CAE-MNIST_E.R

evegas

2022-04-05

```
# Convolutional Autoenconder  (CAE) digits from MNIST dataset

library(keras)


#### Data
mnist <- dataset_mnist()
```

```
## Loaded Tensorflow version 2.7.0
```

```
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y


#dim(x_train)
#dim(y_train)

######################## Selection several digits

cifra <- c(0:9)

x_train_cifra<-x_train[which(y_train %in% cifra),,]
y_train_cifra<-y_train[which(y_train %in% cifra)]

x_test_cifra<-x_test[which(y_test %in% cifra),,]
y_test_cifra<-y_test[which(y_test %in% cifra)]


dim(x_train_cifra)
```

```
## [1] 60000    28    28
```

```
length(y_train_cifra)
```

```
## [1] 60000
```

```
dim(x_test_cifra)
```

```
## [1] 10000    28    28
```

```
length(y_test_cifra)
```

```
## [1] 10000
```

```
unique(y_train_cifra)
```

```
##  [1] 5 0 4 1 9 2 3 6 7 8
```

```
unique(y_test_cifra)
```

```
##  [1] 7 2 1 0 4 9 5 6 3 8
```

```
#dim(x_train_cifra)


# Input image dimensions
img_rows <- 28  # dim(x_train_cifra)[2]
img_cols <- 28  # dim(x_train_cifra)[3]

# Redefine dimension of train/test inputs
x_train_cifra <- array_reshape(x_train_cifra, c(nrow(x_train_cifra), img_rows, img_cols, 1))
x_test_cifra <- array_reshape(x_test_cifra, c(nrow(x_test_cifra), img_rows, img_cols, 1))
input_dim <- c(img_rows, img_cols, 1)

######################### rescale

x_train_cifra <- x_train_cifra / 255
x_test_cifra <- x_test_cifra / 255
#y_train_cifra <- to_categorical(y_train_cifra, 10)
#y_test_cifra <- to_categorical(y_test_cifra, 10)


################################ Autoencoder
# Based on
# 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)
```

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  conv2d_2 (Conv2D)                  (None, 28, 28, 16)              160
##
```

```
##  max_pooling2d_2 (MaxPooling2D)     (None, 14, 14, 16)              0
##
##  conv2d_1 (Conv2D)                  (None, 14, 14, 8)               1160
##
##  max_pooling2d_1 (MaxPooling2D)     (None, 7, 7, 8)                 0
##
##  conv2d (Conv2D)                    (None, 7, 7, 8)                 584
##
##  max_pooling2d (MaxPooling2D)       (None, 4, 4, 8)                 0
##
## =================================================================================
## Total params: 1,904
## Trainable params: 1,904
## Non-trainable params: 0
## _________________________________________________________________________________
```

```
#### 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: "sequential_1"
## _________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =================================================================================
##  conv2d_5 (Conv2D)                  (None, 4, 4, 8)                 584
##
##  up_sampling2d_2 (UpSampling2D)     (None, 8, 8, 8)                 0
##
##  conv2d_4 (Conv2D)                  (None, 8, 8, 8)                 584
##
##  up_sampling2d_1 (UpSampling2D)     (None, 16, 16, 8)               0
##
##  conv2d_3 (Conv2D)                  (None, 14, 14, 1)               73
##
##  up_sampling2d (UpSampling2D)       (None, 28, 28, 1)               0
##
## =================================================================================
## Total params: 1,241
## Trainable params: 1,241
## Non-trainable params: 0
## _________________________________________________________________________________
```

```
# input dimension  == output dimension

#### Autoencoder

model<-keras_model_sequential()
model %>%model_enc%>%model_dec
```

```
## Model: "sequential_2"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  sequential (Sequential)            (None, 4, 4, 8)                 1904
##
##  sequential_1 (Sequential)          (None, 28, 28, 1)               1241
##
## =========================================================================
## Total params: 3,145
## Trainable params: 3,145
## Non-trainable params: 0
## _________________________________________________________________________
```

```
############################################################

summary(model)
```

```
## Model: "sequential_2"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  sequential (Sequential)            (None, 4, 4, 8)                 1904
##
##  sequential_1 (Sequential)          (None, 28, 28, 1)               1241
##
## =========================================================================
## Total params: 3,145
## Trainable params: 3,145
## Non-trainable params: 0
## _________________________________________________________________________
```

```
######################################## Training


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
#  validation_data = list(x_test_cifra,x_test_cifra)
```

```
)


######## Prediction

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

```
## [1] 10000    28    28     1
```

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

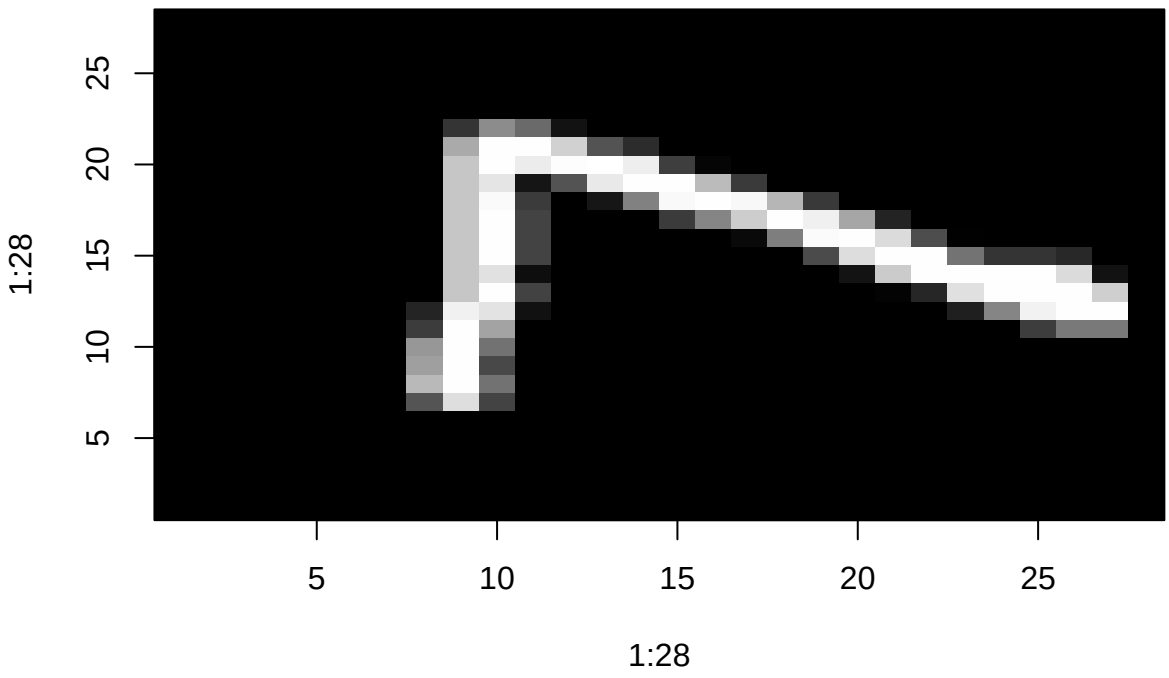
```
## [1] 10000     4     4     8
```

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

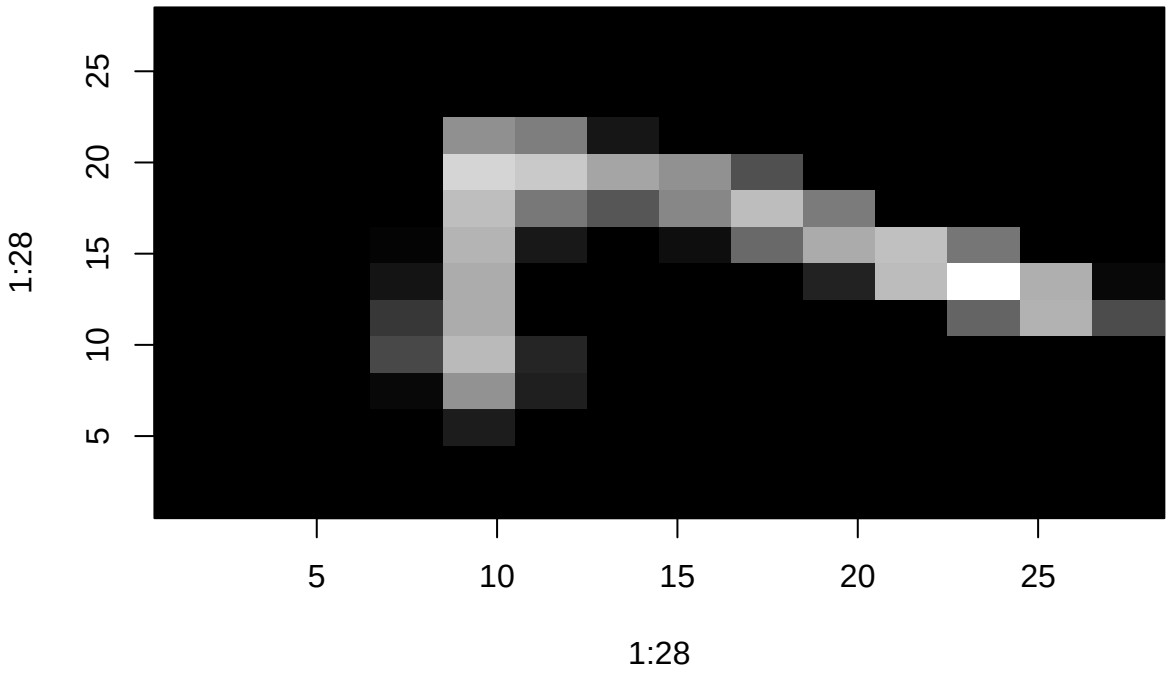
```
## [1] 10000    28    28     1
```

```
# Check

idx<-1
#x_test_cifra[idx,,,1]
im<-matrix(x_test_cifra[idx,,,1], nrow=28, ncol=28)
image(1:28, 1:28, im, col=gray((0:255)/255))
```



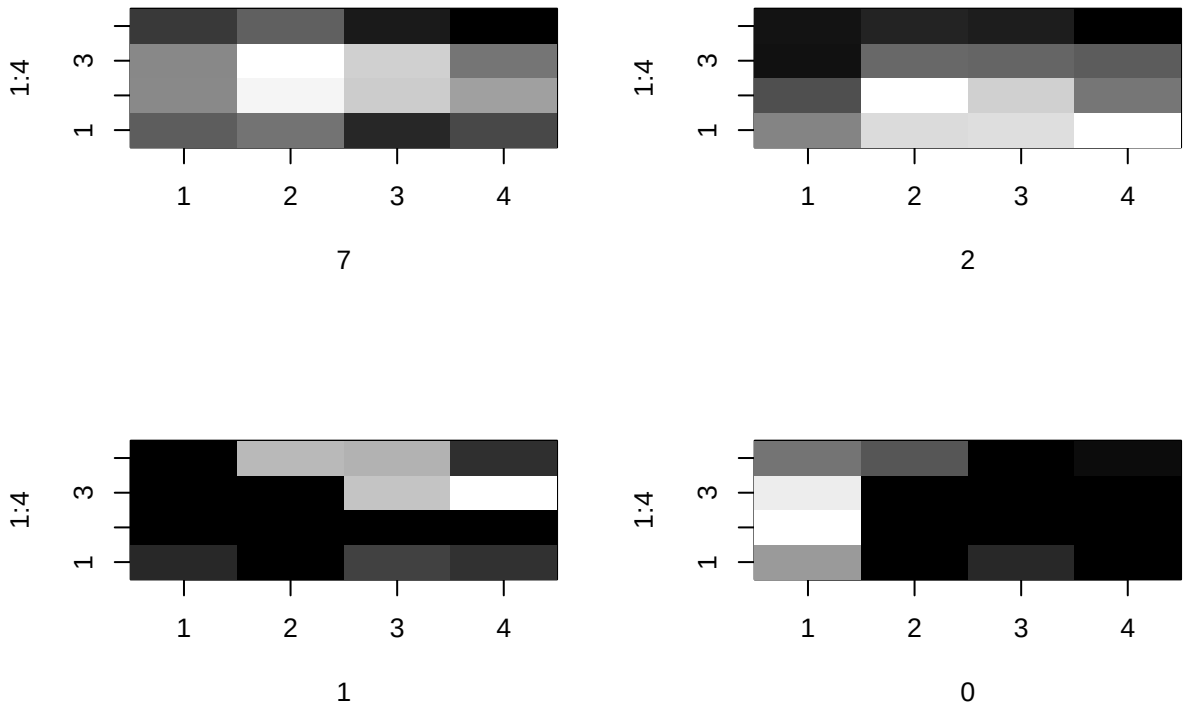

```
#output_cifra[idx,,,1]
im<-matrix(output_cifra[idx,,,1], nrow=28, ncol=28)
image(1:28, 1:28, im, col=gray((0:255)/255))
```




```
#dec_output_cifra[idx,,,1]
im<-matrix(dec_output_cifra[idx,,,1], nrow=28, ncol=28)
image(1:28, 1:28, im, col=gray((0:255)/255))

#which.max(enc_output_cifra[idx,,,1])
#which.min(enc_output_cifra[idx,,,1])
#which(enc_output_cifra[idx,,,1]==cifra)

# Encoder results
# dim(enc_output_cifra)
par(mfrow = c(2,2))
for (k in 1:4){
  im<-matrix(enc_output_cifra[idx,,,k], nrow=4, ncol=4)
  image(1:4, 1:4, im, col=gray((0:255)/255),
        xlab = y_test_cifra[k])
}
```




```
par(mfrow = c(1,1))
############################################################


#dim(x_train_cifra)
#dim(x_test_cifra)

# Save encode digit in a Rdata file

save(enc_output_cifra, y_test_cifra, file=paste0("Conv_Encod_orig", paste(cifra, collapse = ""), ".RData

# flat file

dim(enc_output_cifra) <- c(nrow(enc_output_cifra),prod(dim(enc_output_cifra)[-1]))

# Flatten array
save(enc_output_cifra, y_test_cifra, file=paste0("Conv_Encod_Flat_", paste(cifra, collapse = ""), ".RDa




# flat file
#enc_output_cifra_flat <- enc_output_cifra
#dim(enc_output_cifra_flat) <- c(nrow(enc_output_cifra),prod(dim(enc_output_cifra)[-1]))

# Flatten array
```

```
# save(enc_output_cifra_flat, y_test_cifra, file=paste0("Conv_Encod_flat", paste(cifra, collapse = ""),
```