Deep Learning
SS 2024
TU Ilmenau
Dr. Marco Seeland & Prof. Patrick Mäder

Graded Assignment 2

Deadline: Friday, June 7, at 9:00 am.

Submission: You must submit your solution as `ipynb` (Jupyter notebook) file via Moodle. You should attempt all tasks and questions in the assignment. Most of them can be done and answered at least partially even if you were unable to finish earlier tasks or questions. If you think your computational results are incorrect, say so - it might help you get partial credit.

Note: You are expected to work on the assignments by yourself. Sharing solutions and submitting copied solutions will be reported as attempted fraud. **If you use external sources, e.g., ChatGPT, reference their usage at each place! State your prompts and critically discuss the outputs. Any use of external resources without proper referencing will result in a deduction of points and we will reserve further measures.**

Tasks

1 Prepare Your Jupyter Notebook [3 pts]

Navigate to colab.research.google.com and create a new Jupyter notebook named

`DL24S_A2_FirstName_Surname.ipynb`.

> **Note:** For training your networks, make sure you activate hardware acceleration by navigating to `Runtime` → `Change runtime type` and selecting `GPU` as hardware accelerator.
> To avoid running into GPU usage limits, we recommend to **first work on the implementation without hardware acceleration** and only switch to GPU when you are ready to train your models.

Create a new code cell, paste the following lines into it, then run the cell:

```
module = "utils.py"
url = "https://cloud.tu-ilmenau.de/s/bdsqAywoLfca9e4/download"
!wget -nv -t 0 --show-progress -O $module $url
```

The code cell above downloads a `utils` module for *Graded Assignment 2*. Import `numpy`, `pyplot`, and the `utils` module in a new code cell and download the dataset for this assignment:

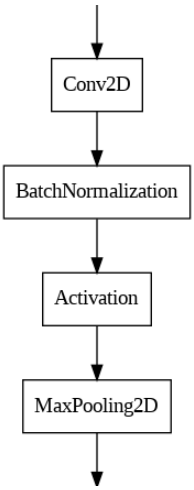

Figure 1: Architecture of one convolutional block.

```
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import utils

(train_ds, val_ds, test_ds), class_labels = utils.download_dataset()
```

The dataset is already provided as `tf.data.Dataset`. Take 16 samples from the train dataset and plot them as images along with their associated labels as provided in `class_labels`. Write a preprocessing function to resize the images to a shape of `(128, 128, 3)`. Do not normalize the images at this point, you will incorporate the normalization in the model. Apply the preprocessing function to the training, validation, and test datasets.
At last, create a function named `optimize_for_train()`. The function shall take a dataset as input and return a batched dataset allowing for efficient processing by

1. caching,
2. shuffling of the data (only if training data),
3. batching with a batch size of 32,
4. and prefetching

in this order.

2 Small ConvNet [5 pts]

Create a new section entitled `"2 - Small ConvNet"`. In this section, you shall create a small convolutional neural network (ConvNet) and train it on the provided dataset.
Create a `build_convnet()` function that returns the ConvNet model. The model shall be built according to the following specifications:

- Add a rescaling layer to the model to rescale the input images to a range of `[0, 1]`.
- Use optional augmentation layers to augment the input images. Add a boolean argument named `augment` to the function signature to control the augmentation.

- Use augmentation layers for the following operations:
  - random horizontal flip,
  - random rotation by up to 36°,
  - random zoom by up to 20%, and
  - random contrast with a factor of max 20%.
- The ConvNet shall consist of `num_blocks` convolutional blocks. The architecture of a block is displayed in Fig. 1. The first block shall use `num_filters` filters. In each subsequent block, the number of filters shall be doubled with respect to the previous block.
- Each 2D convolutional layer shall use filters of size `(size_filter, size_filter)`, linear activation, padding to maintain the input size, and a stride of 1.
- Use rectified linear unit (ReLU) activation function in the blocks.
- 2D max pooling shall use a filter size of 2.
- The head shall be constituted of a 2D GlobalAveragePooling layer with 30% dropout and a Dense layer for classification across the classes of your dataset.

Using parameters (`augment`, `num_blocks`, `num_filters`, and `size_filter`) of your choice, initialize one model and print the model summary. Investigate the ConvNet architecture. Given the list of layers as displayed in Fig. 1, discuss the role of the bias terms in the convolutional layers in a **markdown cell**. Based on your findings, implement the bias terms in your `build_convnet()` function accordingly. Also think about the correct size and activation function of the output layer and justify your choices in a markdown cell.

3 Experiments [8 pts]

Create a section "3 - Experiments". Next, create a function called `train_model()` that trains your ConvNet model. Given `augment`, `num_blocks`, `num_filters`, and `size_filter`, the `train_model()` function shall

- Initialize a ConvNet model using your `build_convnet()` function.
- Compile the model using the Adam optimizer with a learning rate of `0.003`, the sparse categorical crossentropy loss function, and the sparse categorical accuracy metric.
- Print the model summary.
- Reduce the learning rate by a factor of `0.2` if the validation loss does not improve for 30 epochs, check the ReduceLROnPlateau callback.
- Train the model for a maximum of 200 epochs and validate after each epoch.
- Stop the training if the validation loss does not improve for 60 epochs and return the best model.
- Return the trained model and history.

Use the `optimize_for_train()` function to create batched datasets for the train, validation, and test data.

3.1 Number of Blocks

Create the subsection "3.1 - Number of Blocks" by executing the following markdown cell:

```
##3.1 - Number of Blocks
```

Disable the augmentation for this experiment and use `num_filters = 32` and `size_filter = 3` for all models. For an increasing number of convolutional blocks (`num_blocks` ∈ `[2, 3, 4]`), train your model using your `train_model()` function.
For each model, use `utils.plot_history()` to plot the training and validation history and evaluate the model on the test data.
Take the best performing model architecture, enable augmentation, and train the model again from scratch. Plot the training and validation history and discuss the model's behavior in comparison to the model without augmentation in a markdown cell.

3.2 Filter Size

Create the subsection "3.2 - Filter Size". Disable the augmentation again and use `num_filters = 32` and `num_blocks = 3` for all models. For filter sizes of `(3x3, 5x5, 7x7)`, train your model using your `train_model()` function. **For each model**, use `utils.plot_history()` to plot the training and validation history and evaluate the model on the test data.

3.3 Results & Discussion

Create the subsection "3.3 - Results & Discussion". Aggregate the number of blocks, filter size, number of parameters and your results in a table and discuss the results in a markdown cell. Below is a markdown template for the table:

```
| No. Blocks | Filter Size | No. Params | Train Acc | Test Acc |
|------------|-------------|------------|-----------|----------|
| 3          | 3x3         |            |           |          |
| ...        | ...         | ...        | ...       | ...      |
```

4 Fine-Tuning [4 pts]

Create a new section "4 - Fine-Tuning". Import a pre-trained EfficientNetB0 from the TF Keras Applications module. Initialize the EfficientNetB0 model with the weights pre-trained on the ImageNet dataset and set the input shape to `(128, 128, 3)`.
Make a suitable choice for the new head of the model, freeze the EfficientNetB0 and train the new head for five epochs using the same learning rate as before. Plot the training and validation history and evaluate the model on the test data.
Finally, fine-tune the entire model for a maximum of 50 epochs. Make a suitable choice for the learning rate and use the `ReduceLROnPlateau` callback to reduce the learning rate by a factor of 0.1 if the validation loss does not improve for 10 epochs. Stop the training if there is no improvement in the validation loss for 15 epochs. Plot the training and validation history, evaluate the model on the test data, and discuss the results.

Submit for Grading

Download your notebook as Jupyter notebook

`File` → `Download` → `Download .ipynb`.

Make sure that your cells outputs are saved! Navigate to `Moodle` and submit your downloaded notebook.