Pretraining analysis

The purpose of the pretraining is to stimulate the networks using a smaller dataset in order to identify milestones and the best hyper-parameters to use for this project.

It is also a good practice to at least check if the project is feasible and in what conditions.

For this experiment I will be using a dataset of 15 images for training and 5 for validation per class (62 classes in total). The quantity is determined by the capacity of the GPU of my computer.

Once we have demonstrated that the project is feasible then it will be again executed using a framework provided by Amazon Web Services called SageMaker that can provide environments with more capabilities. The complete dataset is near to one hundred times larger than we need this kind of support.

According to the paper 4 architectures were used.

The architectures were created and adapted to the project.

The optimizer used for the training was Adam. I also tried to train using decent gradient, but the results were not good.

The models were initially loaded with imagenet weight as suggested by the paper.

Then there were just two hyper-parameters to find. The Learning rate and Reduce/Modify the Learning Rate during the training.

The experiment was executed with this pseudocode:

Initial step: Start training with LR = 0.00004 and without Reduce LR on Plateau enabled and train

1) Reduce LR in a factor of 0.00004

2) Train

3) Check train and validation dataset accuracy

4) If the result is better than the last execution jump to step 1).

5) If the result is not better than the las execution Increase LR in a factor of 0.00004 and jump to step 2).

One important behavior that was found is that using more than 7 classes reduced significantly the precision of the model, then we decided to work with groups of models of 6 classes or less taken alphabetically.

For this experiment in particular and because the HW limitation the epoch consisted in 10 images randomly selected by a generator that can flip vertical and horizontally the images plus a turn rage of 270°. Such features were selected because the paper suggest increasing the data set by eight by turning and flipping the images this means a lot of data, then we rather used a generator to handle the images.

Once we have obtained the best learning rate, we execute one last time enabling Reduce LR on Plateau to check if there is any improvement.

The results are the following:

\* All the architectures required different LR.

\* All the architectures reached at some point 100% precision of the training dataset, but anyone reached more than 80% precision in validation dataset.

\* RestNet had the lowest precision in validation dataset, DenseNet the highest and InseptionV3 and Xception had a similar behavior.

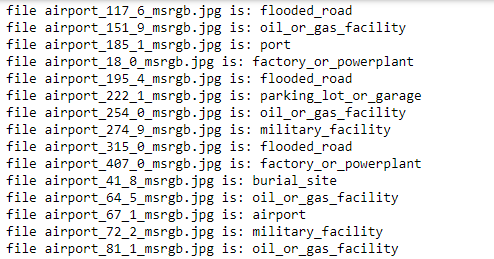
\* RestNet, InsetionV3 and Xception Models required Reduce LR on Plateau feature enabled.

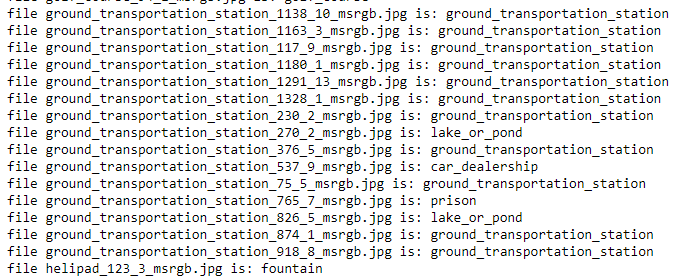
\* DenseNet doesn’t require Reduce LR on Plateau feature enabled, this feature even affects the behavior of DenseNet.

\* RestNet, InsetionV3 and Xception Models get the 100% precision on train dataset faster than DenseNet. The former ones need near to 80 epochs to reach the value while DenseNet needs near to 125 epochs.

The methodology that we are going to use is to separate the models that are going to be trained with less than 6 epochs. After trained to perform the evaluation the image will be evaluated by all the models and the results are going to be joined expecting that the correct classes will be the one with the highest value.

I noticed that in general the assumption works but there are cases were other classes different than the correct one also provides a high number sometimes slightly but bigger than the correct class and that produce errors. Then a solution that I am proposing is to regroup but not alphabetically but in similitude order.





Precision

Jaccar Index

According to the results we are going to continue working with the DenseNet and Xception Neural Networks.