POLITECNICO DI MILANO

MSc in Computer Science and Engineering

Artificial Neural Networks and Deep Learning

A.Y. 2020/2021

CHALLENGE 1
IMAGE CLASSIFICATION

Team: *Spatial Extent*

Nicola Rosetti 940435

Alessio Russo 945781

Vittorio Torri 945208

Custom Model

We started to approach the problem using a custom model similar to the one seen during the lab lessons. It is reported in its final version in [Notebook 1].
In its first version it consisted of a sequence of 8 blocks with conv 3x3, Relu and max pooling 2x2. There were 8 filters in the first layer and we doubled them at each layer. The top was constituted by a flatten, a dense layer with 1024 neurons and Relu activation and finally a softmax layer with 3 neurons.
We used data augmentation since the beginning, flipping the images, rotating them up to 45 degrees, zooming up to 30%, and shifting of 30-50 px.
The input size is 612 x 408, being this the most common size of the images in the training set. Initially we split the training set into training and validation sets with a proportion of 80 – 20 %.
We started using large batch sizes, between 64 and 256.
We always used early stopping in training, stopping after 5-10 epochs of no improvement on validation accuracy (later validation loss).
We started to train with a learning rate of 10^{-3} but the loss was not improving, so we decreased it to 10^{-5} and we reached a validation accuracy around 60%.
An increase in the initial number of filters from 8 to 16 and a reduction to 512 neurons in the dense layer provided a relevant gain in the validation accuracy which reached more than 75%. At this point we tried a submission on Kaggle, to be sure that we were consistent on the test set and we scored 84%.
We introduced the `ReduceLROnPlateau` callback, starting from 10^{-4}, in this way the model overfitted.
To counter this, we tried to include *Batch Normalization*, both in the convolutional layers and/or in the FC part. It helped to speed up the training, but not to counter overfitting (which in fact is not its major purpose). At this point we also started to reduce the batch size from 256 to 128, due to memory limits which lead us to `ResourceExhaustedError` even on Kaggle Notebook.
We tried to insert dropout layers with and without batch normalization, before and after it, both in convolutional layers and FC or only in FC, with dropout rates from 0.2 to 0.5.
We saw that reducing batch size up to 8 helped, but we didn't go over 78%.
We substituted dropout and BN with L_2 regularization in convolutional layers, trying various values of lambda, from 10^{-6} to 10^{-3}.
We also tried to reduce the complexity of the architecture, with the number of convolutional layers decreased to 7 or 6 and the initial number of filters with various intermediate values between 8 and 16, but we didn't see any relevant improvement.
At the end the best with this architecture (87%) has been achieved with L_2 only, lambda=10^{-3}, 512 neurons in the FC layer, 8 convolutional layers, starting from 16 filters, bs=8, training it with a manual decrease of learning rate from 10^{-6} up to 10^{-9}, for 10-15 epochs for each learning rate value.

Transfer Learning

We moved to transfer learning, with a model based on VGG16 [Notebook 2], fine-tuned from the 15th layer on. It is provided with a flatten layer, a dense layer with 512 units and a final softmax layer.
The model has been trained with batch size of 16, a starting learning rate of 10^{-7} and a validation split of 0.15 with L_2 regularization of 10^{-3}. After about 40 epochs the model was having increasing performance values and has been tested leading to a score on the test set of 0.9333. At this point the validation split has been further reduced to 0.1 to allow a better training on the network.
After about 20 epochs, the learning rate has been decreased in different steps from 10^{-7} up to the value of 10^{-9}. The training continued for around 10-15 epochs until an overfitting behaviour came out. A submission done during this phase scored 0.9422.
In the end the hold out set has been removed and the network has been retrained from scratch for a nearly equal number of epochs as those used to train the model with the hold out set.
This achieved the best performance on the test set reaching 0.9533.

We tried a similar approach with other pretrained models: ResNet50v2, EfficientNetB0 and Inceptionv3. None of them was able to reach the same performances of VGG, the first two arrived to 88-89%, the third one at 87%. With more time/computational resources it would have been possible to do more experiments with them, finding a better top part.
In the end we tried to use an ensemble of VGG, Resnet and EfficientNet [Notebook 3] with two different voting methods, but the best score was 0.92666, worse than the single VGG.