project

Kapraju_naveen_kumar

Make a stylized picture. Using AI/Neural Network.

```
if (!require(keras)) install.packages('keras')
```

```
Loading required package: keras
```

```
library(keras)
library(tensorflow)
```

```
library(imager)
```

```
Loading required package: magrittr
```

```

Attaching package: 'imager'
```

```
The following object is masked from 'package:magrittr':

    add
```

```
The following objects are masked from 'package:stats':

    convolve, spectrum
```

```
The following object is masked from 'package:graphics':

    frame
```

```
The following object is masked from 'package:base':

    save.image
```

```
base_image_path <- get_file(

"1200px-Sunflower_from_Silesia2", origin = "https://upload.wikimedia.org/wikipedia/commons/t

style_reference_image_path <- get_file(

"starry_night.jpg",

origin = "https://img-datasets.s3.amazonaws.com/starry_night.jpg")

c(original_height, original_width) %<-% {

base_image_path %>%

tf$io$read_file() %>%

tf$io$decode_image() %>%

dim() %>% .[1:2]

}

img_height <- 400

img_width <- round(img_height * (original_width / original_height))
```

```
preprocess_image <- function(image_path) {
  image_path %>%
    tf$io$read_file() %>%
    tf$io$decode_image() %>%
    tf$image$resize(as.integer(c(img_height, img_width))) %>%
    k_expand_dims(axis = 1) %>%
    imagenet_preprocess_input()
}

deprocess_image <- tf_function(function(img) {
  if (length(dim(img)) == 4)
    img <- k_squeeze(img, axis = 1)
```

```
  c(b, g, r) %<-% {
    img %>%
      k_reshape(c(img_height, img_width, 3)) %>%
      k_unstack(axis = 3)
  }

  r %<>% `+`(123.68)
  g %<>% `+`(103.939)
  b %<>% `+`(116.779)

  k_stack(c(r, g, b), axis = 3) %>%
    k_clip(0, 255) %>%
    k_cast("uint8")
})
```

```
model <- application_vgg19(weights = "imagenet", include_top = FALSE)

outputs <- list()
for (layer in model$layers)
  outputs[[layer$name]] <- layer$output

feature_extractor <- keras_model(inputs = model$inputs,
                                 outputs = outputs)
```

```
content_loss <- function(base_img, combination_img)
    sum((combination_img - base_img)^2)
```

```
gram_matrix <- function(x) {
  n_features <- tf$shape(x)[3]
  x %>%
    tf$reshape(c(-1L, n_features)) %>%
    tf$matmul(., ., transpose_a = TRUE)
}

style_loss <- function(style_img, combination_img) {
  S <- gram_matrix(style_img)
  C <- gram_matrix(combination_img)
  channels <- 3
  size <- img_height * img_width
  sum((S - C) ^ 2) /
    (4 * (channels ^ 2) * (size ^ 2))
```

```
}
```

```
total_variation_loss <- function(x) {
  a <- k_square(x[, NA:(img_height-1), NA:(img_width-1), ] -
                x[, 2:NA             , NA:(img_width-1), ])
  b <- k_square(x[, NA:(img_height-1), NA:(img_width-1), ] -
                x[, NA:(img_height-1), 2:NA            , ])
  sum((a + b) ^ 1.25)
}
```

```
style_layer_names <- c(
    "block1_conv1",
    "block2_conv1",
    "block3_conv1",
    "block4_conv1",
    "block5_conv1"
)
content_layer_name <- "block5_conv2"
total_variation_weight <- 1e-6
content_weight <- 2.5e-8
style_weight <- 1e-6

compute_loss <-
  function(combination_image, base_image, style_reference_image) {

    input_tensor <-
      list(base_image,
           style_reference_image,
           combination_image) %>%
      k_concatenate(axis = 1)

    features <- feature_extractor(input_tensor)
    layer_features <- features[[content_layer_name]]
    base_image_features <- layer_features[1, , , ]
    combination_features <- layer_features[3, , , ]

    loss <- 0
    loss %<>% `+`(
      content_loss(base_image_features, combination_features) *
        content_weight
    )
```

```
    for (layer_name in style_layer_names) {
      layer_features <- features[[layer_name]]
      style_reference_features <- layer_features[2, , , ]
      combination_features <- layer_features[3, , , ]

      loss %<>% `+`(
        style_loss(style_reference_features, combination_features) *
          style_weight / length(style_layer_names)
      )
    }

    loss %<>% `+`(
      total_variation_loss(combination_image) *
        total_variation_weight
    )

    loss
  }
```

```
compute_loss_and_grads <- tf_function(
  function(combination_image, base_image, style_reference_image) {
    with(tf$GradientTape() %as% tape, {
      loss <- compute_loss(combination_image,
                           base_image,
                           style_reference_image)
    })
    grads <- tape$gradient(loss, combination_image)
    list(loss, grads)
  })

optimizer <- optimizer_sgd(
  learning_rate_schedule_exponential_decay(
    initial_learning_rate = 100, decay_steps = 100, decay_rate = 0.96))



optimizer <-
  optimizer_sgd(learning_rate = learning_rate_schedule_exponential_decay(
    initial_learning_rate = 100,
    decay_steps = 100,
    decay_rate = 0.96
  ))
```

```
base_image <- preprocess_image(base_image_path)
style_reference_image <- preprocess_image(style_reference_image_path)
combination_image <- tf$Variable(preprocess_image(base_image_path))
```

```
library(tensorflow)
library(imager)


library(tensorflow)
library(imager)
library(fs)

output_dir <- path("style-transfer-generated-images")
iterations <- 200
dir.create(output_dir, recursive = TRUE, showWarnings = FALSE)

for (i in seq(iterations)) {
  c(loss, grads) %<-% compute_loss_and_grads(
    combination_image, base_image, style_reference_image)

  optimizer$apply_gradients(list(
    tuple(grads, combination_image)))

  if ((i %% 100) == 0) {
    cat(sprintf("Iteration %i: loss = %.2f\n", i, loss))
    img <- deprocess_image(combination_image)
    img_array <- as.array(img)  # Convert TensorFlow tensor to R array
    img_cimg <- as.cimg(img_array)  # Convert array to cimg object

    fname <- sprintf("combination_image_at_iteration_%04i.png", i)
    save.image(img_cimg, file.path(output_dir, fname))  # Save the image
  }
}
```

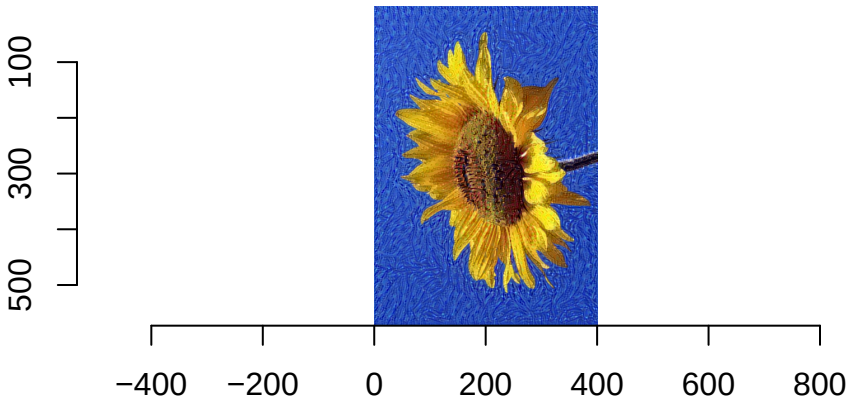
```
Iteration 100: loss = 6390.59

Warning in as.cimg.array(img_array): Assuming third dimension corresponds to
colour

Iteration 200: loss = 5075.24
```

```
Warning in as.cimg.array(img_array): Assuming third dimension corresponds to
colour
```

```
plot(img_cimg)  # Display the image using imager
```



- **Extra Credit:** Research the algorithm used. Who created it? How is the neural network implemented in Keras?
- **Solution:**Neural Style Transfer (NST) combines the content of one image with the artistic style of another. It's done using a pre-trained neural network like VGG19 in Keras. The network extracts features from both images, which are then used to generate a new image that merges their content and style.
- **Extra Credit:** Explain what the following text-to-image Generative AI products do.

  Solution: Microsoft Image Creator is a tool that turns words into pictures. You write a description, and it makes a matching image. It's like magic drawing from your words!