Introduction to Machine Learning project report

Eugenio Berretta
UNITN
eugenio.berretta@unitn.it

1. Introduction

This project's purpose was to correctly classify the types of galaxy that some image represented. The provided dataset consisted of 17736 images, where 5321 were unlabeled because used for test. The images were coloured and of size 256x256.

There were 10 possible types of galaxy, namely: *Barred Spiral*, *Cigar Shaped Smooth*, *Disturbed*, *Edge-on with Bulge*, *Edge-on without Bulge*, *In-between Round Smooth*, *Merging* , *Round Smooth*, *Unbarred Loose Spiral* and *Unbarred Tight Spiral*. The dataset was imbalanced, this means that in the dataset there were types of galaxy less common than others.

The results' accuracy was evaluated mainly *per-sample* (Acc) and *per-class* (mAcc). Sometimes, also the *mloU* has been used, even if it was not requested.

Being the test data unlabeled, 20% of the training dataset has been used for validation.

2. Proposed Method

Initially, shallow methods have been used. They were not the best candidates for this type of task, indeed, recognizing the type of a galaxy is difficult and learning algorithms should take in consideration both multiple features and different types of features. This is surely not the case of DTs, that work well when there are decision boundaries parallel to the axis. Neither SVMs were appropriate, in that they are just linear models.

In any case, being the dataset quite big (images' resolution was relatively high), in order to being able to train shallow methods in a reasonable amount of time and due to my pc's RAM limits, I had to decrease the images' resolution to 72x72.

The best solution was using neural networks, that were able to learn from a more complex set of features than the previous methods. While ad-hoc neural networks have been tried to be implemented, their performance was considerably worse than neural networks obtained from the fine-tuning of already pre-trained models. Transfer learning revealed to be the best way to obtain acceptable performance in terms of easiness of implementation and effectiveness.

In order to be able to train relatively quickly these neural networks, the CUDA technology has being exploited, using the GPU for matrices calculus. Being my GPU's memory not a so much, initially, I had to use very small batches. Being small batches a problem *[0, Shen, 2018]*, I decided to move to Google's *colab*, in order to have a good GPU and to increase the batches' size. I also found that by increasing the initial learning rate, also the accuracy of big-batched networks increased.

Of course the inputs to these networks have been normalized, furthermore, exploiting PyTorch's functionalities, some random transformations such cropping, rotations and flipping have been added to the *ImageFolder* class. This allowed to have an effect similar to the data augmentation without having to manually generating images.

A problem that has been faced is the learning rate; in particular, it was difficult to find a learning rate that behaved well during all the epochs. To tackle this issue, I used a scheduler, that changed the learning rate during the epochs. Among many available adaptive schedulers [1], I finally preferred choosing a fixed one [2], even if the adaptive ones act in base of the actual accuracy variation and not on a constant factor every a certain number of epochs. The reason was that the adaptive ones needed to many epochs to detect a "stall" and change the learning rate, so a good-chosen fixed one was more effective.

Another problem was the dataset unbalance. Some classes were significantly more represented than others and this entailed a good Acc but a bad mAcc. For this reason, instead of just shuffling the images given as input to the network, a sampler has been used. The sampler, by knowing the frequencies for each class, picked more inputs of less-represented classes, increasing the mAcc in particular.

Another technique that has been tried was training a neural network, removing the last layer and passing the extracted features to a shallow method such as SVM. This has not provided a great improvement, though.

3. Results

3.1. Decision Tree

The decision tree required a flat array for each sample. Being the images of my training set about 9000 and being each of them 256x256 (RGB), this meant an array with length

$$9000 * 256^2 * 3$$

In order to have an input that could fit in the RAM, the images' resolution was reduced to 72x72. The three dimension tensors (72x72 for the images, 3 for RGB) have been flatten by using a custom pytorch transform class that used a tensor view. The same approach has been applied to gray scale images, obtaining a result that was almost as good as the coloured one (36% against 37%) and with a very better training time (34s against 119s).

As expected, the result was not very accurate, guessing only around 40% of the validation set. The features, especially the most used ones, were all located in the center of the image [Fig 1 and 2].

Additionally, I tried to sum the quantity for each colour (RGB) of each sample and use it as input, obtaining an even worse result (approx. 20%).

3.2. Random forest

Exactly the same inputs were given to random forests, obtaining slightly better results, that arrived at maximum at a 56% accuracy. Until a certain threshold, increasing the number of trees entailed greater accuracy (and of course a greater training time) [Fig. 3].

3.3. KNN

In terms of computational time, even if the training time was quicker than the DTs, KNNs revealed to be very slower to be evaluated. Also here the result was not so good, obtaining around 40%.

3.4. SVM

SVM has been trained using the same inputs of the previous methods, required a huge amount of time to be trained and gave a result slightly better, 52%. In any case, this result was very lower compared to the one obtained by neural networks.

3.5. NN

In practise, NNs were the only solutions that gave good accuracy. Pre-trained models outperformed all my ad-hoc models and reached good performances in a small amount of epochs. Principally, resnet18 and resnet50 have been used. Higher-numbered models such as resnet101 and resnet154 have not been used, because they required to much GPU and, even with colab, could be used only with small batches.

Changing the SGD optimizer did not provide any improvement. In opposition, adding a scheduler for the learning rate ended up being effective and the same can be said for the balanced sampler. While a custom balancer could have been implemented [3], or even data samples could have been augmented [4], I decided to use Pytorch already-implemented WeightedSampler.

While, by using the balancer, the accuracy for the under-represented Cigar galaxy has been considerably improved, an issue that remained was the disturbed galaxy. Its accuracy was not high as the other ones and, in particular, the mAcc was only 40%. Alas, its frequency was nowhere near as low as the Cigar's one, so unbalancing could not be the issue. The reason was probably that it was quite an "irregular" galaxy. Indeed, *"Disturbed galaxies have a morphology that has been altered by an interaction with another galaxy. Disturbances can take many forms including; warps, tidal tails and asymmetries"* [5]. This probably means that the neural network captured only some features of disturbed galaxies, that however are not present in some others and, consequently, were not recognized.

This last problem is the reason why my networks performances were below 85% and a proper solution has not been found. A network that needed just to discriminate disturbed galaxies and non-disturbed galaxies had a performance (validation set) of 95%. However, I personally did not found the idea of having an ad-hoc network for this purpose an appropriate methodology.

As shown in [Fig 4], the Disturbed type is in conflict with the Barred Spiral. When the Barred Spiral grows up, the Disturbed becomes inaccurate. This is because one of the features of a Disturbed could be a bar. Other times, different features are taken in consideration for the Disturbed and the conflict is with other types of galaxies.

Neural networks have also been used to extract features that were passed to shallow methods. While these were enhanced compared to the shallow methods alone, the results were not as good as the resnets' ones.

References

[0, Shen, 2018] Kevin Shen, 2018, "Effect of batch size on training dynamics", https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

[1] https://www.deeplearningwizard.com/deep_learning/boosting_models_on-loss-plateau-decay

[2] https://www.kaggle.com/isbhargav/guide-to-pytorch-learning-rate-scheduling

[3] https://www.kaggle.com/shonenkov/class-balance-with-pytorch-xla

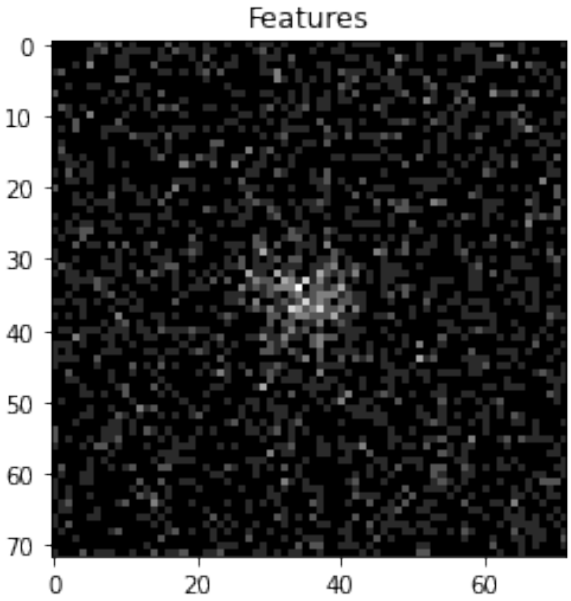

Figure 1. All features of DT (grayscale)

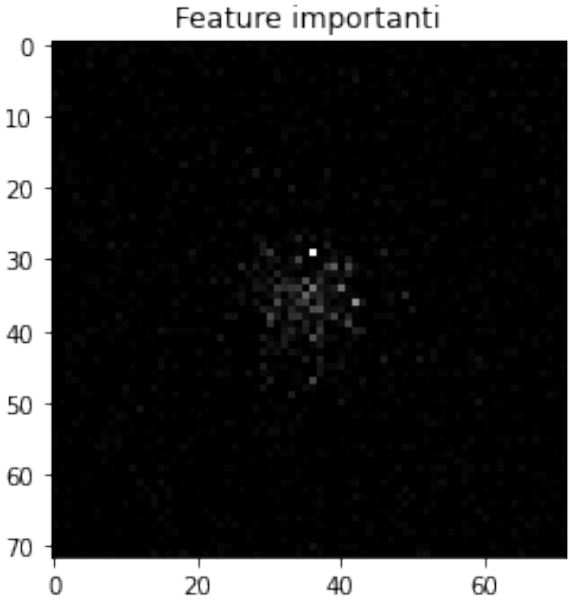

Figure 2. Important features of DT (grayscale)

[4] https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

[5] https://astronomy.swin.edu.au/cosmos/d/Disturbed+Galaxies

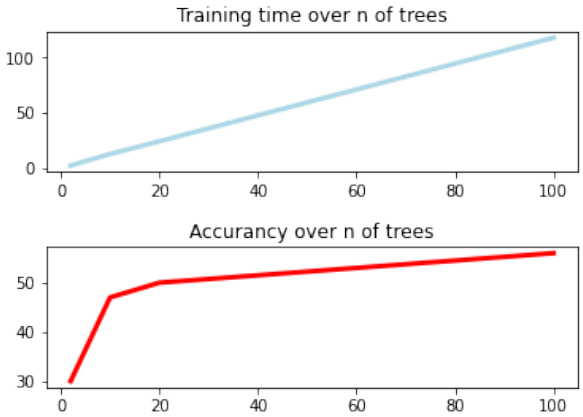

Figure 3. Random forests

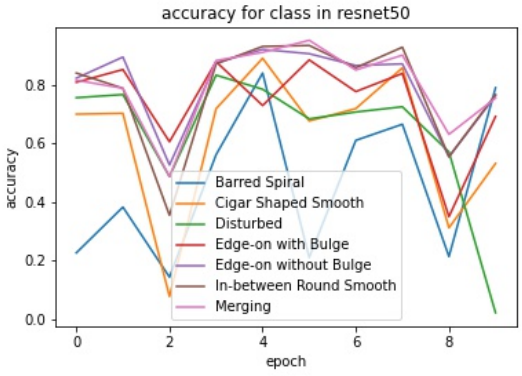

Figure 4. Accuracy for class