Practice 4: Convolutional Networks and Malaria

Statistical Learning with Deep Artificial Neural Networks

Alexander J Ohrt

15 april, 2022

Contents

1 Introduction

In this practice we will apply a convolutional neural networks (CNN) to build a classifier that allows us to discriminate images from uninfected and infected patients.

2 Implementation of CNN

We implement the following CNN

```
model <- keras_model_sequential() %>%
layer_conv_2d(filters=32, kernel_size=3, activation = "relu", input_shape=c(64, 64, 3)) %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_conv_2d(filters=64, kernel_size=3, activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_conv_2d(filters=128, kernel_size=3, activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_conv_2d(filters=128, kernel_size=3, activation = "relu") %>%
layer_max_pooling_2d(pool_size = c(2,2)) %>%
layer_flatten() %>%
layer_dense(units = 512, activation = 'relu') %>%
layer_dense(units = 1, activation = 'sigmoid')
summary(model)
```

```
#> Model: "sequential_4"
#> ________________________________________________________________________________
#>  Layer (type)                                  Output Shape
#> ================================================================================
#>  conv2d_15 (Conv2D)                            (None, 62, 62, 32)
#>
#>  max_pooling2d_15 (MaxPooling2D)               (None, 31, 31, 32)
#>
#>  conv2d_14 (Conv2D)                            (None, 29, 29, 64)
#>
#>  max_pooling2d_14 (MaxPooling2D)               (None, 14, 14, 64)
#>
#>  conv2d_13 (Conv2D)                            (None, 12, 12, 128)
#>
#>  max_pooling2d_13 (MaxPooling2D)               (None, 6, 6, 128)
#>
#>  conv2d_12 (Conv2D)                            (None, 4, 4, 128)
#>
#>  max_pooling2d_12 (MaxPooling2D)               (None, 2, 2, 128)
#>
#>  flatten_3 (Flatten)                           (None, 512)
#>
#>  dense_7 (Dense)                               (None, 512)
#>
#>  dense_6 (Dense)                               (None, 1)
#>
#> ================================================================================
#> Total params: 504,001
#> Trainable params: 504,001
#> Non-trainable params: 0
#> ________________________________________________________________________________
```

The model is compiled below

```
  model %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)
```

The data (images) is loaded

```
# Not sure how to load the data right now.
# Could do something like this (e.g.): https://tensorflow.rstudio.com/tutorials/beginners/load/load_ima

path.folder <- "malaria"

train.infected <- image_load(paste(path.folder, "train", "infected", sep = "/")) %>%
  image_to_array() %>%
  array_reshape(dim = c(1, 224, 224, 3)) %>%
  imagenet_preprocess_input()
```

The model is fit below

```
history <- model %>% fit(
  x = encoded_expression1,
  y = encoded_expression1,
  epochs = 100,
  batch_size = 64,
  validation_split = 0.2,
  callbacks = callback.parameters
)
plot(history)
```

3 `tfruns` for Hyperparameter Tuning

`tfruns` is used to optimize the hyperparameter `batch_size` on the grid 16, 32, 64.

It turns out that the best model is obtained when using the batch size ... I HAVE SKIPPED THIS PART FOR NOW. I choose 64 in order to continue.

```
batch.size = 64
```

4 Early Stopping using Callbacks

Early stopping is implemented using the `keras callbacks() early-stopping API` in order to avoid overfitting. Training is interrupted when validation accuracy stops improving for more than two epochs.

```
callback.parameters <- callback_early_stopping(
  monitor = "val_loss",
  patience = 2,
  verbose = 1,
  mode = "min",
  restore_best_weights = FALSE
)

history <- model %>% fit(
  x = encoded_expression1,
```

```
  y = encoded_expression1,
  epochs = 100,
  batch_size = batch.size,
  validation_split = 0.2,
  callbacks = callback.parameters
)
plot(history)
```

5 Performance Assessment of CNN

The categories of test images are predicted using the CNN and the confusion matrix is shown.

```
# keras/tensorflow version >= 2.6
# se obtiene un objeto tf.tensor
y_pred <- model %>% predict(x_test) %>% k_argmax()
# se pasa a vector
# https://tensorflow.rstudio.com/guide/tensorflow/tensors/
y_pred <- y_pred %>% shape() %>% unlist()
confusionMatrix(as.factor(DATATESTLABELS), as.factor(y_pred))
```

6 Convolutional Autoencoder (CAE)

In this section we implement a CAE with 10 nodes in z layer (or bottleneck). The same number of convolutional layers, filter sizes, number of filters, pooling layers and dense layers as in the CNN are used.

```
# Based o: https://blog.keras.io/building-autoencoders-in-keras.html
#### Convolutional Encoder
model_enc <- keras_model_sequential()
model_enc %>%
layer_conv_2d(filters = 16, kernel_size = c(3,3),
activation = "relu", padding = "same",
input_shape = input_dim) %>%
layer_max_pooling_2d(pool_size = c(2,2), padding = "same")
layer_conv_2d(filters = 8, kernel_size = c(3,3),
activation = "relu", padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2,2), padding = "same")
layer_conv_2d(filters = 8, kernel_size = c(3,3),
activation = "relu", padding = "same") %>%
layer_max_pooling_2d(pool_size = c(2,2), padding = "same")
summary(model_enc)

#### Convolutional Decoder
model_dec <- keras_model_sequential()
model_dec %>%
layer_conv_2d(filters = 8, kernel_size = c(3,3),
activation = "relu", padding = "same",
input_shape = c(4, 4, 8)) %>%
layer_upsampling_2d(size = c(2,2)) %>%
layer_conv_2d(filters = 8, kernel_size = c(3,3),
activation = "relu", padding = "same")
layer_upsampling_2d(size = c(2,2)) %>%
# Important: no padding
layer_conv_2d(filters = 1, kernel_size = c(3,3),
```

```
activation = "relu") %>%
layer_upsampling_2d(size = c(2,2))
summary(model_dec)

model.CAE <- keras_model_sequential()
# input dimension == output dimension
#### Autoencoder
model.CAE %>% model_enc %>% model_dec
summary(model.CAE)
```

We compile and fit the model.

```
model %>% compile(
  loss = "mean_squared_error",
  #optimizer = optimizer_rmsprop(),
  optimizer = "adam",
  metrics = c("mean_squared_error")
)

history <- model %>% fit(
  x= x_train_cifra, y = x_train_cifra,
  # Autoencoder
  epochs = 5, batch_size = 128,
  suffle = TRUE,
  validation_split = 0.2
)
```

We do predictions.

```
# Autoencoder
output_cifra <- predict(model,x_test_cifra)
dim(output_cifra)

# From input to encoder
enc_output_cifra<-predict(model_enc,x_test_cifra)
dim(enc_output_cifra)

# From encoder to decoder
dec_output_cifra<-predict(model_dec,enc_output_cifra)
dim(dec_output_cifra)
```

7 Graphical Representation of Results

In this section we present graphically the results from the test images to show the association between z layer activations and the class images.