Project 2

Anton Holm

2022-02-11

```
suppressWarnings(suppressMessages(library(reticulate)))
suppressWarnings(suppressMessages(library(tensorflow)))
suppressWarnings(suppressMessages(library(keras)))
suppressWarnings(suppressMessages(library(tidyverse)))
```

I was having Certification problems when trying to download the data. This python code solved that problem which was written by Denise Mak on stackoverflow (*https://stackoverflow.com/questions/69687794/unable-to-manually-load-cifar10-dataset*)

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

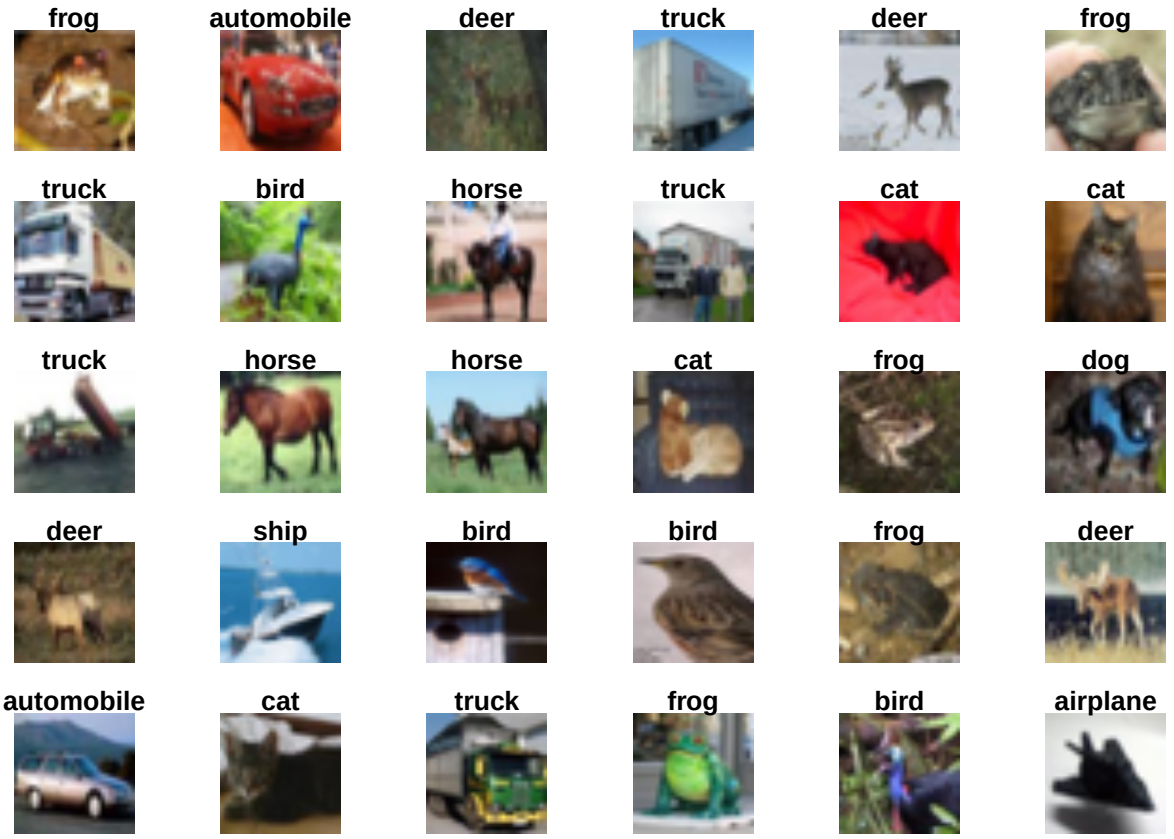
Firstly we load the cifar10 dataset which contains 60.000 color pictures.

```
cifar <- dataset_cifar10()
```

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

Next we take a look at some of the pictures from the dataset to see that everything went well when importing the data.

```
class_names <- c('airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck')

index <- 1:30

par(mfcol = c(5,6), mar = rep(1, 4), oma = rep(0.2, 4))
cifar$train$x[index,,,] %>%
  purrr::array_tree(1) %>%
  purrr::set_names(class_names[cifar$train$y[index] + 1]) %>%
  purrr::map(as.raster, max = 255) %>%
  purrr::iwalk(~{plot(.x); title(.y)})
```



We can now start constructing the convolution part of our model. The CNN part of our model is constructed by taking as input images with 3 channels (RGB) and a size of 32x32 pixels. We then apply 3x3 kernels where we go from 3 to 32 channels while downsampling the pixelsize to 30x30 and apply max pooling where the pixelsize is downsampled further to a size of 15x15 pixels. Next we apply new kernels, going from 32 to 64 channels and decrease the pixelsize from 15x15 to 13x13. Furthermore, we apply another maxpooling layer, going from 13x13 to 6x6 pixelsize followed by the last layer where we apply new kernels, keeping the total number of channels while decreasing the pixel size to 4x4. We use relu as activation function in our convolution layers. A brief summary of the CNN structure can be seen below.

```
model_conv <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(32,32,3)) %>%
  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 = 64, kernel_size = c(3,3), activation = "relu")
```

```
summary(model_conv)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                        Output Shape                    Param #
## ================================================================================
##  conv2d_2 (Conv2D)                   (None, 30, 30, 32)              896
##
```

```
##  max_pooling2d_1 (MaxPooling2D)     (None, 15, 15, 32)              0
##
##  conv2d_1 (Conv2D)                  (None, 13, 13, 64)              18496
##
##  max_pooling2d (MaxPooling2D)       (None, 6, 6, 64)                0
##
##  conv2d (Conv2D)                    (None, 4, 4, 64)                36928
##
## ================================================================================
## Total params: 56,320
## Trainable params: 56,320
## Non-trainable params: 0
## ________________________________________________________________________________
```

In order to be able to reproduce the results in this report we need to set seed since we have randomness in both weight initialization (which the tutorial skipped) and in the Adam algorithm. We now construct the FF NN part of our model. A summary of the full model can be seen below.

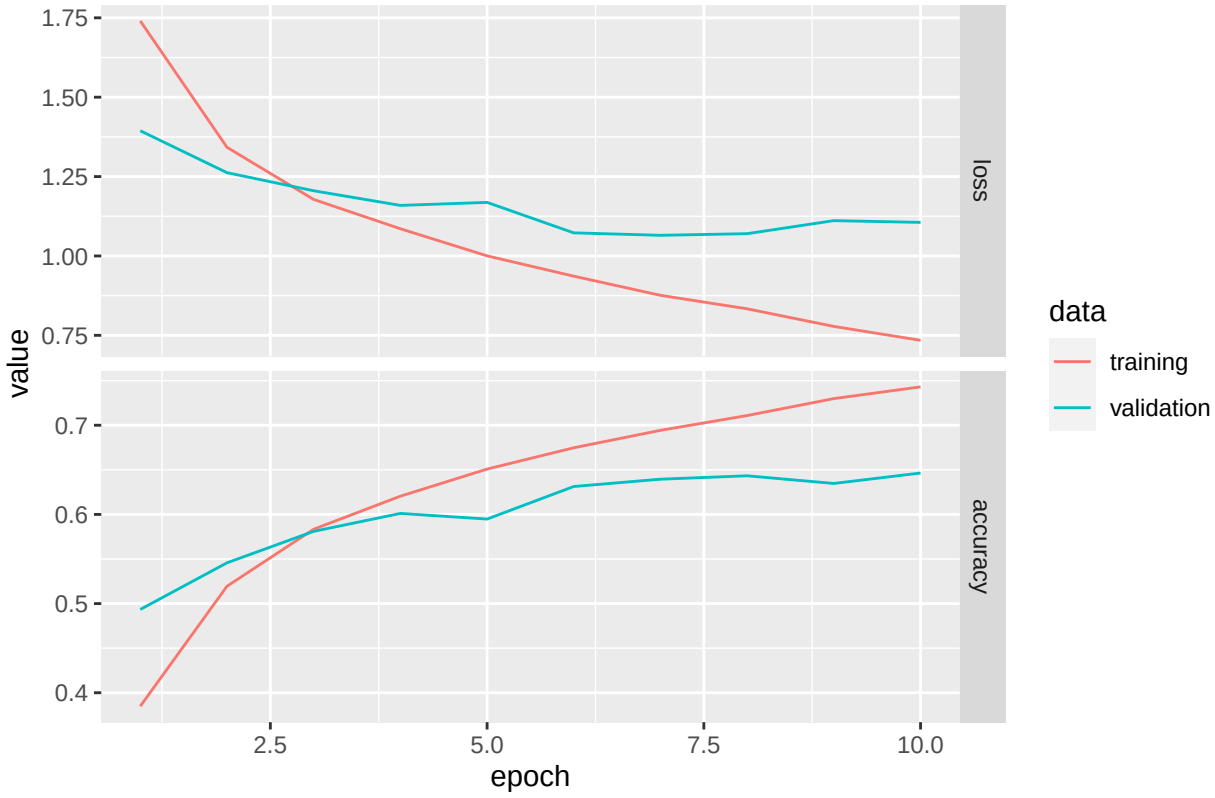
```
set_random_seed(931031)

#Maybe add random initiation on weights
model_conv %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

summary(model_conv)
```

```
## Model: "sequential"
## ________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## ================================================================================
##  conv2d_2 (Conv2D)                  (None, 30, 30, 32)              896
##
##  max_pooling2d_1 (MaxPooling2D)     (None, 15, 15, 32)              0
##
##  conv2d_1 (Conv2D)                  (None, 13, 13, 64)              18496
##
##  max_pooling2d (MaxPooling2D)       (None, 6, 6, 64)                0
##
##  conv2d (Conv2D)                    (None, 4, 4, 64)                36928
##
##  flatten (Flatten)                  (None, 1024)                    0
##
##  dense_1 (Dense)                    (None, 64)                      65600
##
##  dense (Dense)                      (None, 10)                      650
##
## ================================================================================
## Total params: 122,570
## Trainable params: 122,570
## Non-trainable params: 0
## ________________________________________________________________________________
```

Finally we start training the model. We can see in Figure 1 that the validation error has started to plateau and seem to begin increasing towards the latest epochs. We can also see that the accuracy of the validation set has plateaued. This would imply that the model has converged.

```
model_conv %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy", #Use one vector with factors 1-10 for classes instead of on
  metrics = "accuracy"
)
# Training time around 1 minute per epoch
history <- model_conv %>%
  fit(
    x = cifar$train$x, y = cifar$train$y,
    epochs = 10,
    validation_data = unname(cifar$test),
    verbose = 2 #Only show loss/acc for the last batch in the epoch
  )

history %>%
  as_tibble() %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free") +

  labs(title = "Figure 1: Training and validation loss using relu activation function")
```

Figure 1: Training and validation loss using relu activation function



We can now use our test data to see how well our model performs.

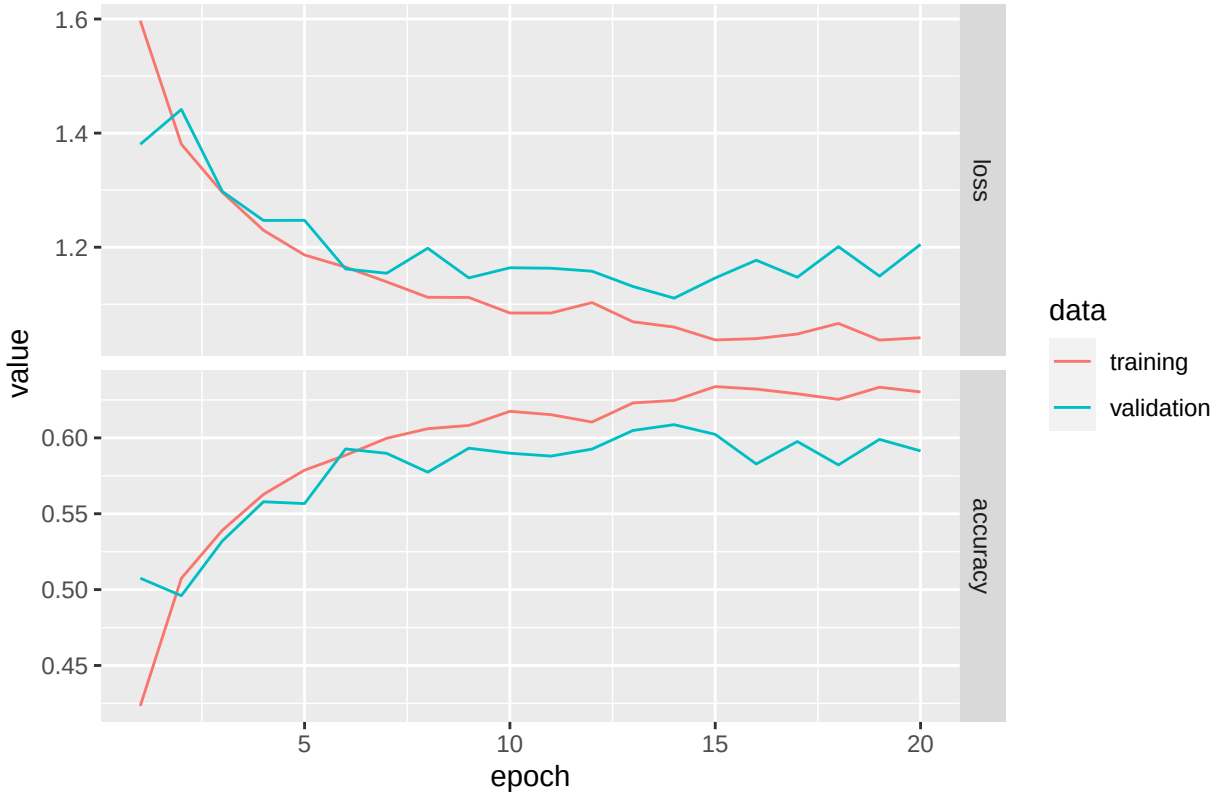
```
test_score <- evaluate(model_conv, cifar$test$x, cifar$test$y, verbose = 0)

test_score
```

```
##     loss accuracy
## 1.105695 0.646400
```

We now repeat the procedure using tanh as an activation function instead.

```
model_conv_tanh <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "tanh",
                input_shape = c(32,32,3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "tanh") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "tanh")

model_conv_tanh %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "tanh") %>%
  layer_dense(units = 10, activation = "softmax")


model_conv_tanh %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy", #Use one vector with factors 1-10 for classes instead of on
  metrics = "accuracy"
)

history_tanh <- model_conv_tanh %>%
  fit(
    x = cifar$train$x, y = cifar$train$y,
    epochs = 20,
    validation_data = unname(cifar$test),
    verbose = 2 #Only show loss/acc for the last batch in the epoch
  )

history_tanh %>%
  as_tibble() %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free") +

  labs(title = "Figure 2: Training and validation loss using tanh activation function")
```

Figure 2: Training and validation loss using tanh activation function



We see in Figure 2 that the validation loss seem to start increasing again after around 13 epochs.

```
test_score_tanh <- evaluate(model_conv_tanh, cifar$test$x, cifar$test$y, verbose = 0)

test_score_tanh
```

```
##     loss accuracy
## 1.205018 0.591400
```

One of the issues of using tanh in this situation is the max pooling layers. When using tanh, project each output to a span between -1 and 1. However, when we afterwards apply the max pooling layer, we neglect largly negative pixels. We could fix this if we were to use absolute max pooling instead.

Lastly we want to test out using only 4 units in the last layer, keeping everything else the same. The code can be seen below but since it gives an error when running it, the code will not be ran and as such no information or plots will be shown. The error given mentions that we get label values outside the interval $[0, 4)$.

First we need to consider the loss function used. Due to the fact that we have our labels ranging from 0 to 9, i.e. as integers and not as one-hot-vectors, we use the loss function "sparse categorical crossentropy". If we instead would have one-hot-vector representation we would use "categorical crossentropy". Both loss functiosn use the same loss, what differs them is how we represent our labels. So, since we use the sparse categorical crossentropy, the program expects integer values as labels. However, since we only use 4 units, we can only express 4 classes because we use integer representation. Hence, when we have labels larger than 3 (since we include 0 as a class), we get the error since our loss expects a value of 0, 1, 2 or 3.

```r
model_conv_tanh2 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "tanh",
                input_shape = c(32,32,3)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "tanh") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "tanh")

model_conv_tanh2 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "tanh") %>%
  layer_dense(units = 4, activation = "softmax") #This does not work


model_conv_tanh2 %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history_tanh2<- model_conv_tanh2 %>%
  fit(
    x = cifar$train$x, y = cifar$train$y,
    epochs = 20,
    validation_data = unname(cifar$test),
    verbose = 2 #Only show loss/acc for the last batch in the epoch
  )

history_tanh2 %>%
  as_tibble() %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free") +

  labs(title = "Figure 3: Loss using only 4 units in output layer")
```