Convolutional Neural Networks for CIFAR-10 Image Classification

From now, the neural network computations will require to be executed on GPU instead of CPU. Because they are more complex and CPU won't be enough powerful for running our models. For the moment, we use the GPU inside Google colab. Later, we will provide you with a technical report to activate your GPU on your computer if you have a GPU on your computer.

How to activate GPU in google colab:

In order to activate GPU on colab, go to edit tab of your project. Select Notebook setting and choose GPU from the hardware accelerator section. To check if GPU is working on the google colab run:

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
```

```
import tensorflow as tf
tf.test.gpu_device_name()
```

If it is the case, you should receive a message as `'/device:GPU:0'`.
In order to check which GPU are you using:

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

In general for more information about the using RAM or CPU, you can use the following codes respectively:

```
!cat /proc/meminfo
```

```
!cat /proc/cpuinfo
```

Exercises

Exe. 1 Load the `CIFAR-10` dataset from Keras[1].
Exe. 2 Convert train and test X values to float32 and normalize data between

[1] `https://www.cs.toronto.edu/~kriz/cifar.html`

range 0.0 and 1.0. (Since that the pixel values fall in the range of 0 to 255, normalise train and test sets.)
Hint:

```
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D

keras.layers.Conv2D(filters, kernel_size =, strides=, padding= ,
    data_format=, activation= , input_shape= )
```

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If activation is not None, it is applied to the outputs as well. When using this layer as the first layer in a model, provide the keyword its input shape. For more details check Keras Documentation

```
keras.layers.MaxPooling2D(pool_size=, strides=, padding=)
```

Max pooling operation for spatial data.
Arguments:

- pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
- strides: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.
- padding: One of "valid" or "same" (case-insensitive).

Exe. 3 Define a network model (NET I) with the following architecture:

$$CONV - POOL - CONV - POOL - CONV - POOL - FC - FC$$

In this architecture, 8 layers comprising alternating sequence of convolution neural network (CONV) and Maxpooling (POOL) layers are used. All three 2D convolution layers comprise filters of size 3×3 (in the `conv2D()`, define `kernel_size = 3`). Varying number of used channels for three Conv are 32, 64 and 64 respectively. 2×2 max pooling layers with stride of 2 are used. The two fully connected layers are of size 512 and 10 respectively. In all but the last FC layer, a reLU activation function is used. A SoftMax activation function is used for the final layer. The input size of the network was set that of CIFAR-10 RGB images, i.e. $32 \times 32 \times 3$. Before calling the first FC layer, do not forget to flatten output of the previous layer using `keras.layers.Flatten()`.
Exe. 4 The number of parameters in a convolutional layer is given by:

$$P_n = [(m \times n \times C_i) + 1] \times C_o$$

where P_n is the number of parameters, $m \times n$ is the filter dimension, C_i and C_o are the input and output channel (feature map) dimensions respectively for a

convolutional layer. The addition of one to the inner factor accounts for biases applied to the feature maps. Also, the dimensions of the output data matrix, O_n, upon filtering or pooling is given by the equation below:

$$O_n = \frac{N - F}{S} + 1$$

where N is the input dimension and F is the filter dimension similar to $m \times n$: $m = n = F$, and S is the filter stride length. Pooling layers do not offer any parameters to be optimized as it is a mere computation of average or the maximum value captured by the pooling filter.
compute the P_n and O_n of your network and verify your calculation by printing `model.summary()`.
Exe. 5 Compile the model with an `adam` optimizer and a `sparse_categorical_crossentropy` loss function.
Exe. 6 Train the model with 20% of validation data with 128 batches and 20 epochs. Save the fitting result in a variable called `history`.
In `model.fit( validation_split)`, validation_split is a float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the X and y data provided, before shuffling. This argument is not supported when X is a generator or Sequence instance.
Notice: *what is validation data?*
If you want to build a solid model you have to follow that specific protocol of splitting your data into three sets: training, validation and test set (evaluation). The idea is that you train on your training data and tune your model with the results of metrics (accuracy, loss etc) that you get from your validation set.
Your model doesn't "see" your validation set and isn't in any way trained on it, but you as the architect and master of the hyperparameters tune the model according to this data. Therefore it indirectly influences your model because it directly influences your design decisions. You nudge your model to work well with the validation data and that can possibly bring in a tilt.
Exactly that is the reason you only evaluate your models final score on data that neither your model nor you yourself has used – and that is the third chunk of data, your test set. Only this procedure makes sure you get an unaffected view of your models quality and ability to generalize what is has learned on totally unseen data.
Exe. 7 After fitting the model, evaluate it on the test set using `model.evaluate()`. Print accuracy and loss of the evaluated model on the test set.
Exe. 8 As a test, predict the test dataset using `model.predict()`, and check if the predicted image label is the same as the real label for the first image of the test set. Notice that, in classification problems, the predicted output is a vector with a size of total number of classes where each element represents a probability of each class.
Exe. 9 Using the `history` saved in Exe. 6, demonstrate the `'accuracy'`

(train accuracy changing), versus `'val_accuracy'` (validation accuracy changing) changing regarding epochs evolutions in a graph. To better understanding of the history `print(history.history.keys())`. What are your observations? How is the deviation between two curves?

Exe. 10 Similar to the previous Exe., demonstrate the training loss (`'loss'`) versus the validation loss (`'val_loss'`) w.r.t the epochs evolutions. What are your observations? After how many epochs, the overfitting happens?

Exe. 11 Define a new model (NET II) with higher filter/kernel size (filter size: 5 * 5) as below:

$$CNV - CONV - POOL - CONV - POOL - FC - FC$$

This architecture adopts 7 layers comprising the 3 convolution layers (same as before) and 2 reduced number of pooling layers. The sequential layer organization is implemented as: two convolution layers followed by a max pooling layer, another convolution layer, a max pooling layer and finally two fully connected layer of size 512 and 10 respectively as classification is being done into 10 classes. In all but the last fully connected layer, a reLU activation function is used. A SoftMax activation function is used for the final layer. The network input size adopted is same as that used in NET I. In the NET II model, increase the convolution filter size to (5×5) in comparison with the NET I, which has a smaller filter size (3×3). Before calling the first FC layer, do not forget to flatten output of the previous layer using `keras.layers.Flatten()`

Exe. 12 Print the summary of this model, check the number of parameters using the formula given in Exe. 4.

Exe. 13 Compile the model with the same parameters of Exe. 5

Exe. 14 Fit the model same as Exe. 6. Do not forget to save it as history in the same time.

Exe. 15 Evaluate the model on test dataset and test a first image of test set as Exe. 8

Exe. 16 Similar to Exes 9 and 10, demonstrate how the loss and accuracy change for the training and validation according to epochs? What are your observations? how is the overfitting now? after how many epochs happens? How is the deviation between training and validation in comparison to NET I model?

Note: In general, a smaller filter/ kernel size can capture complex non-linearities in the input as a lesser number of pixels are averaged at an instance, thus variations in data are better sustained.

Exe. 17 Define a deeper model (NET III) as an improvement of NET I and NET II:

$$CONV-CONV-POOL-CONV-POOL-CONV-CONV-CONV-POOL-FC-FC-FC$$

This model comprises of six convolutional layers with channel size 96, 96, 128, 128, 128 and 128. Each of these layers, employs a filter size of 3×3, a stride length of 1, and reLU activation functions. The choices of 96 and 128 were inspired by the AlexNet and VGGNet architectures. Three max pooling layers

were also incorporated with size of 2×2 each and stride of 2. To address overfitting, a drop rate of 50% after each pooling layer and in the fully connected layers as well as $L2$ regularization (with hyperparameter of 0.0005) in the fully connected layers were applied. Also, three fully connected layers (instead of two as in the earlier networks) of sizes 1024, 512 and 10 with reLU activation for the first two and softmax for the final layer.

Exe. 18 Do the same things as Exes. 12 to 16 for this model too. How are the results? Do you have overfitting in this model too? how is the deviation between training and evaluation accuracy?

Exe. 19 Based on your observations until now, can you get better results by modifying the model architecture, regularization or optimisation?