Lista 4: Classificação de imagens.

César A. Galvão - 190011572 João Vitor Vasconcelos - 170126064

Índice

Boa parte dos resultados dessa lista foram obtidos utilizando o apoio do ChatGPT. Na ausência, ou na dificuldade em achar, de fontes com instruções para exercícios similares, o exercício se tornou um aprendizado orientado por prompts do último modelo disponível. Buscou-se documentar todos os passos da resolução com o intuito de demonstrar que não foi feito apenas um processo de copiar e colar os resultados da plataforma, mas uma tentativa de conciliar seus resultados com as aulas, materiais disponíveis e, quando possível, a documentação do TensorFlow e Keras. O atendimento de monitoria também foi essencial para nos ajudar a explorar mais argumentos das funções e possíveis fontes de problemas na classificação do conjunto de teste.

Repositório do GitHub: https://github.com/cesar-galvao/Topicos-1---Redes-neurais

Caderno no Colab: https://colab.research.google.com/drive/13DnHbWgiF4bZfCUiU9F2Ejv2yDWCqxUB?usp=sharing

1 Características dos dados

A lista foi resolvida usando duas plataformas. A parte que diz respeito à estruturação e compreensão dos dados foi feita no RStudio, enquanto o treinamento da rede neural foi feito no Google Colab.

Antes de iniciar a construção da rede neural, foi necessário entender como os dados precisariam estar estruturados para serem utilizados. Questões como a disposição dos arquivos, informações sobre os *labels* das imagens, formato e dimensão das imagens, entre outras, foram levantadas. Isso é importante para que se possa construir um modelo que seja capaz de lidar com os dados de maneira eficiente.

A seguir, os arquivos são transformados. A justificativa para isso é descrita a seguir.

```
pacman::p_load(dplyr, magick)
```

```
# diretorios das imagens
dir_imagens <- c("../lista 4/Teste/")

# lista de arquivos .bmp
bmp <- list.files(dir_imagens, pattern = "\\.BMP", full.names = TRUE)

# transforma as imagens em png
for (file in bmp) {
  img <- image_read(file)

  # cria os paths para escrita
  output_file <- file.path(dir_imagens,
↪  paste0(tools::file_path_sans_ext(basename(file)), ".png"))
  # Escreve imagem em png
  image_write(img, path = output_file, format = "png")
}

# exclui arquivos bmp
file.remove(bmp, showWarnings = FALSE)
```

```
img <- image_read("../lista 4/Treino/F_0015.png")

image_info(img)
```

format	width	height	colorspace	matte	filesize	density
PNG	96	103	Gray	TRUE	6268	38x38

```
as.integer(image_data(img)) %>% dim()
```

```
[1] 103  96   1
```

Observa-se que:

- Há 740 imagens de digitais femininas e 3260 imagens de digitais masculinas para treino;
- Há 2000 imagens para teste;
- As imagens estão em formato `bmp`, que é um formato de imagem nativo do Windows. Optou-se por transformá-las em `png`, que é um formato mais comum e que é suportado por mais plataformas;
- As imagens em `png` possuem dimensão 96x103, com apenas um canal em escala de cinza.

Quanto à atribuição dos labels de treino, o guia TensorFlow for R foi consultado, porém não houve instruções claras. Para esclarecimento, foi consultado o ChatGPT, que indicou a utilização de uma arvore de arquivos para a organização dos dados. A seguinte estrutura foi sugerida, mas utilizada com adaptações, visto que a função utilizada inclui uma separação nativa dos dados para validação:

```
/path_to_dataset/
    /train/
        /class1/
            img1.jpg
            img2.jpg
            ...
        /class2/
            img1.jpg
            img2.jpg
            ...
    /validation/
        /class1/
            img1.jpg
            img2.jpg
            ...
        /class2/
            img1.jpg
            img2.jpg
            ...
```

Dessa forma, os arquivos foram organizados da seguinte forma:

- Treino
  - male
  - female
- Teste

2 Ajuste do ambiente

O ambiente do Colab foi ajustado com os seguintes pacotes e Runtime T4 GPU, visando acelerar o treinamento da rede.

```
install.packages("keras")
library(keras)
install.packages("tensorflow")
library(tensorflow)
install.packages("reticulate")
library(reticulate)

# GPU
```

```
tf$config$list_physical_devices("GPU")
```

3 Pré-processamento

De acordo com o a documentação do Keras, o pré-processamento via `image_data_generator()` já não é mais adequado, sendo sugerida uma função mais nova. No entanto, não identificamos um fluxo de trabalho que utilize, conforme a documentação, a função `image_dataset_from_directory()` e as demais camadas de pré-processamento[1].

Dessa forma, optamos por utilizar a função antiga. Além disso, pedimos uma sugestão ao chatGPT de hiperparâmetros para o pré-processamento e *data augmentation*, de forma que obtivemos o seguinte resultado:

```
# Data augmentation sugerida
datagen <- image_data_generator(
  rescale = 1/255,                # Rescale pixel values
  validation_split = 0.3,         # Split the data into training and validation
  rotation_range = 20,            # Randomly rotate images by up to 20 degrees
  width_shift_range = 0.2,        # Randomly shift images horizontally by 20% of the
  ↪  width
  height_shift_range = 0.2,       # Randomly shift images vertically by 20% of the
  ↪  height
  shear_range = 0.2,              # Randomly shear images
  zoom_range = 0.2,               # Randomly zoom in and out
  horizontal_flip = TRUE          # Randomly flip images horizontally
)
```

Considerando que estamos tratando de imagens de digitais, algumas dessas sugestões fazem sentido, enquanto outras não. Os argumentos são discutidos individualmente a seguir:

- Reescalar os valores dos pixels para o intervalo [0, 1] faz sentido. Imaginamos que isso possa reduzir um problema de escala;
- Utilizaremos 30% dos dados de treinamento para validação;
- Rotacionar as imagens em até 20 graus faz sentido, visto que as digitais podem estar em diferentes orientações no momento de captura da imagem;
- A rotação também aparenta fazer sentido, ja que pode existir variação no momento de captura;
- Alterações verticais e horizontais (artificialmente neste caso, para *data augmentation*) também fazem sentido como forma de generalização de formatos de dedos;
- A operação de sheer (distorção da imagem apenas no sentido horizontal) não parece fazer sentido. Até poderia fazer sentido caso houvesse uma distorção natural ou no processo de digitalização das imagens, mas decidimos não incluir;
- O zoom também faz sentido, já que podemos estar tratando de generalização de pedaços das imagens;

[1] "*image_data_generator is not recommended for new code. Prefer loading images with image_dataset_from_directory and transforming the output TF Dataset with preprocessing layers. For more information, see the tutorials for loading images and augmenting images, as well as the preprocessing layer guide*".

- Finalmente, a inversão horizontal também faz sentido, já que isso possivelmente estaríamos tratando de imagens de mãos direitas e esquerdas.

Outros argumentos disponíveis não parecem fazer sentido de serem alterados. Por exemplo, não faria sentido suavizar a imagem com normalização *feature* ou *samplewise*, já que estamos tratando de imagens com contornos finos e isso possivelmente contribui para a classificação correta da imagem. Por outro lado, `fill_mode` indica formas de preenchimento de pixels na fronteira. Para evitar que o comportamento na convolução seja dominado por `cval = 0` por padrão, selecionamos `fill_mode = "nearest"`, que expande os valores dos pixels mais próximos.

Além disso, depois de algumas tentativas de calibragem do modelo e classificação das imagens de teste com esses hiperparâmetros notamos que algumas configurações geravam resultados melhores com a mesma arquitetura de rede. O principal resultado era obter todas as classificações de uma única classe e suspeitamos que isso ocorria devido ao desbalanceamento da amostra, que foi arrumado utilizando pesos para as classes, cuja solução é apresentada no código de classificação. Junto a essa solução, os hiperparâmetros que apresentaram a melhor classificação foram os seguintes:

```
# Data augmentation utilizada
datagen <- image_data_generator(
  validation_split = 0.3,
  rotation_range = 20,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE
)
```

4 Ajuste da rede neural

Definidas as configurações de pré-processamento, a rede neural foi ajustada novamente com sugestões do ChatGPT com algumas alterações. A implementação tem os seguintes passos:

1. Carregamento de pacotes;
2. Definição do pré-processamento;
3. Definição do diretório de dados;
4. Definição dos conjuntos de treinamento e de validação;
5. Sequenciamento do modelo;
6. Compilação do modelo.

O passo 3 foi definido conforme o código a seguir:

```
train_dir <- "/content/Treino"
# contem as pastas com labels de male e female
```

O passo 4 ocorreu como segue. A função recebe o diretório dos dados de treinamento (com as imagens de homens e mulheres em pastas distintas) e o objeto com as especificações de pré-processamento. `target_size` é definido para o tamanho das imagens, de modo que não reverta para o padrão de 256x256 pixels, com cores em escala de cinza e um batch size de 32, por padrão. Como a classificação de fato é binária — homens e mulheres — faz sentido manter o argumento de `class_mode`. O subset é indicado para utilizar aquele que foi indicado no pré-processamento. O argumento `shuffle` não foi alterado, utilizando `TRUE` como padrão e embaralhando as imagens para favorecer uma amostragem probabilística. O argumento `classes` foi incluido na mesma ordem de exibição das pastas no diretório de treino para atribuir classes identificáveis.

```
train_generator <- flow_images_from_directory(
  train_dir,
  generator = datagen,
  target_size = c(96, 103),
  color_mode = "grayscale",
  batch_size = 32,
  class_mode = "binary",
  subset = "training",
  classes = c('female', 'male')
)

#image_Dataset_From_directory: labels na ordem

validation_generator <- flow_images_from_directory(
  train_dir,
  generator = datagen,
  target_size = c(96, 103),
  color_mode = "grayscale",
  batch_size = 32,
  class_mode = "binary",
  subset = "validation",
  classes = c('female', 'male')
)
```

As etapas 5 e 6 foram definidas considerando a arquitetura de rede sugerida pelo ChatGPT, com alterações. A arquitetura sugerida é composta por três camadas convolucionais, que seguem o exemplo dado em aula. A quantidade de filtros aumenta na medida em que a imagem vai sendo reduzida a subprodutos de dimensões menores e é seguida de uma camada densa com 512 neurônios. Enquanto todas as ativações são ReLU, assumindo valores zero ou positivos, a última camada de ativação é uma sigmóide, que faz sentido com a classificação binária.

Considerando as aulas, decidimos incluir duas camadas *fully connected* ao final, cada uma com 256 neurônios, com drop-out rate de 0.5. A camada de saída com um neurônio parece adequada com a atividade proposta de classificação.

Não foram feitas alterações na compilação do modelo.

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
  ↪  input_shape = c(96, 103, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  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_flatten() %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")

# Compile the model
model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(lr = 0.0001),
  metrics = c("accuracy")
)
```

Considerando ainda as aulas sobre otimização e sugestões de listas anteriores, optamos por incluir *early stopping* para evitar *overfitting* e possivelmente reduzir tempo computacional infrutífero. O *early stopping* é definido em função dos resultados da função de perda e o treinamento será interrompido se não houver melhora em 10 épocas. Em seguida, o modelo será salvo em `best_model.h5` e apenas o melhor modelo será salvo.

```
early_stopping <- callback_early_stopping(
  monitor = "val_loss",
  patience = 10,
  restore_best_weights = TRUE
)

model_checkpoint <- callback_model_checkpoint(
  filepath = "best_model.h5",
  monitor = "val_accuracy",
  save_best_only = TRUE
)
```

Por fim, o modelo é treinado, utilizando como argumentos todas as especificações dadas anteriormente. Os pesos das classes são definidos em `class_weights` para lidar com o desbalanceamento de classes da amostra.

```
class_weights <- list('0' = 3260 / (3260 + 740), '1' = 740 / (3260 + 740))

history <- model %>% fit(
  train_generator,
  steps_per_epoch = ceiling(train_generator$n / train_generator$batch_size),
  epochs = 50,
  validation_data = validation_generator,
  validation_steps = ceiling(validation_generator$n /
  ↪  validation_generator$batch_size),
  callbacks = list(early_stopping, model_checkpoint),
  class_weight = class_weights
)
```

5 Classificação dos dados de teste

A classificação dos dados de teste foi feita com o melhor modelo gerado (conforme definições de *callback*). Além disso, foi necessário indicar que não há label para esses dados, assim como indicar que as imagens estão em escala de cinza e com as mesmas dimensões das imagens de treino.

Outro hiperparâmetro que foi ajustado manualmente foi o limiar de classificação. O limiar que parecia apresentar resultados mais condizentes com a amostra de treinamento era de 0.7, visto que o primeiro quartil das probabilidades geradas como resultado era de aproximadamente 0.69.

O resultado de `summary(predictions)` gerava a seguintes medidas de posição:

```
Min.   :0.4378
1st Qu.:0.6937
Median :0.7370
Mean   :0.7263
3rd Qu.:0.7752
Max.   :0.8715
```

```
best_model <- load_model_hdf5("best_model.h5")

test_images <- image_dataset_from_directory("/content/Teste",
                                            label_mode = NULL,
                                            color_mode = "grayscale",
                                            image_size = c(96, 103))

predictions <- predict(best_model, test_images)

# Interpret the predictions
predicted_labels <- ifelse(predictions > 0.7, "male", "female")
```

Finalmente, os resultados foram salvos em um arquivo `csv` com o nome e classificação correspondente.

```r
# Get the filenames
test_image_filenames <- list.files("/content/Teste", recursive = TRUE, full.names =
↪  TRUE)
test_image_filenames <- basename(test_image_filenames)

# Create a data frame with filenames and predicted labels
results <- data.frame(
  Filename = test_image_filenames,
  PredictedLabel = predicted_labels
)

# Export results to CSV
write_csv(results, "predictions_no_rescale.csv")
```