DLCV – Challenge 1
Classical neural network for clothing image classification

Description

In this challenge, the main goal is to develop a classical neural network to classify "mini-images" of clothing, footwear and bags. The dataset to use is FASHION_MNIST[1], which is also provided by `tensorflow/keras`[2]. Some examples of images contained in the dataset can be observed in Figure 1.

Figure 1 - Examples of FASHION_MNIST images

To perform this exercise, you can start with the code found in the `fashionNet.py` script provided together with this document. The script includes an instruction that automatically downloads the dataset, so you'll need an internet connection.

The FASHION_MNIST consists of 60,000 samples for training and 10,000 samples for testing. The downloaded dataset does not contain a validation set and this will be an issue to consider during this work assignment. Each sample is a grayscale image with a dimension of 28x28 pixels. There are 10 classes that represent the types of clothing, footwear and bags present in the dataset:

Class Id	Meaning	Class Id	Meaning
0	*T-shirt/top*	**5**	*Sandal*
1	*Trouser*	**6**	*Shirt*
2	*Pullover*	**7**	*Sneaker*
3	*Dress*	**8**	*Bag*
4	*Coat*	**9**	*Ankle boot*

Although in this case we use image data instead of tabular data, a classical neural network architecture can be applied by using the luminance value of each pixel as an attribute (*feature*).

However, since images are 2D matrices, they need to be "vectorized" in order to be treated as feature vectors (similarly to tabular data). This task can be easily performed by including a layer of the `Flatten` type, similarly to as done as in the following example:

```
myModel = keras.Sequential([
                layers.Input(shape=(28, 28)),
                layers.Flatten(),

                # from now on, we have feature vectors, not 2D matrices …
```

[1] https://github.com/zalandoresearch/fashion-mnist
[2] https://www.tensorflow.org/api_docs/python/tf/keras/datasets

Work assignment

Part 1 – Implement and evaluate a neural network for multiclass classification

Implement a neural network for multiclass classification[3]:

1. Get a validation set to monitor network training (split the original training set).
2. Build a suitable neural network model that is able to classify according to the 10 classes of the *dataset* (see the previous section).
3. Compile the network, choosing a suitable loss function and an optimization algorithm.
4. Train the model – it is suggested to set a maximum of 50 epochs and to use *callbacks* (check the appendix at the end of this document) – you are expected to achieve an accuracy rate of about 85-90% in the test set.
5. Produce plot(s) showing the accuracy and loss evolution during the training process.
6. Compute and show the accuracy rate on the test set.
7. Show a confusion matrix, analyze it, and check which classes the network most easily confuses with each other.

Part 2 – Assemble and evaluate a neural network for binary classification

In this part of the assignment, you will develop another neural network, using FASHION_MNIST, but this time the goal is to perform a binary classification into two classes: *Clothing* and *Footwear and Bags*.

This binary classification neural network should be deployed in another *script* or *notebook*.

1. Make the necessary adaptations to the dataset in order to obtain binary labels: 1 could represent the *Clothing* class, while 0 could represent the *Footwear and Bags* class (or vice versa) – check the table in the previous section, in order to map the original ten FASHION_MNIST classes into these two classes;
2. Adapt steps b) to g) of Part 1 to the neural network developed for the binary case.

Part 3 – Question

Perform the necessary calculations and comparisons to answer the following question (and justifying it):

For the Clothing/Footwear and Bags binary classification, is it better to use a neural network for multiclass classification and then binarize the network predictions, or is it better to directly use a neural network specifically designed for the binary classification?

---

[3] Similar to the example of the wine makers, presented in the classes of February 27 and 28.

Delivery

The delivery of the challenge is carried out through Moodle (one submission per group). The submitted materials must include the developed *scripts* or *notebooks*. For those submitting *python scripts*, please include a short report (max. 4 pages) showing the most relevant results and comments. For those delivering *notebooks*, make sure to include the results and comments in the *notebook* (they'll serve as report).

The deadline for this work's submission is **March 9 (Sunday) at 11:59 pm**. Submissions after the deadline are accepted but penalized with 1 point (out of 10) for each delayed day.

Evaluation

The grade will be a 0 to 10 points score, and feedback will be provided through Moodle. The contributions of each part to the final score are as follows: Parts 1 and 2 – 40% each; Part 3 – 20%.

In doubt, feel free to contact the teacher.

Appendix – *Callbacks*

Callbacks can be described as utility mechanisms that allow automatic function calls while the neural network is being trained (typically the auto function calls are performed at the end of each epoch).

In the provided `fashionNet.py` script, two *callbacks* have been defined:

1. `BEST_MODEL_CHECKPOINT` – allows saving a file with the weights of the best model obtained so far during training, according to a certain criterion. In the callback that has been defined, the "best model" will be the one achieving the lowest loss value in the validation set (see `monitor` and `mode` parameters). The file to save is specified in the `filepath` parameter (in the script it was set to `"tmp/best_model.weights.h5"`).

2. `EARLY_STOPPING` – allows stopping the training process if there is no improvement on the loss function during a specified number of consecutive epochs – the `patiente` parameter.

A list of the *callbacks* to be used during the training process is supplied to the `fit(...)` function, using the parameter `callbacks` as follows:

```
history = myModel.fit( … , callbacks=[BEST_MODEL_CHECKPOINT, EARLY_STOPPING])
```

After training the neural network, the weights of the "best model" saved can be restored using the `load_weights function`:

```
myModel.load_weights("tmp/best_model.weights.h5")
```

From there, predictions can be performed using the restored model. This approach usually provides better results when compared with considering the model obtained in the last training epoch, which in most cases is not the best one.

You can find more information about *callbacks* at:

https://blog.paperspace.com/tensorflow-callbacks/
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks