Introduction to Machine Learning project Report

Leonardo Vicentini
Università degli Studi di Trento
leonardo.vicentini@studenti.unitn.it

1. Introduction

The project's objective was to develop one or more algorithms to classify the morphological class of galaxies given an RGB image of those. The classes involved were: {*Barred Spiral*, *Cigar Shaped Smooth*, *Disturbed*, *Edge-on with Bulge*, *Edge-on without Bulge*, *In-between Round Smooth*, *Merging*, *Round Smooth*, *Unbarred Loose Spiral*}. In other words: it is a multiclass classification task.

The dataset provided was composed of 17736 images already splitted in train set and test set with a proportion of 70/30. We were advised to split furthermore the train set in order to obtain a validation set useful for testing design choices and tuning hyperparameters.

The dataset suffered class imbalance and this was taken into account for the validation split. This characteristic could also affect the different accuracy between classes, in addition to the fact that, by their nature (unique shape of the galaxy, unique colors, etc.), some classes are easier to learn than others. For every class group in training set, 20% of that portion was reserved for the validation set.

In total: 9928 for training set, 2487 for validation set, 5321 for train set.

2. Proposed Methods

I considered both deep methods and shallow methods to pursue the project's task. I noticed that fine tuning shallow methods is enormuosly faster compared to the methods that use a neural network alone.

2.1. Data augmentation

One thing to consider for every method is the data augmentation performed: I availed myself of the methods present in the class torch.transforms. In particular: *Resize*, *RandomHorizontalFlip*, *RandomVerticalFlip*. Considering the fact that images were already centered, I discarded *RandomCrop*, being afraid of losing important features unnecessarily.

While methods that intervene on colors such as *ColorJitter* worsened performances so they were removed from the tranformations.

A method that I introduced in a second time that made a valuable increase of the performance is *RandomRotation*.

3. Results

I started implementing algorithms based on a neural network alone to exploit these later and perform feature extraction for shallow methods. The best result on test set has been achieved using VGG19 as feature extractor and Support Vector Machines. Surely there is room to improve the performance of ResNet50 and VGG19 used alone as a single method.

3.1. Neural networks (Deep methods)

I tried to implement a neural network from scratch but I quickly changed direction to focus on pretrained networks. My choice fell on ResNet50 and VGG19.

There are some characteristics in common to point out. The first one is that after a certain amount of epoch, in my experiment usually 35/40, both ResNet50 and VGG19 tend to overfit, this means that the accuracy on the training set keep growing but the accuracy on validation remains stable. It is then necessary to change the learning rate in order to achieve a longer improvement during epochs. The solution that I adopted is using a very simple learning rate scheduler that decreased the *lr* by a factor of 10 after a certain number of epoch.

The loss function used for both neural networks was Cross-entropy loss which is well known as the standard for image classification and for multi-class classification problems in general.

Every epoch the model was saved if the loss on validation set was lower than the best until then. After the entire training cycle, the best model was loaded to perform the testing phase.

3.1.1 ResNet50

At the very beginning I struggled using this type of method because I could not break the threshold of 50% of sample wise accuraccy even after 40+ epochs. When I removed the part where I freezed some layers of the network the

accuracy started rising up. I first used this network without particular changes. Mini-batch sizes of the data loader were changed first, the sized used were 128, 64, 32 but this didn't lead to major performance improvement. The optimizer chosen was Adam with a starting learning rate of 0.001. In my experiments with this neural network (and as well with VGG19), after epoch 30/35 the performance are stable and without any notable improvement as the plateaus in 4 and 5 testify.

3.1.2 VGG19

VGG is a convolutional neural network developed by Visual Geometry Group (University of Oxford). There are other variants of VGG like VGG11, VGG16 but I started right away with VGG19 which is a deeper network than VGG16 but I didn't have any major timing problems, time required remained quite similar to using ResNet50. At the beginning I used the same optimizer as ResNet50 but the results were poor. I found that for VGG the advised optimizer was SDG so I introduced it with an initial learning rate of 0.01. The final results were slightly better than ResNet50.

3.2. Shallow methods

For these type of methods, it is necessary first to extract some features from the image. In order to perform this operation, I preferred the recommended way, so I used a pretrained convolutional neural network. The only change needed in order to use a neural network as a feature extractor is to substitute the last layer, the one used to classify in the 10 classes with an identity layer. In particular, referring to the 2 networks used previously, I changed the layer *fc* of the ResNet50 and the layer *classifier* for the VGG19.

3.2.1 VGG19 + K-nearest neighbors

I tried to improve the initial performance with the default number of neighbors of 5, tuning only the K neighbors hyperparameter with odd numbers. The best result on the validation set has been achieved with the values 9 and 11, with an Acc of 0.87, mAcc of 0.86 and mIoU of 0.76. After K = 11 the results are stable and not improving.

3.2.2 VGG19 + Decision Trees

Initial results were the lowest I discovered so I stepped quickly towards Random Forests without focusing much on tuning.

3.2.3 VGG19 + Random Forests

The hyperparameteres tuned in this case were the criterion (the function to measure the quality of a split) and the number of estimators (the number of trees). For the first one I tried both *entropy* (E) and *gini* (G) impurity measures. The parameter *max_features* that changes the number of features to consider when looking for the best split was left as default, so all features were taken into consideration. As we can see in Figure 2 the performances on every accuracy measure vary by 3% changing the hyperparameters.

3.2.4 VGG19 + Support Vector Machines

This method is the one that reached the best performances on the test set, with 85.6% on sample wise accuracy and 83.8% on class wise accuracy with the default settings. I tried to change the regularization parameter C, from the default 1.0 to 0.1. Another parameter that has been tuned is the kernel function with the values of: *linear* (L), *poly* (P), *rbf* (R)(default), *sigmoid* (S). As we can see in Figure 3, the best model is L0.1 with very slight improvements of 0.5%, 0.2% on Acc and mAcc from the default settings.

3.2.5 Analogous shallow methods using ResNet50

As we can see in Figure 1 the performances with this approach were worse in my experiments. Probably working more on the tuning of the ResNet50 could led to better results.

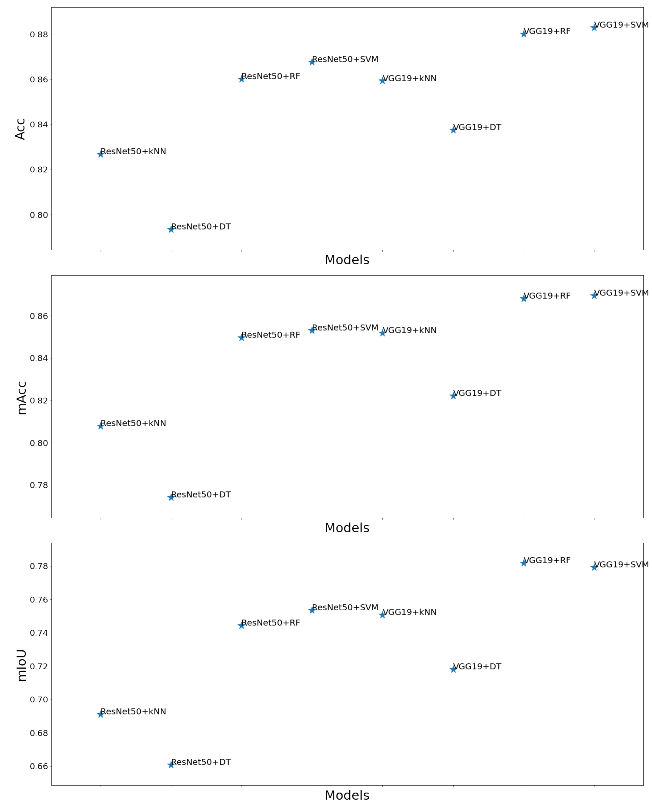

Figure 1: Performances of all shallow methods on validation set

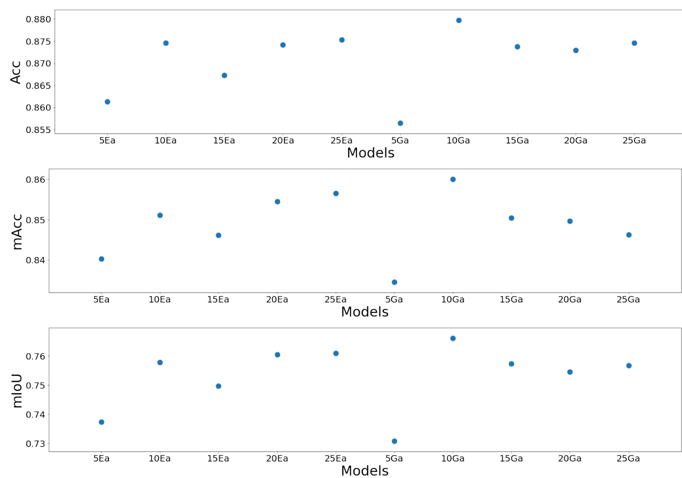

Figure 2: Performances of Random Forests on validation set changing hyperparameters

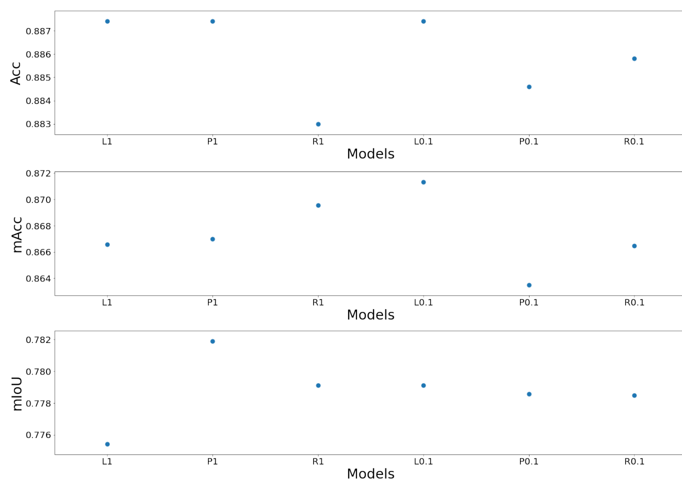

Figure 3: Performances of SVM on validation set changing hyperparameters

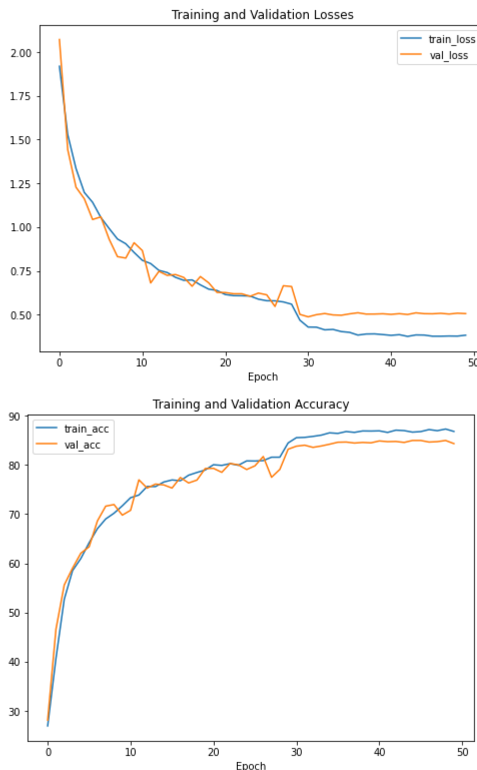

Figure 4: ResNet50: Loss and Sample-wise accuracy during 50 epoch

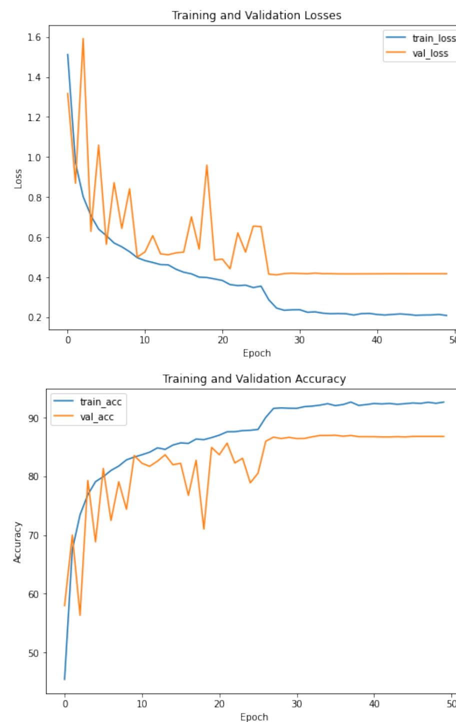

Figure 5: VGG19: Loss and Sample-wise accuracy during 50 epoch