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 with more capabilities in particular in GPU RAM memory that can be provided by Amazon Web Services called SageMaker or another provider. The complete dataset is near to two hundred times larger than we need this kind of support.

Training and definition of epochs

In the paper [?] it is denoted that the dataset is cropped and multiplied by eight by flipping the images horizontally and vertically and rotating the image 90°, 180° and 270° and they train the model with one epoch of this expanded dataset. In my case I will consider the flipping and rotating but I will generate minibatches of 10 images randomly selected and letting the generator randomly perform the flipping and rotation and every minibatch is used four times to train every epoch.

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.

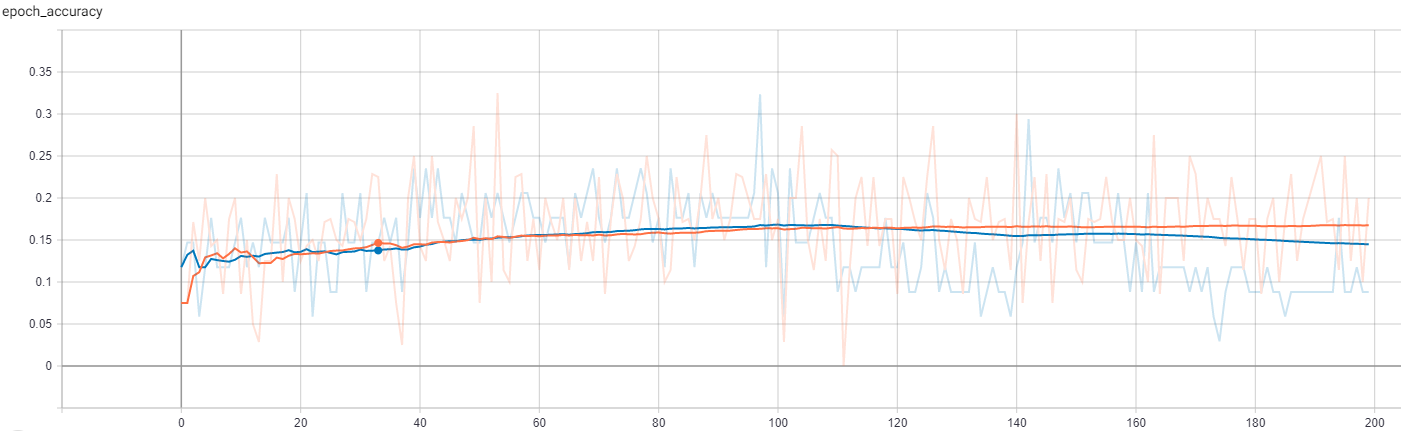
Then, there is just one hyper-parameters to find, it is the learning rate. At this point we are going to avoid other hyperparameters as dropout or modify the learning rate during the training.

The Learning Rate optimization was performed using Tensorboard and doing comparisons between the results.

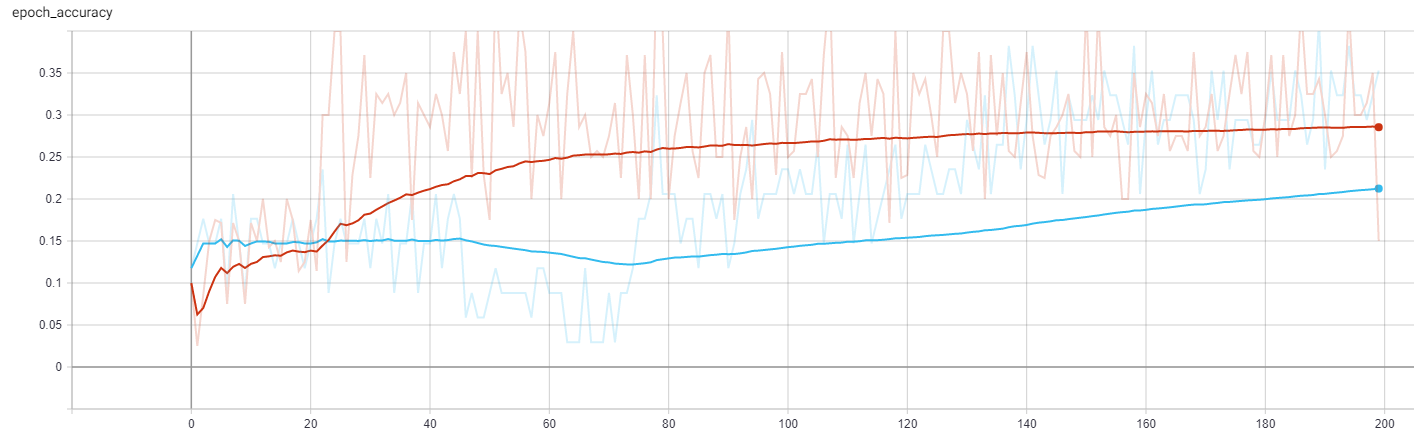
First a code was developed to train the models two hundred epochs starting from a learning rate of 0.5 and divide such learning rate by ten every twenty epochs to analyze the learning rate window from 0.5 to 5x10E-10. The result will be analyzed to check the window were the model shows learning to then focus the analysis in such area.

For DenseNet it was required to modify the learning rate every forty epochs because any visible change appeared using just 20 epochs.

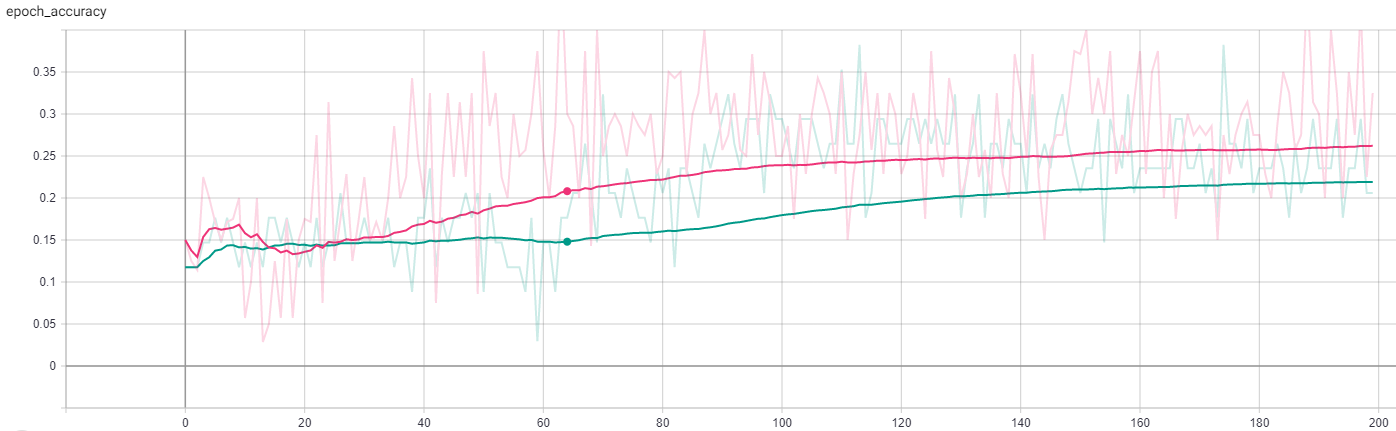
The way to analyze the graphs is making the as smooth as possible and to check were the increasing clear and not too noisy.



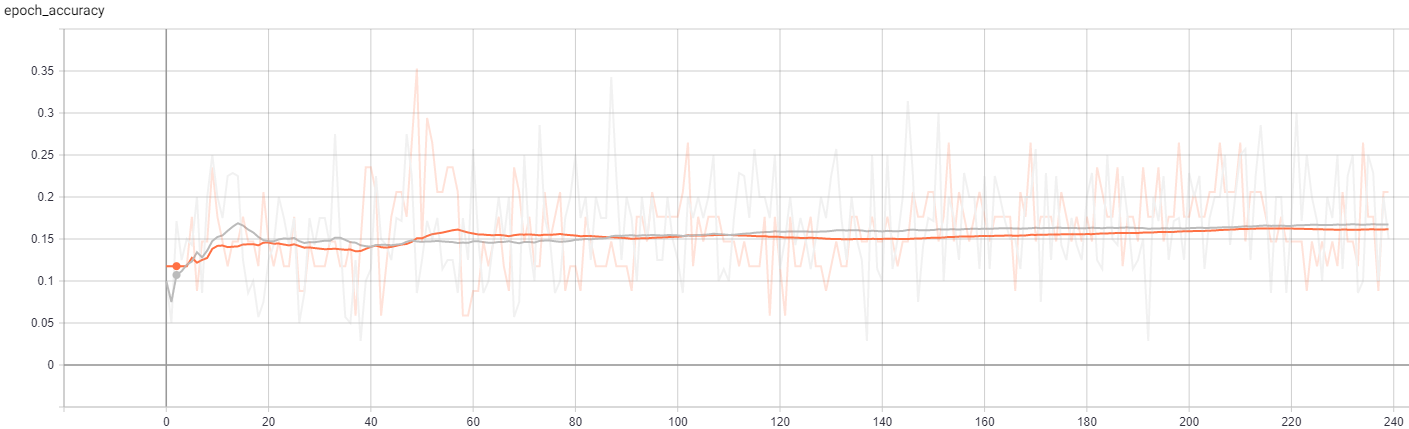
In this example that is the analysis for Resnet152, we can see that the window of improvement goes from the epoch 40 to the epoch 120 that represents the learning rate 0.005 to 0.000005.



For InceptionV3 goes from 0.05 to 0.000005.



For InceptionV3 goes from 0.005 to 0.000005.



For InceptionV3 goes from 0.005 to 0.00005.

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.

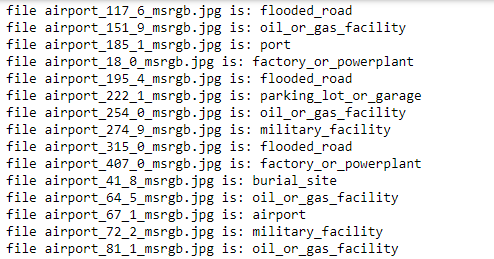
\* RestNet, InsetionV3 and Xception Models required Reduce LR on Plateau feature enabled.

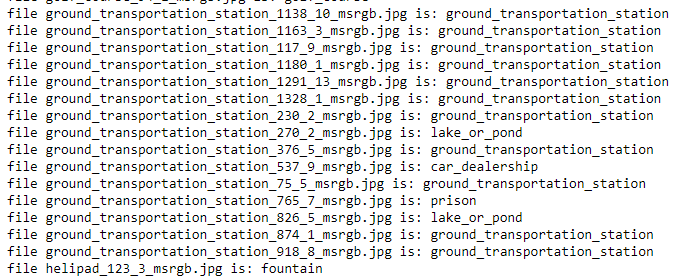
\* DenseNet doesn’t require Reduce LR on Plateau feature enabled, this feature even affects the behavior of DenseNet.

\* 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.

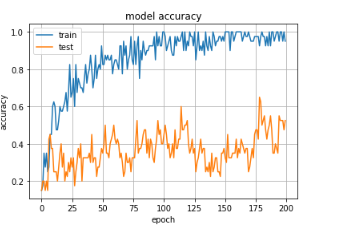




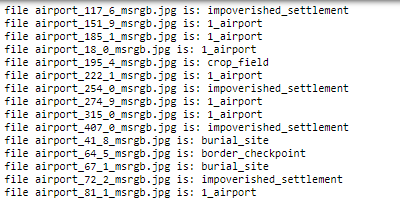
The methodology that I propose to improve this situation is to separate the groups according the similarities. In this case we can clearly see that airport, factory\_or\_powerplant, flooded\_road, military\_facility, oil\_or\_gas\_facility and port are reporting similar probabilities when trying to predict what an airport image is then instead of grouping alphabetically we are going to group that are reporting conflicts.

The first step is to create a model with this groups and check if the accuracy is not affected.





As we can see the accuracy remains then we are going to check again if this change improves the results.



We can notice that there was an improvement from 6% to 46% for in this example that is predicting airports that was one with the lowest prediction rates.

Precision

Jaccar Index

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