````
---
title: "ds4002_project2"
author: "Mianchen Cao"
date: '2023-03-21'
output: html_document
---

# Installing Programs
```{r}
install.packages("tensorflow")
library(tensorflow)
install_tensorflow(extra_packages="pillow")
```

# Loading in Libraries
```{r}
library(tidyverse)
install.packages("keras")
library(keras)
library(tensorflow)
library(reticulate)
```

# Loading in Training Data
```{r}
setwd("/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/Selected")
label_list <- dir("/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/Selected/
train")
output_n <- length(label_list)
save(label_list, file="label_list.R")
```

# Setting basic parameters
```{r}
#Rescaling all images in pixels
width <- 224
height<- 224
target_size <- c(width, height)
rgb <- 3 #color channels
```

# Set Path
```{r}
path_train <- "/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/Selected/
train"
train_data_gen <- image_data_generator(rescale = 1/255, validation_split = .
2)
```

# Loading data
```{r}
train_images <- flow_images_from_directory(path_train,
  train_data_gen,
  subset = 'training',
````
````

```
  target_size = target_size,
  class_mode = "categorical",
  shuffle=F,
  classes = label_list,
  seed = 2023)
```

```{r}
validation_images <- flow_images_from_directory(path_train,
 train_data_gen,
  subset = 'validation',
  target_size = target_size,
  class_mode = "categorical",
  classes = label_list,
  seed = 2023)
```

```{r}
table(train_images$classes)
```

See an example image
```{r}
plot(as.raster(train_images[[1]][[1]][20,,,]))
```

Building Model
```{r}
mod_base <- application_xception(weights = 'imagenet',
   include_top = FALSE, input_shape = c(width, height, 3))
freeze_weights(mod_base)
```

```{r}
model_function <- function(learning_rate = 0.001,
  dropoutrate=0.2, n_dense=1024){

  k_clear_session()

  model <- keras_model_sequential() %>%
    mod_base %>%
    layer_global_average_pooling_2d() %>%
    layer_dense(units = n_dense) %>%
    layer_activation("relu") %>%
    layer_dropout(dropoutrate) %>%
    layer_dense(units=output_n, activation="softmax")

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

  return(model)
```

```
}
```

```{r}
model <- model_function()
model
```

See results of different epochs
```{r}
batch_size <- 32
epochs <- 6
hist <- model %>% fit_generator(
  train_images,
  steps_per_epoch = train_images$n %/% batch_size,
  epochs = epochs,
  validation_data = validation_images,
  validation_steps = validation_images$n %/% batch_size,
  verbose = 2
)
```

Evaluating modal
```{r}
path_test <- "/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/Selected/test"
test_data_gen <- image_data_generator(rescale = 1/255)

test_images <- flow_images_from_directory(path_test,
   test_data_gen,
   target_size = target_size,
   class_mode = "categorical",
   classes = label_list,
   shuffle = F,
   seed = 2023)
model %>% evaluate_generator(test_images,
                     steps = test_images$n)
```

Application
```{r}
test_image <- image_load("/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/
Selected/test/RED SPOTTED PURPLE/5.jpg",
                                  target_size = target_size)
x <- image_to_array(test_image)
x <- array_reshape(x, c(1, dim(x)))
x <- x/255
pred <- model %>% predict(x)
pred <- data.frame("Butterfly" = label_list, "Probability" = t(pred))
pred <- pred[order(pred$Probability, decreasing=T),][1:5,]
pred$Probability <- paste(format(100*pred$Probability,2),"%")
pred
```

See the general performance
```{r}
# define test data generator
test_datagen <- image_data_generator(rescale = 1/255)

# create a generator for the test dataset
test_generator <- flow_images_from_directory(
  directory = '/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/Selected/
test',
  target_size = c(224, 224),
  batch_size = 32,
  class_mode = 'categorical'
)

predictions <- model %>%
  predict_generator(
    generator = test_generator,
    steps = test_generator$n
  ) %>% as.data.frame
names(predictions) <- paste0("Class",0:24)
predictions$predicted_class <-
  paste0("Class",apply(predictions,1,which.max)-1)
predictions$true_class <- paste0("Class",test_generator$classes)
predictions %>% group_by(true_class) %>%
  summarise(percentage_true = 100*sum(predicted_class ==
    true_class)/n()) %>%
    left_join(data.frame(butterfly= names(test_generator$class_indices),
    true_class=paste0("Class",0:24)),by="true_class") %>%
  select(butterfly, percentage_true) %>%
  mutate(butterfly = fct_reorder(butterfly,percentage_true)) %>%
  ggplot(aes(x=butterfly,y=percentage_true,fill=percentage_true,
    label=percentage_true)) +
  geom_col() + theme_minimal() + coord_flip() +
  geom_text(nudge_y = 3) +
  ggtitle("Percentage correct classifications by butterfly species")
```

Tuning results and modifying model
```{r}
# Tuning Results
tune_grid <- data.frame("learning_rate" = c(0.001,0.0001),
                        "dropoutrate" = c(0.3,0.2),
                        "n_dense" = c(1024,256))
tuning_results <- NULL
set.seed(2023)
for (i in 1:length(tune_grid$learning_rate)){
  for (j in 1:length(tune_grid$dropoutrate)){
      for (k in 1:length(tune_grid$n_dense)){

        model <- model_function(
          learning_rate = tune_grid$learning_rate[i],
          dropoutrate = tune_grid$dropoutrate[j],
          n_dense = tune_grid$n_dense[k])
```

```
        hist <- model %>% fit_generator(
          train_images,
          steps_per_epoch = train_images$n %/% batch_size,
          epochs = epochs,
          validation_data = validation_images,
          validation_steps = validation_images$n %/%
            batch_size,
          verbose = 2
        )

        #Save model configurations
        tuning_results <- rbind(
          tuning_results,
          c("learning_rate" = tune_grid$learning_rate[i],
            "dropoutrate" = tune_grid$dropoutrate[j],
            "n_dense" = tune_grid$n_dense[k],
            "val_accuracy" = hist$metrics$val_accuracy))

    }
  }
}
tuning_results

best_results <- tuning_results[which(
  tuning_results[,ncol(tuning_results)] ==
  max(tuning_results[,ncol(tuning_results)])),]

best_results

mod_base <- application_xception(weights = 'imagenet',
   include_top = FALSE, input_shape = c(width, height, 3))
freeze_weights(mod_base)

model_function <- function(learning_rate = best_results["learning_rate"],
  dropoutrate = best_results["dropoutrate"],
  n_dense = best_results["n_dense"]){

  k_clear_session()

  model <- keras_model_sequential() %>%
    mod_base %>%
    layer_global_average_pooling_2d() %>%
    layer_dense(units = n_dense) %>%
    layer_activation("relu") %>%
    layer_dropout(dropoutrate) %>%
    layer_dense(units=output_n, activation="softmax")

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

  return(model)
```

```
}

model2 <- model_function()

path_valid <- "/Users/CMC/Desktop/Spring 2023/DS 4002/P2/images/Selected/
valid"
valid_data_gen <- image_data_generator(rescale = 1/255)

valid_images <- flow_images_from_directory(path_valid,
   valid_data_gen,
   target_size = target_size,
   class_mode = "categorical",
   classes = label_list,
   shuffle = F,
   seed = 2023)

model2 %>% evaluate_generator(valid_images,
                     steps = valid_images$n)

# Save the final model
model2 %>% save_model_tf("butterfly_mod")
```