VARANASI_AKSHAY_FINAL

Akshay Kumar Varanasi

December 15, 2018

Final Project

This project is about using Neural Network for Image Classification. Initially for this project, the idea was to try for Inclusice Image Challenge in Kaggle.(https://www.kaggle.com/c/inclusive-images-challenge) Since the size of the training datset is already big so I thought of using some part of it for training and other for testing. Another advantage of doing that is we also get to find the accuracy as we already have the labels for the full dataset.

Please download it as it is too big for me to share Link to Dataset (Open Image Val ~ 12GB) https://www.kaggle.com/victorhz/open-image-val

Labels for it ('validation-annotations-human-imagelabels.csv') https://www.kaggle.com/victorhz/openimagevallabel

Calling the Required Libraries

As I wanted to use Neural Network and since I am a beginner, I used Keras instead of Neural Network Package.

```
    ## Using Keras for Neural Networks
    library(keras)
  #install_keras()

  ## Using DBI to read from Database
  #install.packages("RPostgreSQL") # Please install this if you don't have it.
  library(DBI)
```

Reading from the database

First we need to connect to the database for that you need to use your username and password. After connecting, read the required table. Look at the VARANASI_AKSHAY_VALIDATION_LABELS.sql file for seeing how did I create the database table.

```
    # # Connecting to database
    con<-dbConnect(RPostgreSQL::PostgreSQL(),
                   dbname="me397",
                   host="db1.wrangler.tacc.utexas.edu",
                   port=5432 ,
                   user="varanasi",
                   password="Krsna@108")

    # Listing the tables
    #dbListTables(con)

    # Reading the table
df=dbReadTable(con,"varanasi_akshay_validation_labels")
head(df)
```

```
##            imageid       source labelname confidence
## 1 0001eeaf4aed83f9 verification   /m/0k5j          1
## 2 0001eeaf4aed83f9 verification /m/013c1m          0
## 3 0001eeaf4aed83f9 verification /m/013sr7          0
## 4 0001eeaf4aed83f9 verification /m/015y8h          1
## 5 0001eeaf4aed83f9 verification /m/015z8s          0
## 6 0001eeaf4aed83f9 verification /m/015z_b          0
```

```
 # This table contains csv file imported to it
```

Subsetting to get the required data

Since we are considering 20 classes only, we need image ids and labels of them only.

```
    # Reading the labels file directly
    #df=read.csv('validation-annotations-human-imagelabels.csv')

  print(nrow(df))
```

```
## [1] 551390
```

```
    df=data.frame(df)
    df=df[c(1,3,4)]
    df = df[df$confidence==1,]


    classes = c('/m/01g317', '/m/09j2d', '/m/04yx4', '/m/0dzct', '/m/07j7r', '/m/05s2s', '/m/03bt1vf',

    df=df[df$labelname %in% classes,]
    head(df)
```

```
##             imageid labelname confidence
## 11 0001eeaf4aed83f9  /m/07yv9          1
## 29 000595fe6fee6369  /m/02wbm          1
## 46 00075905539074f2 /m/01g317          1
## 49 00075905539074f2  /m/09j2d          1
## 50 00075905539074f2  /m/09j5n          1
## 56 0007cebe1b2ba653 /m/01g317          1
```

```
    labels = as.character(df$labelname)
    Imageid = as.character(df$imageid)
    print(nrow(df))
```

```
## [1] 68867
```

Training data

Reading the training data and labels along with validation data into matices

```
    x_train=c()
    x_val=c()
    # To read the filenames in the directory into a list
    #jpgFiles <- list.files("validation", pattern="*jpg$", full.name=TRUE)

    # For reading the training
```

```
for (filename in Imageid[10001:15000] ){
  filepath = paste('validation/',filename,'.jpg',sep = "")
  dl=image_load(path = filepath,grayscale = TRUE,target_size = c(100,100))
  temp=image_to_array(img = dl,data_format = NULL)
  x_train=append(x_train,temp)
}

# For reading the validation
for (filename in Imageid[1:2000] ){
  filepath = paste('validation/',filename,'.jpg',sep = "")
  dl=image_load(path = filepath,grayscale = TRUE,target_size = c(100,100))
  temp=image_to_array(img = dl,data_format = NULL)
  x_val=append(x_val,temp)
}

# Normalizing the images
x_train=x_train/255
x_val=x_val/255

# Labels
y_train = c()
y_val = c()

# For training
for(i in labels[10001:15000]){
  temp=rep(0L,20)
  temp[match(i,classes)]=1
  y_train=rbind(y_train,temp)
  rm(temp)
}
# For validation
for(i in labels[1:2000]){
  temp=rep(0L,20)
  temp[match(i,classes)]=1
  y_val=rbind(y_val,temp)
  rm(temp)
}

# Reshaping
x_train <- array_reshape(x_train, c(nrow(y_train),100,100,1))
x_val <- array_reshape(x_val, c(nrow(y_val),100,100,1))
```

Building the Neural Network

Now we need to build neural network for doing the classification. Since this is Image Classification Problem, it is better to use CNN as it has been shown that they are good for those tasks. I build the following model with some intuition and some experiments with the parameters like filters, units, layers etc.

```
# CNN
  model <- keras_model_sequential()
  model %>%
    layer_batch_normalization(input_shape = c(100L,100L,1L)) %>%
    layer_conv_2d(filters=4,kernel_size = c(2L,2L) ,strides=c(1L,1L))%>%
```

```
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_conv_2d(filters =8,kernel_size = c(2L,2L) ,strides=c(1L,1L))%>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_conv_2d(filters = 16,kernel_size = c(2L,2L) ,strides=c(2L,2L))%>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_conv_2d(filters=32,kernel_size = c(2L,2L) ,strides=c(1L,1L))%>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_conv_2d(filters=32,kernel_size = c(2L,2L) ,strides=c(2L,2L))%>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_flatten() %>%
    layer_dense(units = 2048) %>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 1024)  %>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 512) %>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 128) %>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 64) %>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 32)  %>%
    layer_activation_parametric_relu() %>%
    layer_batch_normalization() %>%
    layer_dropout(rate = 0.25) %>%
    layer_dense(units = 20) %>%
    layer_activation_softmax()
```

After building the Neural Network, I compile it with appropriate loss function, optimizer and the metric. In this case, I used categorical_crossentropy, since cross entropy is preferred loss function for images. And since this is classification problem we use categorical one. Metric which we are interested is accuracy so we use that and optimizer used was Adam. After reading about various optimizations, I felt that is the best.

```
    # Next, compile the model with appropriate loss function, optimizer, and metrics:
    model %>% compile(
```

```
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(),
  metrics = c("accuracy")
)



# look at the final model
summary(model)
```

```
## _________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## =========================================================================
## batch_normalization_1 (BatchNorm (None, 100, 100, 1)           4
## _________________________________________________________________________
## conv2d_1 (Conv2D)                (None, 99, 99, 4)             20
## _________________________________________________________________________
## p_re_lu_1 (PReLU)                (None, 99, 99, 4)             39204
## _________________________________________________________________________
## batch_normalization_2 (BatchNorm (None, 99, 99, 4)             16
## _________________________________________________________________________
## dropout_1 (Dropout)              (None, 99, 99, 4)             0
## _________________________________________________________________________
## conv2d_2 (Conv2D)                (None, 98, 98, 8)             136
## _________________________________________________________________________
## p_re_lu_2 (PReLU)                (None, 98, 98, 8)             76832
## _________________________________________________________________________
## batch_normalization_3 (BatchNorm (None, 98, 98, 8)             32
## _________________________________________________________________________
## dropout_2 (Dropout)              (None, 98, 98, 8)             0
## _________________________________________________________________________
## conv2d_3 (Conv2D)                (None, 49, 49, 16)            528
## _________________________________________________________________________
## p_re_lu_3 (PReLU)                (None, 49, 49, 16)            38416
## _________________________________________________________________________
## batch_normalization_4 (BatchNorm (None, 49, 49, 16)            64
## _________________________________________________________________________
## dropout_3 (Dropout)              (None, 49, 49, 16)            0
## _________________________________________________________________________
## conv2d_4 (Conv2D)                (None, 48, 48, 32)            2080
## _________________________________________________________________________
## p_re_lu_4 (PReLU)                (None, 48, 48, 32)            73728
## _________________________________________________________________________
## batch_normalization_5 (BatchNorm (None, 48, 48, 32)            128
## _________________________________________________________________________
## dropout_4 (Dropout)              (None, 48, 48, 32)            0
## _________________________________________________________________________
## conv2d_5 (Conv2D)                (None, 24, 24, 32)            4128
## _________________________________________________________________________
## p_re_lu_5 (PReLU)                (None, 24, 24, 32)            18432
## _________________________________________________________________________
## batch_normalization_6 (BatchNorm (None, 24, 24, 32)            128
## _________________________________________________________________________
## dropout_5 (Dropout)              (None, 24, 24, 32)            0
```

```
## _________________________________________________________________
## flatten_1 (Flatten)              (None, 18432)                 0
## _________________________________________________________________
## dense_1 (Dense)                  (None, 2048)                  37750784
## _________________________________________________________________
## p_re_lu_6 (PReLU)                (None, 2048)                  2048
## _________________________________________________________________
## batch_normalization_7 (BatchNorm (None, 2048)                  8192
## _________________________________________________________________
## dropout_6 (Dropout)              (None, 2048)                  0
## _________________________________________________________________
## dense_2 (Dense)                  (None, 1024)                  2098176
## _________________________________________________________________
## p_re_lu_7 (PReLU)                (None, 1024)                  1024
## _________________________________________________________________
## batch_normalization_8 (BatchNorm (None, 1024)                  4096
## _________________________________________________________________
## dropout_7 (Dropout)              (None, 1024)                  0
## _________________________________________________________________
## dense_3 (Dense)                  (None, 512)                   524800
## _________________________________________________________________
## p_re_lu_8 (PReLU)                (None, 512)                   512
## _________________________________________________________________
## batch_normalization_9 (BatchNorm (None, 512)                   2048
## _________________________________________________________________
## dropout_8 (Dropout)              (None, 512)                   0
## _________________________________________________________________
## dense_4 (Dense)                  (None, 128)                   65664
## _________________________________________________________________
## p_re_lu_9 (PReLU)                (None, 128)                   128
## _________________________________________________________________
## batch_normalization_10 (BatchNor (None, 128)                   512
## _________________________________________________________________
## dropout_9 (Dropout)              (None, 128)                   0
## _________________________________________________________________
## dense_5 (Dense)                  (None, 64)                    8256
## _________________________________________________________________
## p_re_lu_10 (PReLU)               (None, 64)                    64
## _________________________________________________________________
## batch_normalization_11 (BatchNor (None, 64)                    256
## _________________________________________________________________
## dropout_10 (Dropout)             (None, 64)                    0
## _________________________________________________________________
## dense_6 (Dense)                  (None, 32)                    2080
## _________________________________________________________________
## p_re_lu_11 (PReLU)               (None, 32)                    32
## _________________________________________________________________
## batch_normalization_12 (BatchNor (None, 32)                    128
## _________________________________________________________________
## dropout_11 (Dropout)             (None, 32)                    0
## _________________________________________________________________
## dense_7 (Dense)                  (None, 20)                    660
## _________________________________________________________________
## softmax_1 (Softmax)              (None, 20)                    0
```

```
## ============================================================================
## Total params: 40,723,336
## Trainable params: 40,715,534
## Non-trainable params: 7,802
## ____________________________________________________________________________
```

Fitting the model

I fit the model. I set the batch size as 300 and run 20 epochs. The More, the better but due to computational limitations could not go further.

Test Data

Reading the test data and its labels into a matrices.

```
x_test=c()
  # For reading the testing data
  for (filename in Imageid[25001:27000] ){
    filepath = paste('validation/',filename,'.jpg',sep = "")
    dl=image_load(path = filepath,grayscale = TRUE,target_size = c(100,100))
    temp=image_to_array(img = dl,data_format = NULL)
    x_test=append(x_test,temp)
  }

  # For evaluation
  y_test=c()
  for(i in labels[25001:27000]){
    temp=rep(0L,20)
    temp[match(i,classes)]=1
    y_test=rbind(y_test,temp)
    rm(temp)
  }

  x_test<-x_test/255
  x_test <- array_reshape(x_test, c(nrow(y_test),100,100,1))
```

Using the models to predict and get the probabilities

```
pre=model%>%predict_proba(x_test)
evaluated=model%>%evaluate(x_test,y_test)
```

Getting the probability of each classes in to a dataframe and selecting the one with highest probability.

```
predict_labels=data.frame()
  for(i in 1:nrow(pre)){
    temp=sort(pre[i,],index.return=TRUE)$ix
    predict_labels=rbind(predict_labels,temp)
  }
  colnames(predict_labels)=c('1st','2nd','3rd','4th','5th','6th','7th','8th','9th','10th','11th','12t

  # Taking only the first column
  pre_1=predict_labels$`1st`
```

Getting the labels for the predicted classes and calculating the accuracy.

```
  # Getting the predicted labels
  labels_test =c()

    for (it in 1:length(pre_1)){
       q=classes[pre_1[it]]
       labels_test=append(labels_test,q)
    }

  # Calculating the accuracy
    ind=which(labels_test==labels[25001:27000])
    acc=length(ind)/length(labels_test)
    acc=acc*100
    print(paste("The classification accuracy is",acc,"%"))
```

```
## [1] "The classification accuracy is 2.4 %"
```

```
    # Writing out the labels predicted into a file
    df_test=df[25001:27000,]
    df_test$LabelName=labels_test
    write.csv(df_test,file = 'Labels_test.csv',row.names = FALSE)
```

Thus we get the classification results!