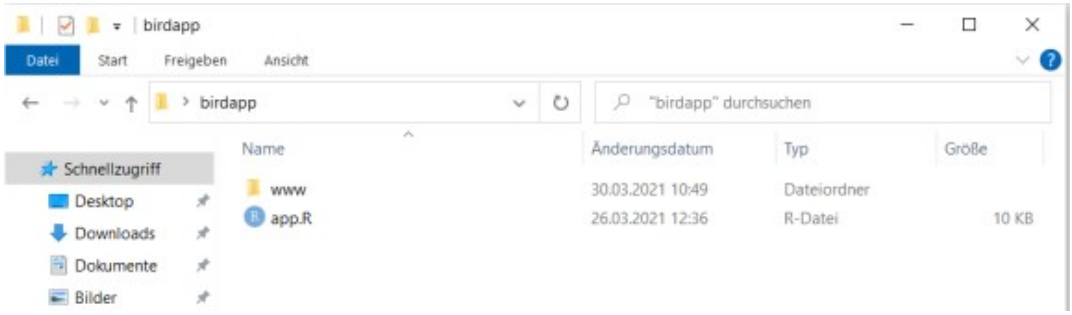
Getting started

Create a folder on your computer and name it the way you want your app to be called. If later on you are deploying your app on RStudio's shinyapps.io server, this is going to be in the URL by default. For instance, if you name the folder "birdapp", then your URL on shinyapps will read http://www.youraccount.shinyapps.io/birdapp (with "youraccount" being your account's name, of course. But we will get to that later).

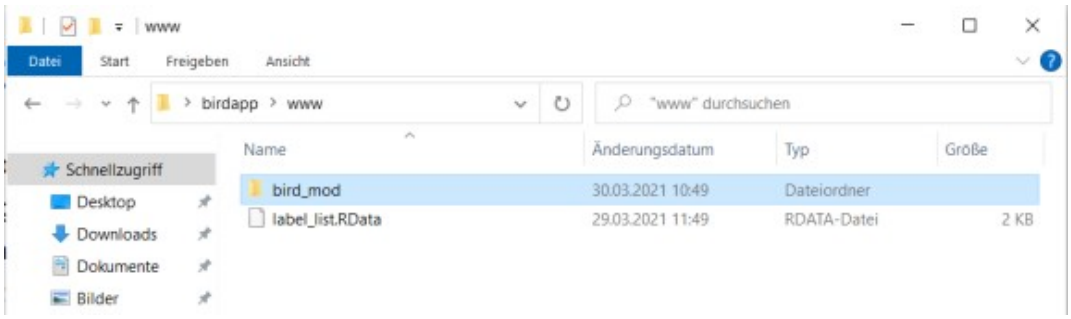
Inside this folder, you have to create a folder "www" and an app.R file (In RStudio, create a new R script and save as app.R in the respective folder), so your folder looks like this:



The app.R file is the R-script we are going to write in a minute. But first, move the model you have trained in the previous part of this blog post into the "www" folder. Remember that at the end of the previous post we saved our model with the command

```
model %>% save_model_tf("bird_mod")
```

which creates a folder with the Tensorflow model in your working directory. You move this into the newly created "www" directory so it will later be uploaded to the shinyapps server. You also need to move the file "label_list.RData" which we exported in the beginning of the first part of this blog post into your www folder. So your www folder should look like this:



Now let's open the "app.R" script and create the app!

Inside app.R

Once you opened RStudio, set your working directory by entering the following command into the console – but don't save it in your script! This would later cause an error on the shinyapps server because there your local folder is obviously not available.

```
setwd("C:/Users/my_name/Desktop/birdapp")
```

Of course, you edit the path to where your folder is located. If you are on mac, the path typically starts only with an "/".

Inside your app.R, you start by loading the required libraries.

```
library(shiny)
```

```
library(shinydashboard)
library(rsconnect)
library(keras)
library(tensorflow)
library(tidyverse)
```

We also load the model as well as the list with birds names from our www directory and define the image size (must be the same as the input size for the keras model you trained):

```
model <- load_model_tf("www/bird_mod")
load("www/label_list.RData")
target_size <- c(224,224,3)
options(scipen=999) #prevent scientific number formatting
```

Now we define the “ui” object, i.e. the user interface. We are keeping it simple here. In the dashboardHeader() element, we set the name of the app to be displayed in the upper right corner and a dropdown menu where we put a link to this blog. Next, we want to have a sidebar with the upload button for the images. Finally, we define the body of the page by inserting a bit of text and the placeholders for the image and text outputs.

```
ui <- dashboardPage(
  skin="green",

  #(1) Header

  dashboardHeader(title=tags$h1("Bird-App",style="font-size: 120%; font-
weight: bold; color: white"),
                  titleWidth = 350,
                  tags$li(class = "dropdown"),
                  dropdownMenu(type = "notifications", icon = icon("question-
circle", "fa-1x"), badgeStatus = NULL,
                               headerText="",
                               tags$li(a(href = "https://forloopsandpiepkicks.
wordpress.com",
                                   target = "_blank",
                                   tagAppendAttributes(icon("icon-circle"),
class = "info"),
                                   "Created by"))
                             )),


  #(2) Sidebar

  dashboardSidebar(
    width=350,
    fileInput("input_image","File" ,accept = c('.jpg','.jpeg')),
    tags$br(),
    tags$p("Upload the image here.")
  ),


  #(3) Body

  dashboardBody(

    h4("Instruction:"),
    tags$br(),tags$p("1. Take a picture of a bird."),
```

```
    tags$p("2. Crop image so that bird fills out most of the image."),
    tags$p("3. Upload image with menu on the left."),
    tags$br(),

    fluidRow(
      column(h4("Image:"),imageOutput("output_image"), width=6),
      column(h4("Result:"),tags$br(),textOutput("warntext",), tags$br(),
             tags$p("This bird is probably a:"),tableOutput("text"),width=6)
    ),tags$br()

))
```

Of course, this is just an exemplary setup. You can play around with the different elements and styles to find your desired style. We use the predefined “green” skin here, look up the shiny documentation to find out which other skins you can use. In the dropdown menu where I added an icon(“question-circle”), you can of course add other stuff; look up the “font awesome” names, most of these symbols can be used in shinydashboard. The other things are probably self-explanatory, e.g. tags$br() forces a line break, a column() with width= 6 inside the fluid row takes up half of the entire (width = 12) space in the dashboard body; so you could create a layout with e.g. three columns setting width = 4 for all three…

Next, we create the server object. This is where the interactive elements are defined.

```
server <- function(input, output) {

  image <- reactive({image_load(input$input_image$datapath, target_size =
target_size[1:2])})


  prediction <- reactive({
    if(is.null(input$input_image)){return(NULL)}
    x <- image_to_array(image())
    x <- array_reshape(x, c(1, dim(x)))
    x <- x/255
    pred <- model %>% predict(x)
    pred <- data.frame("Bird" = label_list, "Prediction" = t(pred))
    pred <- pred[order(pred$Prediction, decreasing=T),][1:5,]
    pred$Prediction <- sprintf("%.2f %%", 100*pred$Prediction)
    pred
  })

  output$text <- renderTable({
    prediction()
    })

  output$warntext <- renderText({
    req(input$input_image)

    if(as.numeric(substr(prediction()[1,2],1,4)) >= 30){return(NULL)}
    warntext <- "Warning: I am not sure about this bird!"
    warntext
  })


  output$output_image <- renderImage({
    req(input$input_image)

    outfile <- input$input_image$datapath
```

```
    contentType <- input$input_image$type
    list(src = outfile,
         contentType=contentType,
         width = 400)
  }, deleteFile = TRUE)

}
```

Inside the server function we first load the image with the keras-function image_load(). We then let our model which we trained in the last blog post predict which bird species is on this image. Important: In our last blog post, we named the model “model” and saved it into the bird_mod folder. This is why we write *model %>% predict(x)* here. If you named your model differently, then of course you need to adapt this here accordingly.

After the prediction we create a data.frame with the top 5 predicted classes and let it render as a table in the next call to renderTable().

The output$warntext is an optional output that gets displayed if the highest predicted probability is below 30%. Change at will, of course. Finally, we render the uploaded image and, with the argument deleteFile = TRUE, we delete it immediately afterwards to ensure we are not encountering memory issues after uploading a couple of images.

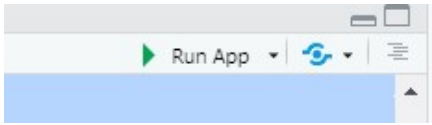
Again, this is an example and there are a million other ways to do it. If you need more inspirations, for instance, Ger Inberg has a nice blog post on a shiny app using keras as well.

Only one line left to enter into the script:

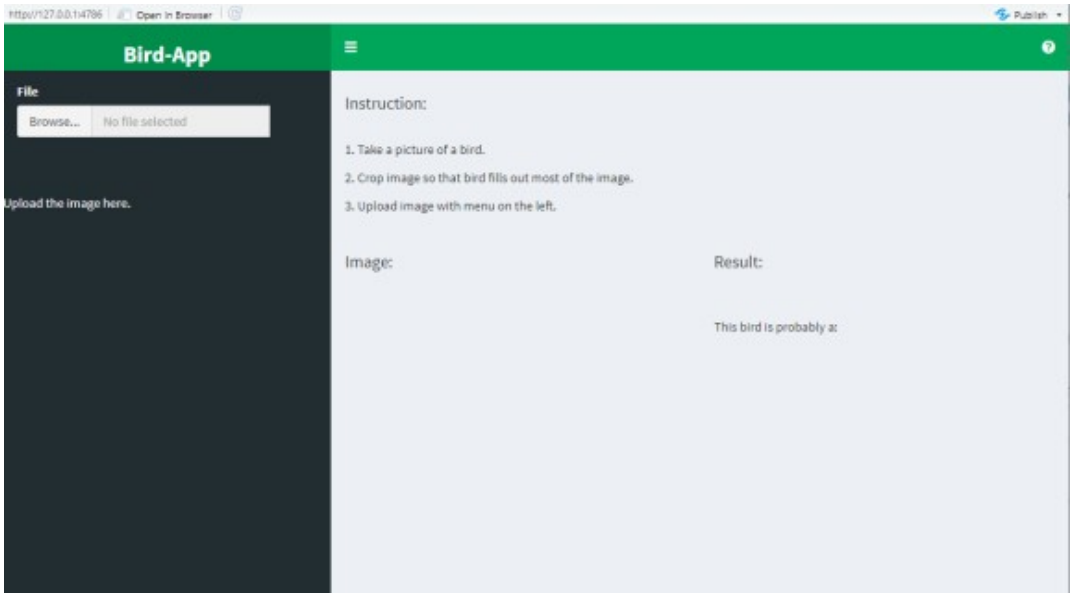
```
shinyApp(ui, server)
```

Test the app locally

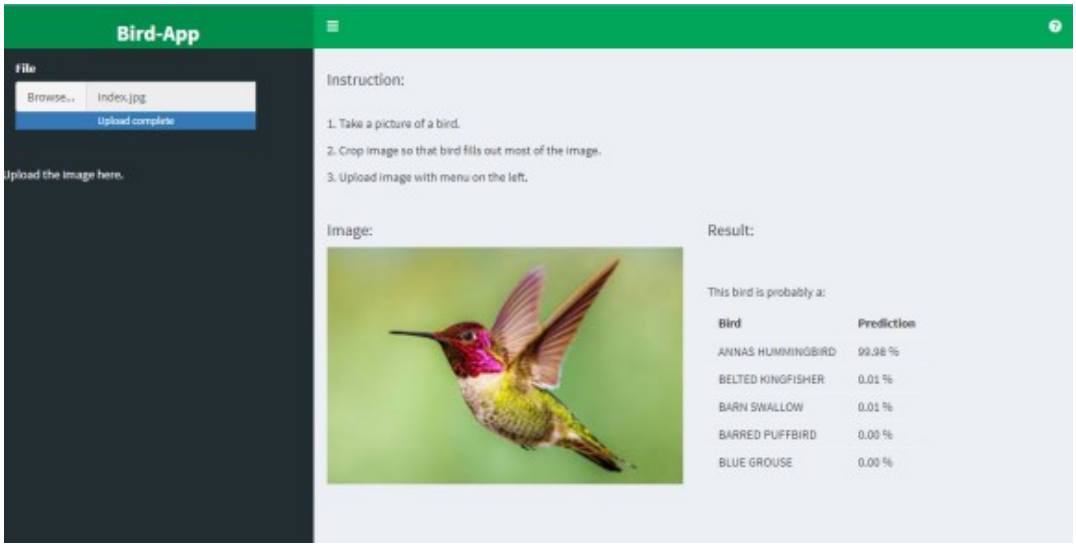
In RStudio, click on the upper right button “Run App” to test whether it is working:



The app should open in a new window:



Let’s try it out! Upload an image of one of the birds we trained the model to identify. Obviously it should not be included in the training data. I uploaded an image of a hummingbird, to start off easy:

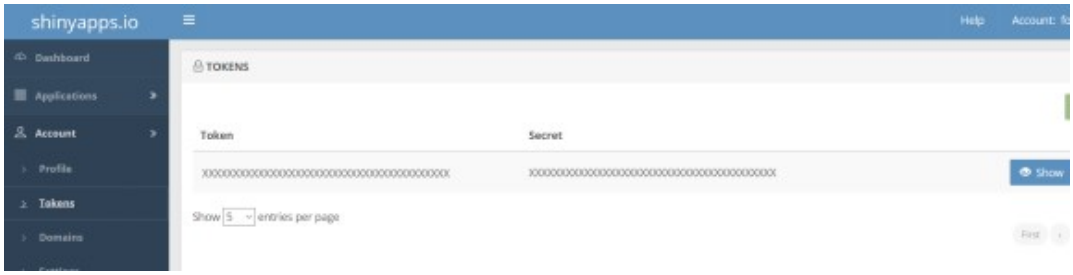


Deploy the app to a server

What fun would it be to only be able to run your app from your local RStudio?

There are multiple ways to deploy your app to a server. The easiest one is to use RStudio's service shinyapps.io. You can create a free account that lets you upload up to 5 apps. Upgrade to a paid plan in order to get more apps, more memory, custom domain names, and so on. I have a paid shinyapps.io-subscription (supporting RStudio that way), but there are other ways to deploy a shiny app into production that might be more appropriate for your case at hand. Depending on your knowledge of cloud computing, these other options are more or less feasible. Charles Bodet has a great guide on how to deploy shiny apps on AWS. We will stick to the most simple option here and use shinyapps.io.

Head over to shinyapps.io and create an account. Once you have logged in, click on "Account" and "Tokens" and then on the "show" button on the right to reveal your token and secret.



Go back to RStudio and paste the following into the console, where you change your account name, token, and secret obviously:

```
rsconnect::setAccountInfo(name='my_account',
            token='XXXXXXXXXXXXXXXXXXXXX',
            secret='XXXXXXXXXXXXXXXX')
```

Having set your credentials, you can now transfer your app to the shinyapps.io server by running the following command from the console:

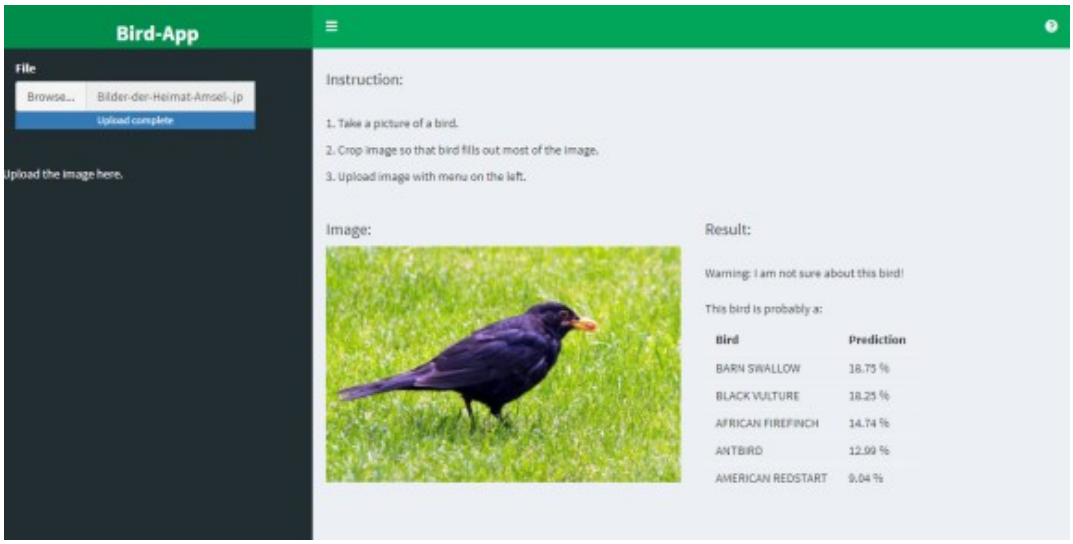
```
rsconnect::deployApp()
```

And that's it! After a few minutes processing time, you should automatically be redirected to your app in your browser.

Link to demo app

Click here!

But keep in mind that we only trained the model to recognize 40 bird species (from African Crowned Kane to Blue Heron). So if you don't have any Bananaquits or Baltimore Orioles in your backyard and instead upload an image of a common Blackbird, be prepared for strange mis-classifications…:



Thank you for your interest,…