CAP 6618 Spring 2019
Christian Garbin
Module 6 Assignment - Image classification using transfer learning

Objective of this assignment

The objective of this assignment is to "d*esign and implement a complete solution to the problem of image classification using the transfer learning paradigm."*

The CIFAR-10 dataset was used in the assignment.

Files submitted

Two Jupyter notebooks are submitted as part of this assignment:

1. CAP6618_PP4_cgarbin_experiments.ipynb: based on the provided CAP6618_PP4.ipynb notebook, contains experiments with transfer learning and a conclusion section at the end, explaining what approach was chosen to be developed in the final notebook.
2. CAP6618_PP4_cgarbin_final.ipynb: develops the chosen approach further.

Experiments to select a solution

In the *experiments* notebook we tried four different networks:

1. A CNN network built from scratch
2. The same CNN network, trained with data augmentation
3. A pretrained VGG19 network, with features extracted from the dense layer, then fed into a standard (not a neural network) classifier
4. A pretrained VGG16 network, with features extracted from the last convolution layer, then feed into an MLP classifier (a process called *bottlenecking*)

The results from these experiments are summarized in the following table.

Solution	Description	Test accuracy	Approximate training time
1	CNN built from scratch	68.7%	24 minutes
2	Solution 1 + data augmentation	65.4%	24 minutes
3	Pretrained VGG19, features extracted from dense layers, fed into standard classifier	33.2%	16 seconds + 5 hours for feature extraction
4	Pretrained VGG16, features extracted from convolution layers, fed into an MLP classifier	67.1%	1 minute + 12 minutes for feature extraction

Some observations collected when performing the experiments:

- Data augmentation and low number of training epochs do not always lead to better performance. Adding data augmentation resulted in slightly lower accuracy, but this may have been caused by not training long enough. If trained for more epochs, it may turn out that the network was going through a plateau.
- Pretrained networks are worth it if we can extract features once and save them, then adjust the (much smaller) trainable layers with multiple passes. Otherwise, the feature extraction process will dominate the training process to the point where it may be as long as training a (smaller, but equally good) network from scratch.
- Extracting features from dense layers leads to low accuracy (see explanation in [1], also quoted in the notebook).
- A less powerful network (VGG16) can perform better than a more powerful network (VGG19) when used correctly (summing up the lesson above). In other words, starting small may be a better strategy than going for the biggest network in the initial stages.

Solution #4 was chosen as the best solution to proceed in the separate notebook for these reasons:

1. It is faster to train, even if we count the feature extraction step. Once features are extracted, it is the fastest to train by a large margin.
2. It is based on a well-known neural network. It is a chance to experiment with that network.

Results for the final solution

In CAP6618_PP4_cgarbin_final.ipynb the final solution was developed further by changing from bottlenecking (freezing all convolution layers and extracting features from the last convolution layer), to unfreezing the last set of convolution layers and training it together with the (new top layer).

The results of this refinement are mixed:

1. Accuracy improved, but not by a significant amount (from approximately 68% to approximately 70%).
2. Training time is much larger because unfreezing the last set of convolution layers alone unlocks about half of the network weights (which then need to be retrained). Training on a GPU could compensate for that.

Applications that require maximum accuracy should unfreeze a convolution layer. Applications that require the minimum possible training time (perhaps online training, as samples are collected) should freeze all convolution layers and train only the smallest possible dense layer.

Possible next steps to improve the results

The training process is already stopping when overfit begins to happen (using early stopping), so training for longer in this configuration may not yield better results.

With that in mind, a few ideas to improve accuracy:

1. Increase the `patience` value in early stopping to check if the network is going through a plateau at that point.
2. Add data augmentation.
3. Change the architecture of the top layer, e.g. add more hidden layers, add/remove neurons, etc.

Sources

Sources consulted for this assignment.

[1] Deep Learning with Python, F. Chollet, chapter 5

[2] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html, accessed March 31st., 2019

[3] CS231n, "Transfer Learning", http://cs231n.github.io/transfer-learning/, accessed March 31st., 2019