Note on model download from SWITCHdrive

Due to space constraints, all the neural network models in this assigment were saved and uploaded on SWITCHdrive. To download them, make sure you have `curl`, `gzip`, and `tar` (all installed by default on MacOS) and then run:

```
./download.sh
```

having as current working directory (pwd) the root assigment directory `as2_Maggioni_Claudio`.
The script will download automatically the models.
Then, in order to run the models with the `run_task*.py`, make sure you first `cd` in the `deliverable` directory.
Should the script not work, please download the compressed T1 and T2 models from `https://drive.switch.ch/index.php/s/F0ubFgS6PBy8UM5/download` and uncompress the downloaded `.tar.gz` file in the `deliverable/` directory, and then make sure the directories `deliverable/nn_task1` and `deliverable/nn_task2` contain 3 `.h5` files in total.

Machine Learning
Università della Svizzera italiana

Assignment 2

Claudio Maggioni

June 9, 2021

In this assignment you are asked to:

1. Implement a neural network to classify images from the `CIFAR10` dataset;
2. Fine-tune a pre-trained neural network to classify rock, paper, scissors hand gestures.

Both requests are very similar to what we have seen during the labs. However, you are required to follow **exactly** the assignment's specifications.

1 Follow our recipe

Implement a multi-class classifier to identify the subject of the images from `CIFAR-10` data set. To simply the problem, we restrict the classes to 3: `airplane`, `automobile` and `bird`.

1. Download and load `CIFAR-10` dataset using the following function, and consider only the first three classes. Check `src/utils.py`, there is already a function for this!
2. Preprocess the data:
   - Normalize each pixel of each channel so that the range is [0, 1];
   - Create one-hot encoding of the labels.
3. Build a neural network with the following architecture:
   - Convolutional layer, with 8 filters of size 5×5, stride of 1×1, and ReLU activation;
   - Max pooling layer, with pooling size of 2×2;
   - Convolutional layer, with 16 filters of size 3×3, stride of 2×2, and ReLU activation;

    - Average pooling layer, with pooling size of 2×2;
    - Layer to convert the 2D feature maps to vectors (Flatten layer);
    - Dense layer with 8 neurons and tanh activation;
    - Dense output layer with softmax activation;

4. Train the model on the training set from point 1 for 500 epochs:
    - Use the RMSprop optimization algorithm, with a learning rate of 0.003 and a batch size of 128;
    - Use categorical cross-entropy as a loss function;
    - Implement early stopping, monitoring the validation accuracy of the model with a patience of 10 epochs and use 20% of the training data as validation set;
    - When early stopping kicks in, and the training procedure stops, restore the best model found during training.
5. Draw a plot with epochs on the x-axis and with two graphs: the train accuracy and the validation accuracy (remember to add a legend to distinguish the two graphs!).
6. Assess the performances of the network on the test set loaded in point 1, and provide an estimate of the classification accuracy that you expect on new and unseen images.
7. **Bonus** (Optional) Tune the learning rate and the number of neurons in the last dense hidden layer with a **grid search** to improve the performances (if feasible).
    - Consider the following options for the two hyper-parameters (4 models in total):
        - learning rate: [0.01, 0.0001]
        - number of neurons: [16, 64]
    - Keep all the other hyper-parameters as in point 3.
    - Perform a grid search on the chosen ranges based on hold-out cross-validation in the training set and identify the most promising hyper-parameter setup.
    - Compare the accuracy on the test set achieved by the most promising configuration with that of the model obtained in point 4. Are the accuracy levels statistically different?

1.1 Comment

The network model was built and trained according to the given specification.
The performance on the given test set is of 0.3879 loss and 85.63% accuracy. In order to assess performance on new and unseen images a statistical confidence interval is necessary. Since the accuracy is by construction a binomial measure (since an image can either be correctly classified or not, and we repeat this Bernoulli process for each test set datapoint), we perform a binomial distribution confidence interval computation for 95% confidence. The code use to do this is found in the notebook `src/Assignment 2.ipynb` under the section *Statistical tests on CIFAR classifier*. We conclude stating that with 95% confidence the accuracy for new and unseen images will fall between $\approx$ 84.29% and $\approx$ 86.82%.
The training and validation accuracy curves for the network is shown below:

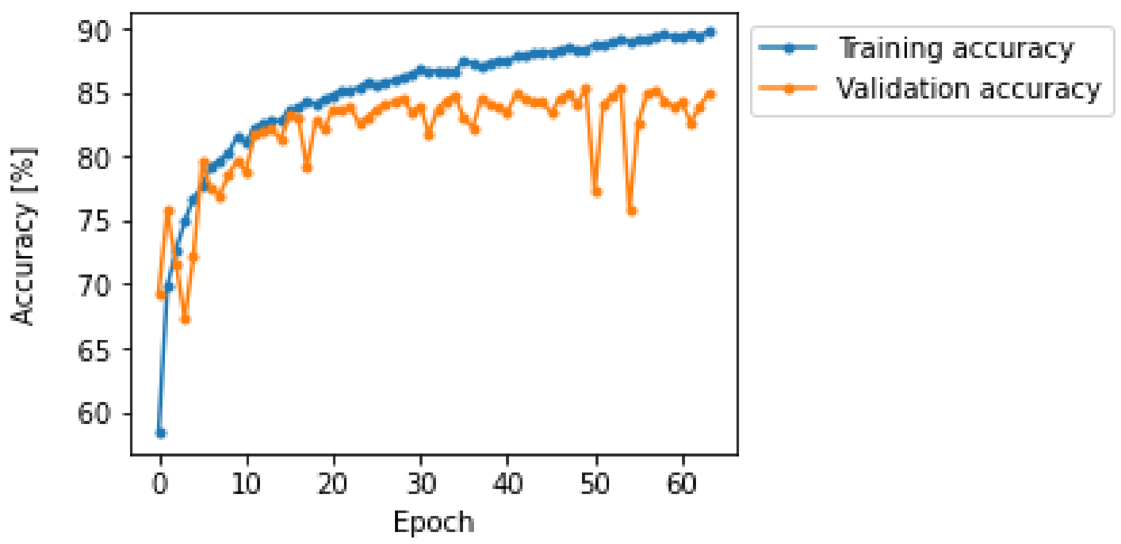

Figure 1: Training and validation accuracy curves during fitting for the CIFAR10 classifier

1.2 Bonus

A plot of the validation loss and accuracy over training epochs for each of the configurations in the grid search are given below. The plot is useful to understand that for both learning rates chosen, choosing 64 neurons will lead to overfitting, due to the validation loss being higher than the training loss. This overfitting cannot be easily avoided with the adopted early stopping procedure, since our implementation monitors validation loss and not validation accuracy.
The second observation that can be found from the plots is that a smaller learning rate increases the number of epochs needed to achieve convergence but it also increases the model accuracy and decreases the model loss.
By running all the models on the test set, using the script `run_task1_bonus.py` in the `deliverable/` directory, we obtain the following loss and accuracy:

Learn rate	# Neurons	Loss	Accuracy
0.01	16	0.4597	0.8117
0.01	64	0.4508	0.8290
0.0001	16	0.4048	0.8437
0.0001	64	0.4073	0.8343

We choose the third configuration as the optimal one since it has highest accuracy (and lowest loss too). We then perform a T-test between this model and the one built in the previous section using the script `src/t_test_bonus.py`.
The T-test gives a T-score of 1.374106, corresponding to a P-value of 0.169511, which makes us accept the null hypothesis therefore concluding that there is no significant difference between this optimal model and the one trained in the previous section.

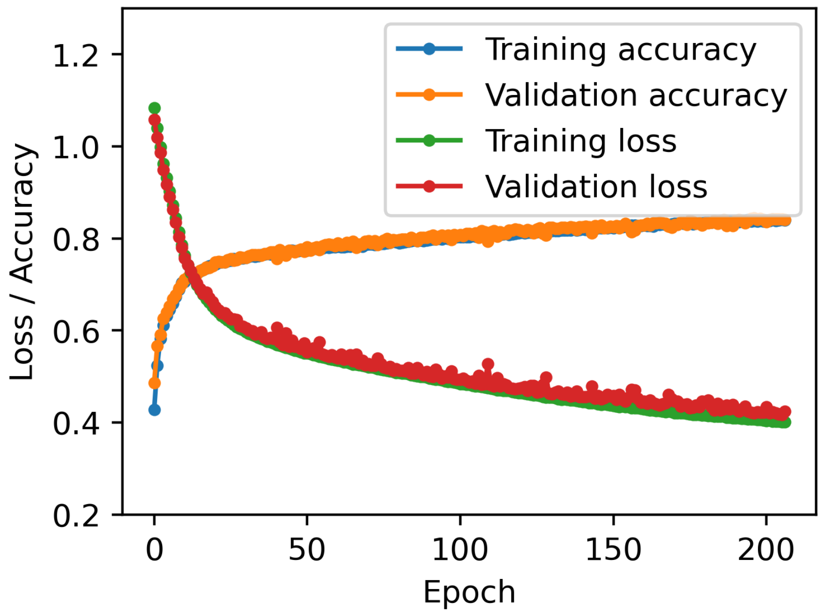

Figure 2: Training and validation accuracy curves during fitting for the CIFAR10 classifier (0.0001 learning rate, 16 neurons)

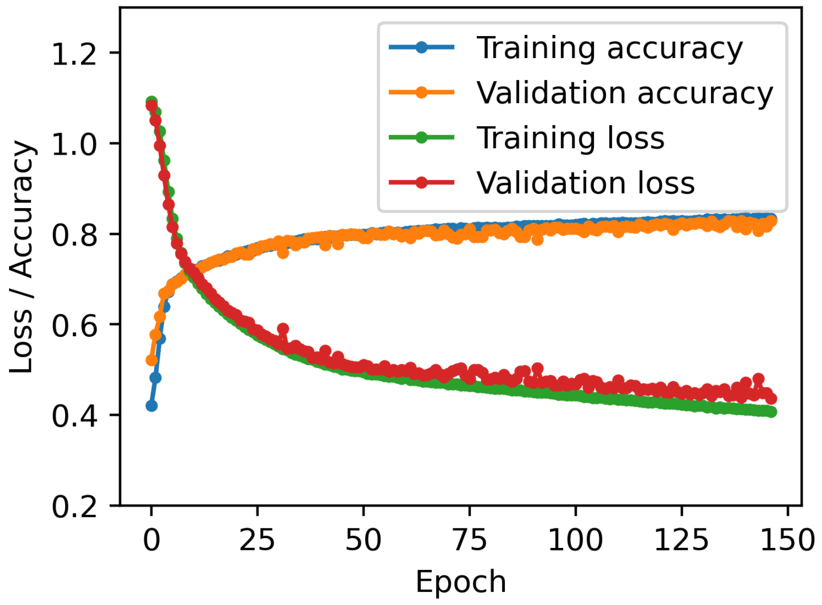

Figure 3: Training and validation accuracy curves during fitting for the CIFAR10 classifier (0.0001 learning rate, 64 neurons)

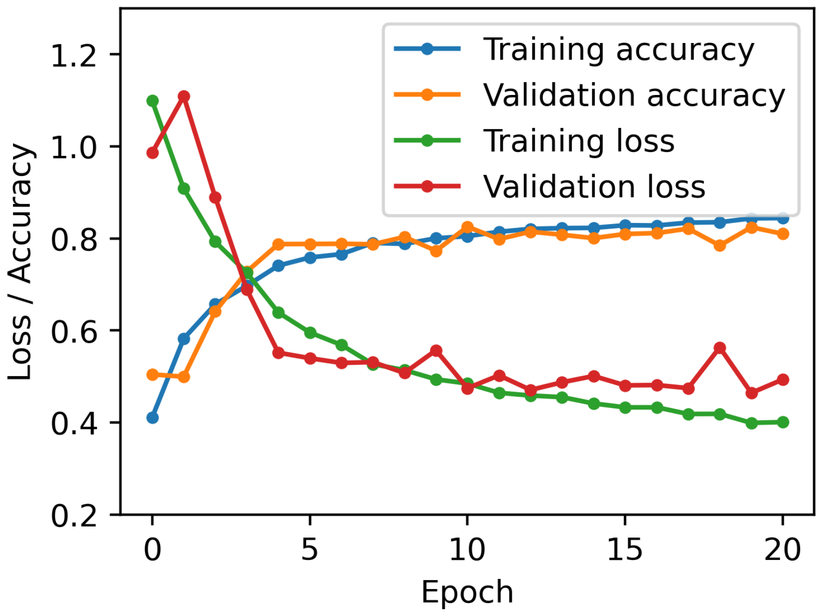

Figure 4: Training and validation accuracy curves during fitting for the CIFAR10 classifier (0.01 learning rate, 16 neurons)

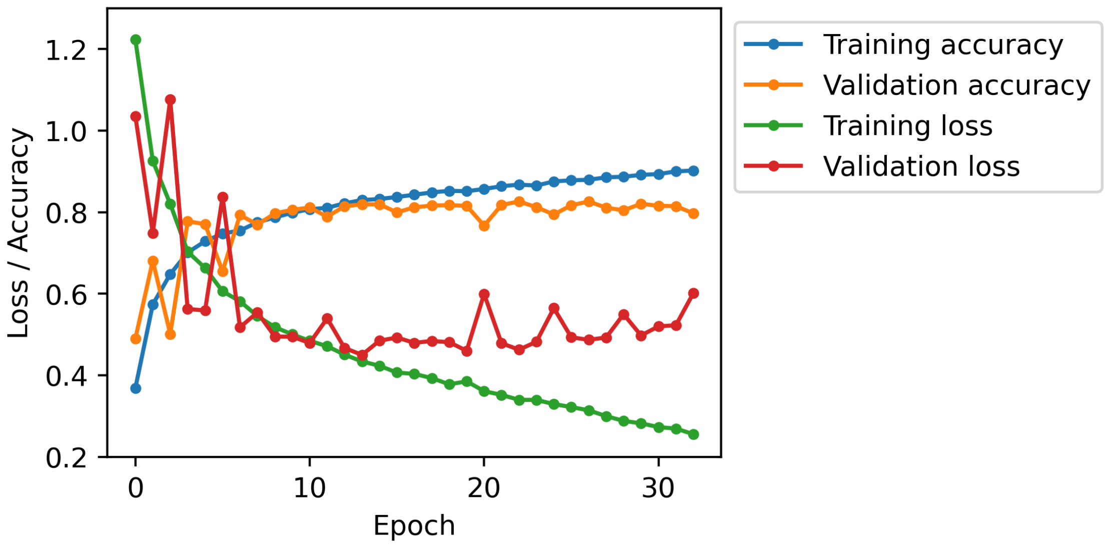

Figure 5: Training and validation accuracy curves during fitting for the CIFAR10 classifier (0.01 learning rate, 64 neurons)

2 Transfer learning

In this task, we will fine-tune the last layer of a pretrained model in order to build a classifier for the *rock, paper, scissors dataset* that we acquired for the lab. The objective is to make use of the experience collected on a task to bootstrap the performances on a different task. We are going to use the `VGG16` network, pretrained on Imagenet to compete in the ILSVRC-2014 competition.

`VGG16` is very expensive to train from scratch, but luckily the VGG team publicly released the trained weights of the network, so that people could use it for transfer learning. As we discussed during classes, this can be achieved by **removing the last fully connected layers** form the pretrained model and by using the output of the convolutional layers (with freezed weights) as input to a **new fully connected network**. This last part of the model is then trained from scratch on the task of interest.

1. Use `keras` to download a pretrained version of the `vgg16` network. You can start from the snippet of code you find on the repository of the assignment.
2. Download and preprocess the rock, paper, scissor dataset that we collected for the lab. You find the functions to download and build the dataset in `src/utils.py`. Vgg16 provides a function to prepropress the input
   `applications.vgg16.preprocess_input`
   You may decide to use it. Use 224×224 as image dimension.
3. Add a hidden layer (use any number of units and the activation function that you want), then add an output layer suitable for the hand gesture classification problem.
4. Train with and without **data augmentation** and report the learning curves (train and validation accuracy) for both cases.
   - Turn on the GPU environment on Colab, otherwise training will be slow.
   - Train for 50 epochs or until convergence.
   - Comment if using data augmentation led to an improvement or not.

2.1 Comment

The built network in its dense part is composed by a 128-neuron ReLU-activated hidden layer and a 3-neuron softmax output layer. The input to the network is at first resized to 224x224 size and then normalized according to VGG16 normalization factors (refer to the function `process_vgg16` in the `src/Assignment 2.ipynb` notebook for details on the normalization process). Classification labels were first converted from string labels to numeric ones, (i.e. `'scissors'` = 0, `'paper'` = 1, `'rock'` = 2), and then the numeric encoding was in turn converted to a one-hot encoding using the `keras.utils.to_categorical` function.
Both the data-augmented and non-augmented network were trained using the ADAM optimizer with 0.001 learning rate for 50 epochs with an early stopping procedure with 10 epochs patience.
Both models were saved and both can be run at the same time on the given test set by executing `deliverable/run_task2.py`.
The training and validation accuracy curves for both the data-augmented and the not augmented networks are shown below:

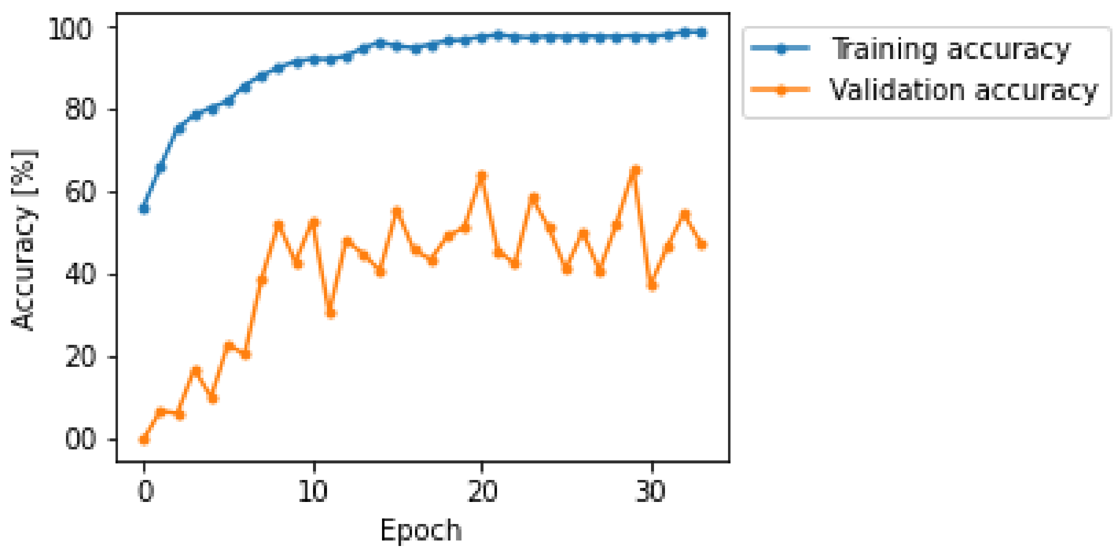

Figure 6: Training and validation accuracy curves during fitting for the not data augmented VGG16 classifier

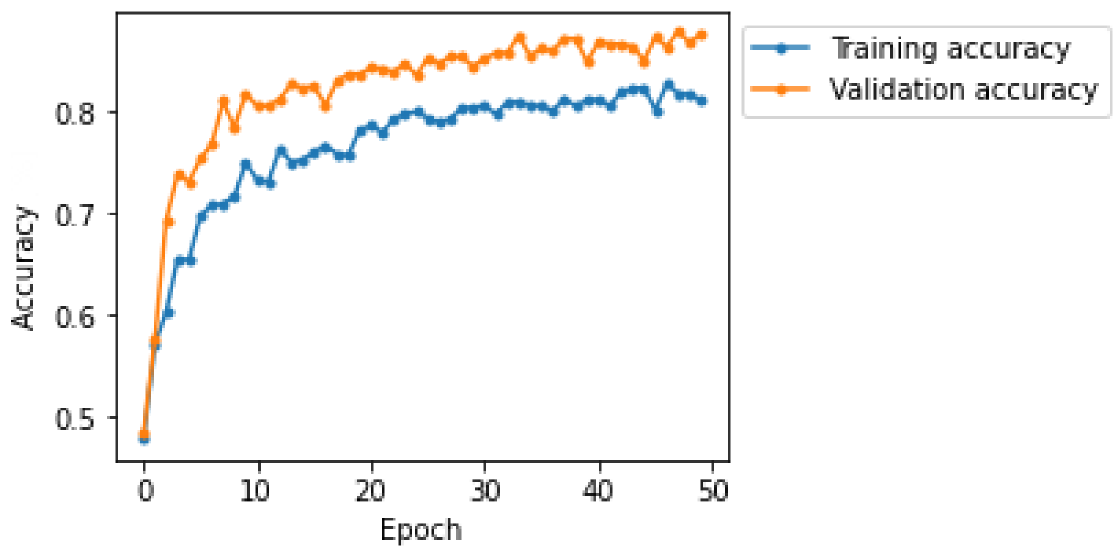

Figure 7: Training and validation accuracy curves during fitting for the data augmented VGG16 classifier

2.2 T-Test

The findings shown below were computed using the script `src/t_test.py`.
To compare the trained model with and without data augmentation, we perform a two-tailed Student T-test between the models. The test report that the models have different accuracy with 99.999973% confidence, and the model trained with data augmentation has lower variance. Therefore, we conclude that the model trained with data augmentation is the statistically better model out of the two.
The student T-test of course is a valid argument only for this specific instance of the application of data augmentation. However, in the general case we can say that performing data augmentation on the training and validation data is intuitively better in

order to assure the network is able to correctly identify rock, paper or scissors from all angles and zoom levels.
On the given test set, the model trained with data augmentation has $\approx 90.00\%$ accuracy while the model trained without data augmentation has $\approx 77.33\%$ accuracy.