Machine Learning
Università della Svizzera italiana

Assignment 2

Scannapieco Davide

June 2, 2022

In this assignment you are asked to:

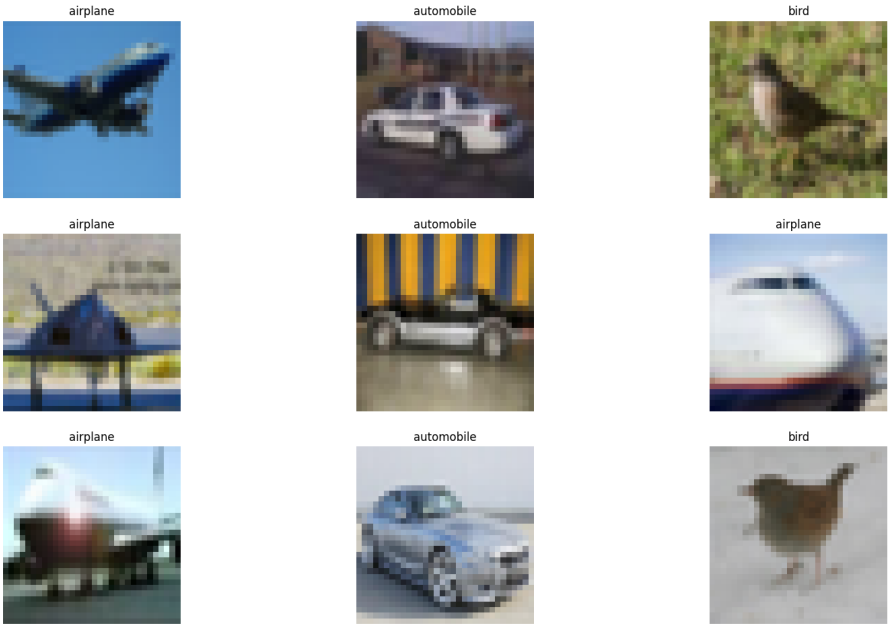
1. Implement a neural network to classify images from the `CIFAR10` dataset;
2. Implement a fully connected feed forward neural network to classify images from the `CIFAR10` dataset.

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.



3x3 images categorization

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;

   I just followed all these instruction to perform this point.

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 Results

For the task1 the code to build the model is in src/task1.py in which it creates and build the models and save the model and the model of the bonus exercise. TO get the results you need to run run_task1.py where it loads the models and print out the results. The models are created both using the CNN but the first is used to build a model based on the learning rate and number of neurons given while the second one is used to perform a grid search on the various configurations of the hyperparameters and get the best one. For this task I just do 2 for loops and then I choose the best model according to the maximum validation accuracy.

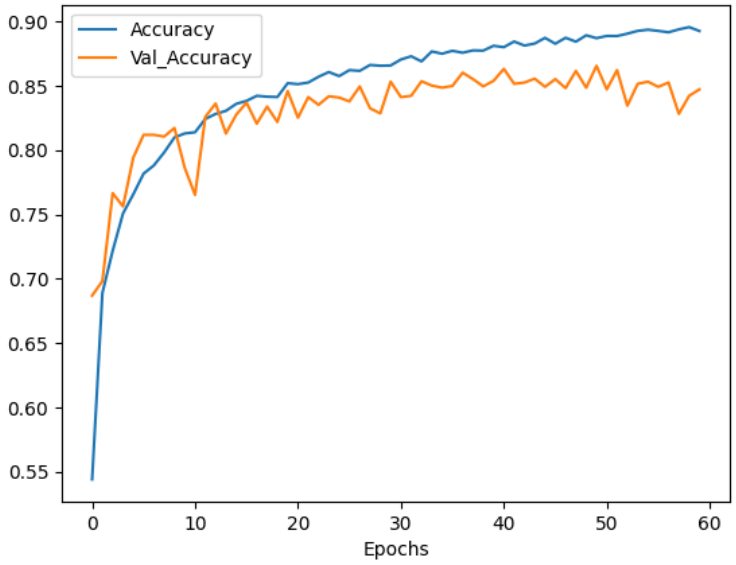
The results that I obtain are the following:

$$Loss : 0.34670695662498474$$

$$Accuracy : 0.8663333058357239$$

$$MSE : 0.06514018028974533$$

The plot obtained is:



Task 1: Plot

For the bonus exercise these are the following results:

Model	Learning Rate	Neurons	Accuracy
1	0.01	16	0.8429999947547913
2	0.01	64	0.828000009059906
3	0.0001	16	0.8066666722297668
4	0.0001	64	0.8203333616256714

We can see that the best model for this task is given with learning rate 0.01 and 16. neurons with an accuracy of 0.8429999947547913.
Here below the comparison between the best model given by the configurations and the model obtained in the task1:

Model	Loss	Accuracy
task1	0.34670695662498474	0.8429999947547913
bonus_task1	0.4070601761341095	0.8429999947547913

As we can see the base model obtained in task1 is slightly better then the bonus one. Also I performed the t-test and this is the result:

$$T - Test = -2.565541594153114$$

Since the value is outside the confidence of interval $[-1,96, 1,96]$
we take the one with best accuracy so we choose the nn_task1.h5 since it has better accuracy than the other one.

2 Image Classification with Fully Connected Feed Forward Neural Networks

In this task, we will try and build a classifier for the first 3 classes of the `CIFAR10` dataset. This time, however, we will not use a Convolutional Neural Network, but a classic Feed Forward Neural Network instead.

1. Follow steps 1 and 2 from T1 to prepare the data.
2. Flatten the images into 1D vectors. You can achieve that by using tf.reshape or by prepending a Flatten layer to your architecture; if you follow this approach this layer will not count for the rules at point 3.
3. Build a Feed Forward Neural Network of your choice, following these constraints:
   - Use only Dense layers.
   - Use no more than 3 layers, considering also the output one.
   - Use ReLU activation for all layers other than the output one.
   - Use Softmax activation for the output layer.
4. Follow step 4 of T1 to train the model.
5. Follow steps 5 and 6 of T1 to assess performance.
6. Qualitatively compare the results obtained in T1 with the ones obtained in T2. Explain what you think the motivations for the difference in performance may be.
7. **Bonus** (Optional) Train your architecture of choice (you are allowed to change the input layer dimensionality!) following the same procedure as above, but, instead of the flattened images, use any feature of your choice as input. You can think of these extracted features as a conceptual equivalent of the Polynomial Features you saw in Regression problems, where the input data were 1D vectors. Remember that images are just 3D tensors (HxWxC) where the first two dimensions are the Height and Width of the image and the last dimension represents the channels (usually 3 for RGB images, one for red, one for green and one for blue). You can compute functions of these data as you would for any multi-dimensional array. A few examples of features that can be extracted from images are:
   - Mean and variance over the whole image.
   - Mean and variance for each channel.
   - Max and min values over the whole image.
   - Max and min values for each channell.
   - Ratios between statistics of different channels (e.g. Max Red / Max Blue)
   - Image Histogram (Can be compute directly on TF Tensors or by temporarely converting to numpy arrays and using np.histogram)

   But you can use anything that you think may carry useful information to classify an image.

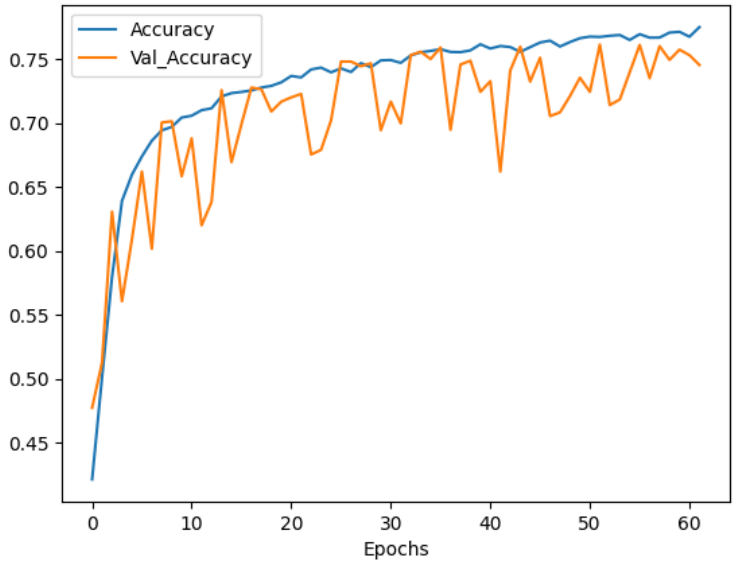
   N.B. If you carry out point 7 also consider the obtained model and results in the discussion of point 6.

2.1 Results

The file to run to get the results is run_task2.py which loads the models and print out the results. In src/task2.py you can find the implementation to build and train the model. I chose not to implement the bonus part. The model is a feed forward neural network of three dense layers each with activation function relu but the last one that has, as requested, softmax activation function. The first layer has 20 neurons, the second one 20 also while the last one (the output one) has the number of classes as number of neurons.

Loss	Accuracy
0.5980533957481384	0.765333354473114

You can also see below the figure:



Task 1: Plot

We can see that the accuracy of the T1 is better than the T2, because the convolution architecture has clear benefits compared to the FFNN one, mainly the shared weights of the convolution which are instead treated independently for a FFNN and the awareness of the spatial positioning of each pixel in the image: it is much easier to, for example, identify an edge when looking at the whole rectangular image compared to looking at its flattened representation.