CNN Model

Yu Yan

2023-05-05

Check data Dimentionality

Before we feed the train data into some model to train, we identified that we are lacking succifient amount of image data.

To account for the inefficiency of data images, we included the structure of image data generator. It is basically a form of data augmentation Technic in the context of image data. We created artificial data that is consisted of transformations of original image without acquiring new images. This would increase the size of train data set to make the model training process more robust. In the keras package, the function image_data_genertor() completes such task. We could specify the transformations that we would like to achieve. For example, we can flip the images horizontally and vertically, changing contrast and hue of the images, zooming, shearing and also changing brightness of the images. Here we are implementing the following changes:

rescale by a factor of 1/255 shift width and height by a factor of 0.2 shear and zoom the image by a factor of 0.2 make both horizontal and vertical flip make whitening and set brightness range to be 0.2

After data augmentation, we would establish objects as image train and test dataset. It is the specific type of format that keras identify as train data input. This reduces the amount of work for users to load each images from folders into the environment and the model could directly call the data from the established directory. We are creating two generators, each one for train and test data. Remember to set the input size of images beforehand so that the dimensions are restricted to make sure reasonable training time. And we also set the batch size parameter to be 32. It means that we let the model handle 32 images at one loop of training. This technique of mini-batch would reduces the memory size required for training while make training process faster as the number of parameters it needs to update(weights and bias) significantly smaller. In addition, we established the color mode to be 'RGB' which is a three color-way regulation with respect to the grey scale. This is because we identified images that are not black and white in the dataset as we explore, even though most typical medical images especially for CT-scanning are black and white. By calling the generator part within the setting, we implemented the previous constructed image generator to get augmented image datasets. The last but most important feature is the class_mode. Here we identified it as 'categorical' since we are dealing with a three-level labeling: COVID 19, Normal lungs, and Pneumonia lungs.

We also performed a step of computing class proportion in the fully construed dataset to make sure that we have matching proportion of images as the original data that we acquired.

Model Tessting

we have build this sequential model using tidy format under keras in R. As denoted above, the model is a combination of three component: input layer, hidden layers (consisted of convolution layer and pooling layer), and an output layer. All the convolution layers are using the activation function of 'relu'.

The protocol of training is to start with a simple model to test the feasibility and trouble shooting any potential issues.

The simple model has one convolution layer with filter size being 16. and a kernel size 3,3. Remember that we have made the input images to be a dimension of 64,64, so the filter size should be some proportion of 64. After this, we appended a max pooling layer to wrap up information gained from convolution layer by pool the maximum number of each filer shift. Then the convolution part is flattened and moved on to dense layer. Here we have two layers, repectively with 16 and 32 nodes.

The final layer, output layer is a dense layer with three units, since we have three labels. We set the activation function to be softmax so that the model will finally give its prediction of probabilities for each label of any given image. And the three probabilities should sum up to 1.

Now that we are finally able to compile the model with our constructed train and test dataset from the last section. We want the loss function to be 'categorical_crossentropy', and optimization algorithm to be adam with a learning rate of 0.0001. TensorFlow enables a great deal of flexibilities here that user can try out different optimization algorithm and learning rate. And we also want the model to output accuracy so that we could evaluate.

Let's fit the mode with train data and evaluate on test data! The epochs are set to be 30 which means the trainning process will go through the entire train data 30 times. To accelerate training time, we added the option of multiprocessing and an early stopping critiera by patience being 5 in terms of accuracy check, so that the model train will stop earlier if detected convergence.

We can see the summary of the model in the following graph.

Simple Model

After training, we can see that the accuracy for the simple model is stabilized around 0.8 and loss around 0.45. It's performance is a little better in the test data with accuracy at around 0.84 and loss around 0.40. We can see the process of trainning in the graph and the training is stopped at 18 epoch. After ensuring the model architecture feasibility, we would like to do some model selections and pruning provided by the flexibility provided by keras package.

Model Pruning

We would like to increase the complexitiy of the model by adding more convolution layers and pooling layers while increase the sizes of the filters to keep the area of feature extraction bigger so the model may potentially get more out of it.

The input layer is now a convolution layer with 32 filters and a 3,3 kernel size. The input_shape should be specified and matched what we set beforehand. We have 64,64,3 since we expect the input image to be at dimension of 64 by 64 and 3 means we are expecting RGB option. If we have grey scale set, we would instead insert 1 at this block. We also added an element to 'stride' so that there's one unit shifting in the input layer. Remember to append a pooling layer after each convolution layer to wrap up the feature information extracted by the convolution layer filtering.

Then we have three convolution layers with filter numbers respectively 64,128,and 128. This is simply the result of our exploration and training, users can have their own exploration over the layers and number of filters to train the model. By inserting a flatten layer, we are end with the convolution part and moved on to the typical networks to perform classification assignment.

Start with a drop out layer of 0.5, we added two dense layer with units 128 and 64. This would convert information of image features to make classification task.

The final layer, output layer is a dense layer with three units, since we have three labels. We set the activation function to be softmax so that the model will finally give its prediction of probabilities for each label of any given image. And the three probabilities should sum up to 1.

Similarly for the simple model we compile the new model with exactly same setting except we set the epoch larger to be 50. Since we are training a significantly larger amound of parameters, we would expect the training to converge later.

In the model summary page we can see that there are a total of 314,947 parameters to prune with respect to the 247,123 parameters of the simple model.

As a result of training, we can see that the model accuracy has improved. It has now achieved approximately 0.9 of accuracy and 0.25 loss. This indicate that adding more layers will increase the accurcay. However, there should be some consideration of running time and over fitting when adding structures to the model.

Model Evaludation

After the CNN model is created and its structure fit, we would like to further test out its validity and accuracy on the previously spitted validation data set from training set. It is the set that has not been seen in the process of training and it is a great option to test if the model has overfitted on the training data.

To do so, we created a validate data set using the same approach when we create train and test dataset, and simply call the evaluate() in keras package to test the model performance on validation dataset.

As the result shows, the accuracy is approximately simillar to the training process, as well as the loss. This indicates that our model has a good extent of generality, and does not overfit on the trainning data.

Code Appendix:

```
knitr::opts_chunk$set(echo = F)
knitr::opts_chunk$set(message = F)
knitr::opts_chunk$set(warning = F)
knitr::opts_chunk$set(eval  = F)
library(kableExtra)
library(stats)
library(tableone)
library(lubridate)
library(knitr)
library(tidyverse)
library(Deriv)
library(plotly)
library(kableExtra)
library(randomForest) # random forests
library(gbm) # gradient boosting
library(pROC) # AUC curve
library(caret) # Test train split and accuracy
library(rpart.plot) # Visualization of Regression Tree
library(OOBCurve)
library(tensorflow)
library(keras)
#library(imager)
library(magick)
setwd("/Users/oscar/Library/CloudStorage/GoogleDrive-yu_yan@brown.edu/Shared drives/PHP 2650 Final Proj
train_covid_list = list.files('Data/train/COVID19/')
train_normal_list = list.files('Data/train/Normal/')
train_pneu_list = list.files('Data/train/PNEUMONIA/')

test_covid_list = list.files('Data/test/COVID19/')
test_normal_list = list.files('Data/test/Normal/')
test_pneu_list = list.files('Data/test/PNEUMONIA/')


# Move some images from both folders to create a validation folder.
# While also keep the same proportion
total_train = length(train_covid_list) + length(train_normal_list) + length(train_pneu_list)
prop_train = c(length(train_covid_list)/total_train, length(train_normal_list)/total_train, length(trai

total_test= length(test_covid_list) + length(test_normal_list) + length(test_pneu_list)
prop_test = c(length(test_covid_list)/total_test, length(test_normal_list)/total_test, length(test_pneu

# originally, train,test is 80/20, so move 10% from train as validate
total_train / (total_train+total_test)

# Calculate number of images moving
num_validate = round(prop_train * round(0.1 * (total_test+total_train)))

# Move the first n files from train to validate
file.rename( paste0("Data/train/COVID19/", train_covid_list[1:num_validate[1]]),
          paste0("Data/validate/COVID19/", train_covid_list[1:num_validate[1]]) )
```

```
file.rename( paste0("Data/train/NORMAL/", train_normal_list[1:num_validate[2]]),
          paste0("Data/validate/NORMAL/", train_normal_list[1:num_validate[2]]) )

file.rename( paste0("Data/train/PNEUMONIA/", train_pneu_list[1:num_validate[3]]),
          paste0("Data/validate/PNEUMONIA/", train_pneu_list[1:num_validate[3]]) )

# Validate list
validate_covid_list = list.files('Data/validate/COVID19/')
validate_normal_list = list.files('Data/validate/Normal/')
validate_pneu_list = list.files('Data/validate/PNEUMONIA/')

total_valid = length(validate_covid_list) + length(validate_normal_list) + length(validate_pneu_list)
valid_list = c(validate_covid_list, validate_normal_list,validate_pneu_list)

train_data_gen <- image_data_generator(rescale = 1/255,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE, # Flip image horizontally
  vertical_flip = T, # Flip image vertically
  rotation_range = 45,# Rotate image from 0 to 45 degrees
  zca_whitening = T,
  brightness_range = c(0.2,0.2)
  )

target_size = c(64,64)
batch_size = 32
train_image_array_gen <- flow_images_from_directory(directory = "Data/train", # Folder of the data
                                                    target_size = target_size, # target of the image di
                                                    color_mode = "rgb", # use RGB color
                                                    batch_size = batch_size ,
                                                    seed = 123,  # set random seed
                                                    subset = "training", # declare that this is for tra
                                                    generator = train_data_gen,
                                                    class_mode='categorical'
                                                    )


test_image_array_gen <- flow_images_from_directory(directory = "Data/test", # Folder of the data
                                                    target_size = target_size, # target of the image di
                                                    color_mode = "rgb", # use RGB color
                                                    batch_size = batch_size,
                                                    seed = 123,  # set random seed
                                                    generator = train_data_gen,
                                                    class_mode='categorical'
                                                    )


# Checking to see proportion correctly identified.
table("\nFrequency" = factor(train_image_array_gen$classes)
      ) %>%
  prop.table()
```

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

  # Convolution Layer
   # Input layer
  layer_conv_2d(filters = 16, kernel_size = c(3, 3), activation = "relu", strides = c(1,1),
                input_shape = c(64, 64, 3)) %>%

  # Max Pooling Layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Flattening Layer
  layer_flatten() %>%

  # Dense Layer
  layer_dense(units = 16,
              activation = "relu") %>%
  layer_dense(units = 32,
              activation = "relu") %>%

  # Output Layer
  layer_dense(units = 3,
              activation = "softmax")

sim.model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(lr=0.0001),
  metrics = c("accuracy")
)


sim.history <- sim.model %>% fit(
  train_image_array_gen,
  epochs = 30,
  validation_data = test_image_array_gen,
  validation_steps = 20,
  use_multiprocessing=T,
  callbacks=list(
      callback_early_stopping(patience = 5, monitor = "accuracy", mode = "max")
  )
)

sim.history$metrics
plot(sim.history)

dir.create("sim.model", showWarnings = FALSE)
sim.model %>% save_model_hdf5("./model/conv1_sim_model.h5")
sim_model <- load_model_hdf5("./model/conv1_sim_model.h5")

model <- keras_model_sequential() %>%
  # Input layer
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", strides = c(1,1),
                input_shape = c(64, 64, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
```

```
  # Hidden convolution layers
  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() %>%
  # dense layers
  layer_dropout(0.5)  %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dense(units = 64, activation = "relu") %>%
  # output layer
  layer_dense(units = 3, activation = "softmax")

model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(lr=0.0001),
  metrics = c("accuracy")
)


history <- model %>% fit(
  train_image_array_gen,
  epochs = 50,
  validation_data = test_image_array_gen,
  validation_steps = 20,
  use_multiprocessing=T,
  callbacks=list(
      callback_early_stopping(patience = 5, monitor = "accuracy", mode = "max")
  )
)


plot(history)

history$metrics


dir.create("model", showWarnings = FALSE)
model %>% save_model_hdf5("./model/conv1_model.h5")
model <- load_model_hdf5("./model/conv1_model.h5")
#load_model %>% summary()
# TODO: Write functions for reading validation image and pass it to the model for evalutation
# Confusion matrix
# Remember to rescale

size = 64
channels = 3
x_valid = array(NA, dim=c(total_valid,size, size, channels))
y_valid = rep(NA,length(total_valid))

for (i in 1:total_valid){
```

```
  v_list <- list(validate_covid_list, validate_normal_list, validate_pneu_list)
  folder_list <- list('COVID19','NORMAL','PNEUMONIA')
    for(j in 1:length(v_list)) {

        if(valid_list[i] %in% v_list[[j]]) {
            img_path <- paste0('Data/validate/', folder_list[[j]], "/", valid_list[i])
            break

        }
    }

  img <- image_load(path = img_path, target_size = c(size,size))
  img_arr <- image_to_array(img)
  img_arr <- array_reshape(img_arr, c(1, size, size, channels))
  img_arr <- img_arr/255
  x_valid[i,,,] <- img_arr
  y_valid[i] <- folder_list[[j]]

}


# valid generator

# Validation data generation
valid_image_array_gen <- flow_images_from_directory(directory = "Data/validate", # Folder of the data
                                                    target_size = target_size, # target of the image di
                                                    color_mode = "rgb", # use RGB color
                                                    batch_size = batch_size,
                                                    seed = 123,  # set random seed
                                                    generator = train_data_gen,
                                                    class_mode='categorical'
                                                    )

# Model Evaluation
eval = model %>% evaluate(train_image_array_gen, steps = 32, seed=2650)



kable(eval,
caption = "Evalutaion Result of the model on Validation data", col.names = NULL,
booktabs=T, escape=F, align = "c")%>%
kable_styling(full_width = FALSE, latex_options = c('hold_position'))
```