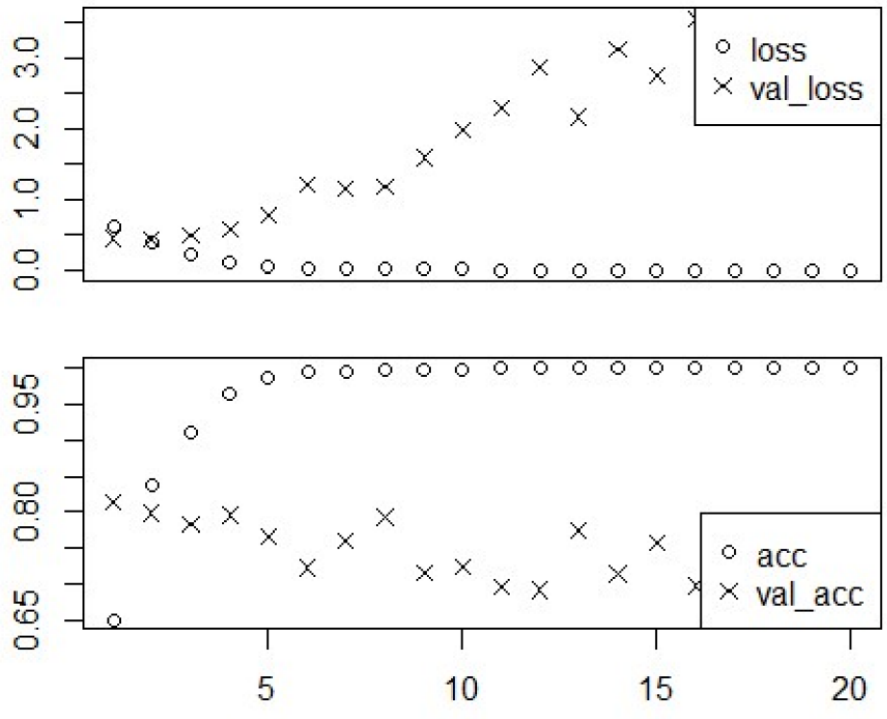


Training the same model without pretrained word embeddings

```
model2 <- keras_model_sequential() %>%          output_dim = embedding_dim,
```

```
 layer_embedding(input_dim =  max_words,
input_length =   layer_simple_rnn(units = 32,
layer_simple_rnn(units = 32,
layer_simple_rnn(units = 32,
layer_simple_rnn(units = 32) %>%
layer_dense(units = 1, activation =

model2 %>% compile(   optimizer =
"rmsprop",   loss = "binary_crossentropy",
metrics = c("acc")
)
history2 <- model2 %>% fit(
x_train, y_train,   epochs = 20,
batch_size = 32,
 validation_data =  list(x_val, y_val)
)
plot(history2)
```

```
   maxlen) %>%
return_sequences = TRUE)
%>% return_sequences =
TRUE) %>% return_sequences
= TRUE) %>%

"sigmoid")
```



Tokenizing the test set data

```
test_dir <- file.path(imdb_dir, "test")
```

```
labels <- c() texts <- c()
        for (label_type in c("neg", "pos")) {   label <-
             switch(label_type, neg = 0,   dir_name <
          file.path(test_dir, label_type)   for (fname in
    list.files(dir_name,                        full.names =
   texts < c(texts, readChar(fname,
file.info(fname)$size))
   labels <- c(labels, label)
 } }
sequences < texts_to_sequences(tokenizer, texts)
x_test <- pad_sequences(sequences, maxlen =
y_test <- as.array(labels)

results2 < model2 %>% evaluate(x_test, y_test)
results2

##     loss      acc
## 3.317801 0.751160
```

pos = 1)

pattern = glob2rx("*.txt"), TRUE)) {

maxlen)

Evaluating on the test set model

```
model        %
%>             load_model_weights_hdf5("pre_trained_glove_model.h5") %>%   evaluate(x_test,
               y_test)

               ##      loss       acc
               ## 0.6346291 0.7012800
```

Summary:

	Embedding Layer		Pre-trained Glove	
	Loss	Accuracy	Loss	Accuracy
Training samples=100 With RNN	0.8566806	0.5072000	0.7793868	0.5156000
Training samples=500 With RNN	2.596861	0.506880	1.767162	0.526520
Task 3 Training samples=10000 With RNN	3.317801	0.751160	0.6346291	0.7012800

The summary table shows that whenever using a small dataset the model did not pass the accuracy above 53% but as soon as the training sample size increase to 10000 then accuracy will reach 75%. When using the embedding layer with the RNN model the model gives an accuracy of 75.12% while loss is high which is 3.32. However, when using the pre-trained glove model, the model accuracy is 70.13% and loss is less which is 0.63 compared to the embedding layer.

Using the dataset of assignment 3, the project's results shows that it will work better with large training samples. In different training samples pre-trained model outclasses the embedding layer model, which is also expected for the large training samples, but the result shows that the embedding layer has higher accuracy which is 75% approximately and the pre-trained Glove model has an accuracy of 70% approximately.