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.

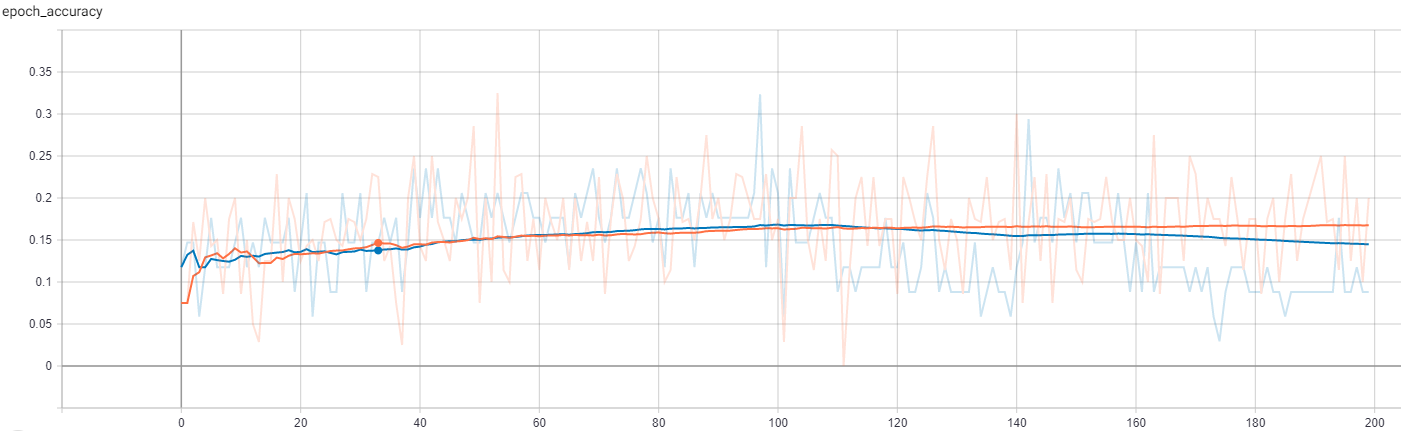
Then, there is just one hyper-parameters to find, it is the learning rate. At this point we are going to avoid other hyperparameters as dropout or modify the learning rate during the training.

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

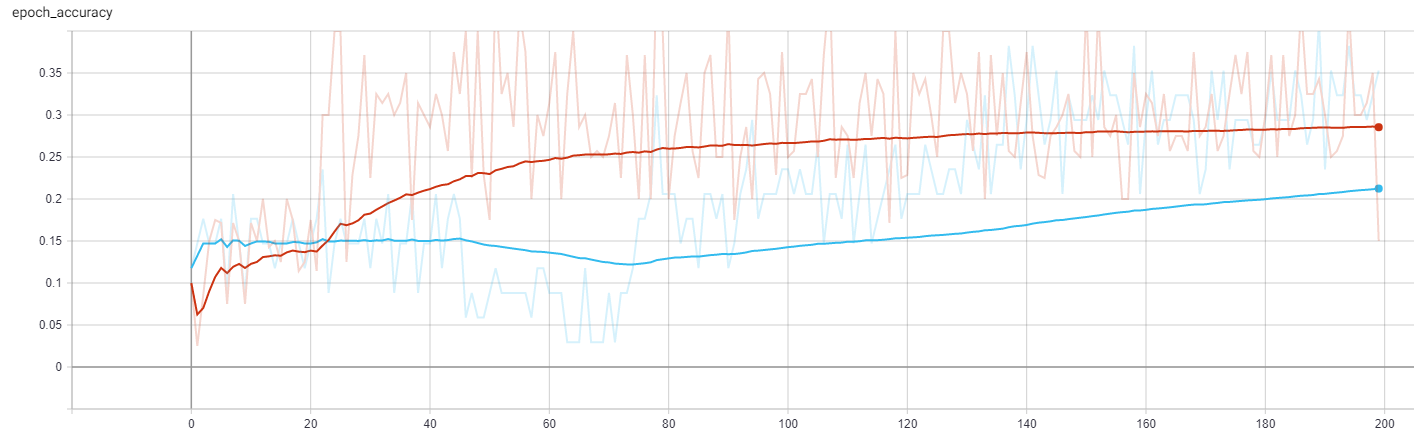
First a code was developed to train the models two hundred epochs starting from a learning rate of 0.5 and divide such learning rate by ten every twenty epochs to analyze the learning rate window from 0.5 to 5x10E-10. The result will be analyzed to check the window were the model shows learning to then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

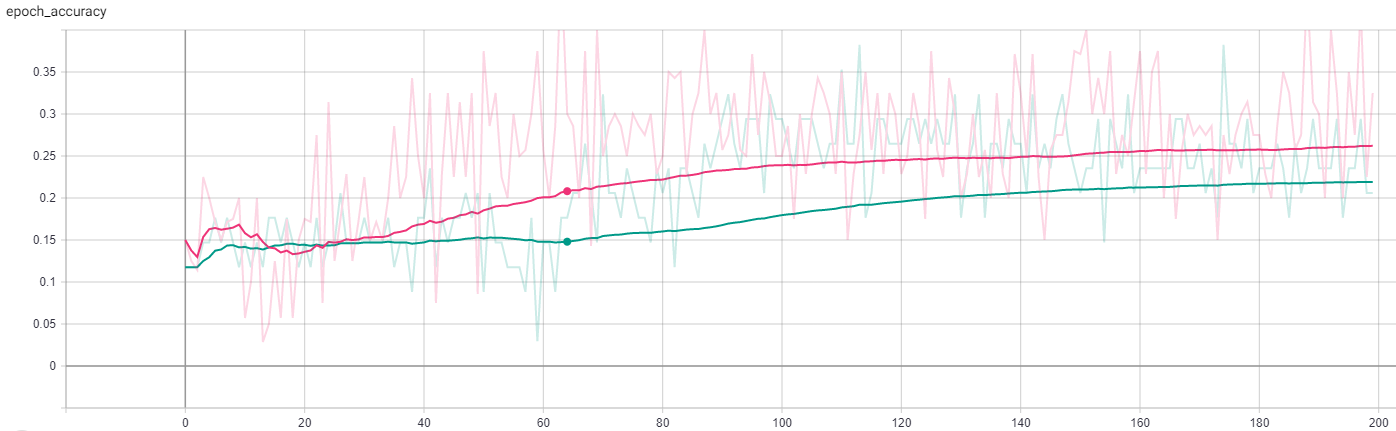
The way to analyze the graphs is making the as smooth as possible and to check were the increasing clear and not too noisy.



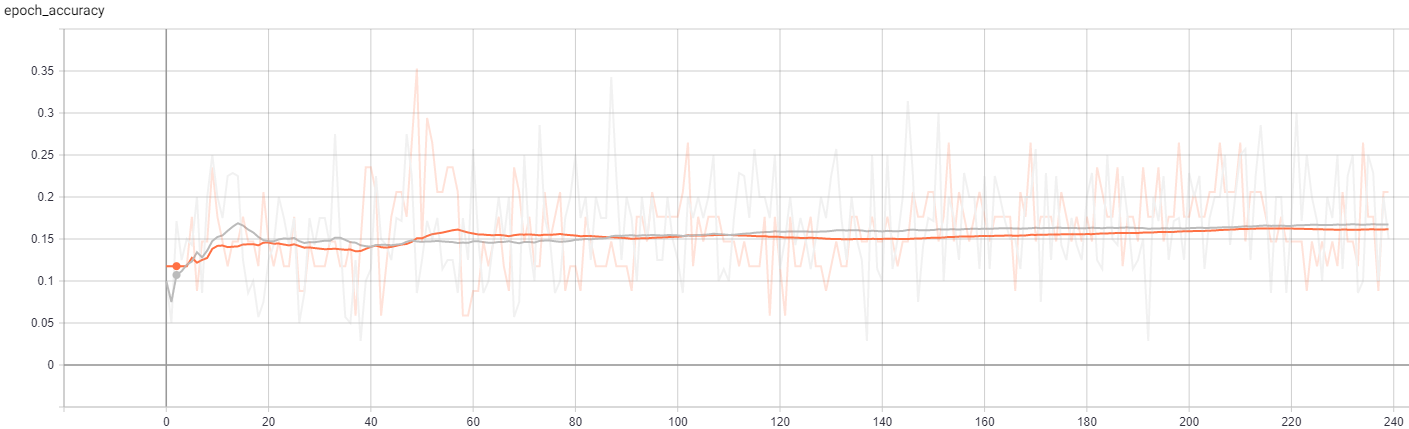
In this example that is the analysis for Resnet152, we can see that the window of improvement goes from the epoch 40 to the epoch 120 that represents the learning rate 0.005 to 0.000005.



For InceptionV3 goes from 0.05 to 0.000005.



For Xception goes from 0.005 to 0.000005.



For DenseNet161 goes from 0.005 to 0.00005.

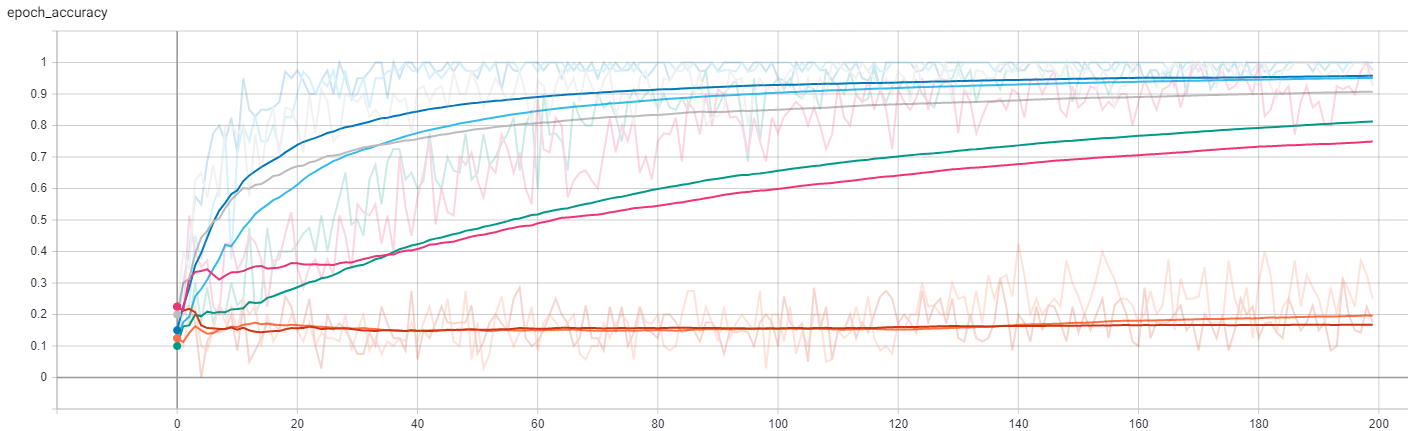
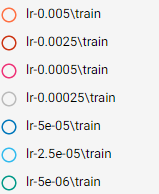
For this experiment I am considering to be able to execute near to 240 epochs in order to be able to have a wide window of analysis for the learning rate. In addition, we are considering for this experiment just to use seven classes to make the training shorter. After having the correct learning rate, the training will be executed using all the sixty two classes and using a more powerful framework to confirm if such learning rate keeps working using all the classes if it isn’t then the experiment has to be performed again from the beginning but using all the classes.

|  |  |
| --- | --- |
| Model | Range to analyze |
| Resnet152 | 0.005 to 0.000005 |
| InceptionV3 | 0.05 to 0.000005 |
| Xception | 0.005 to 0.000005 |
| DenseNet161 | 0.005 to 0.00005 |

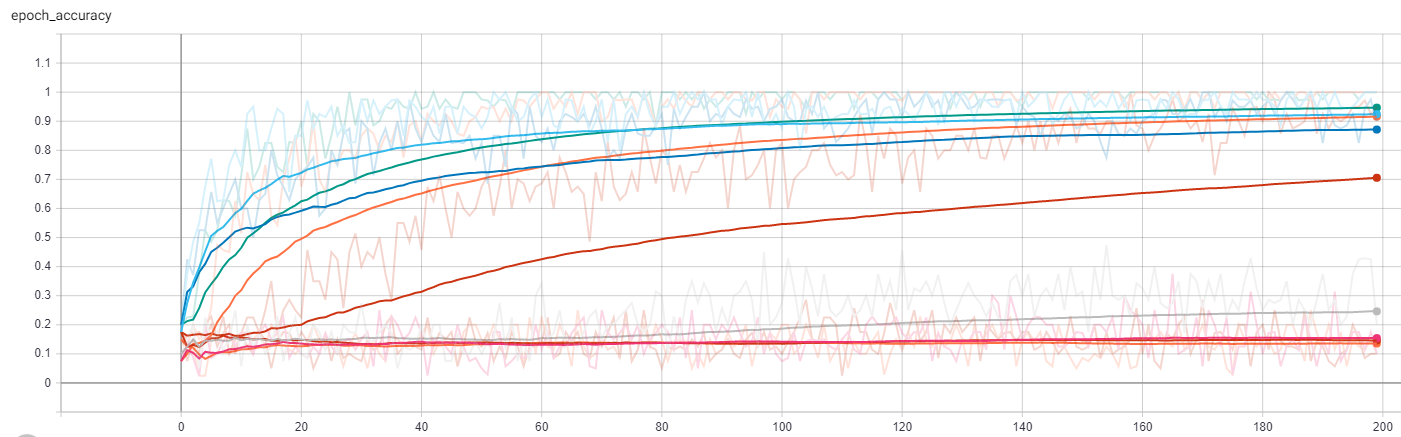
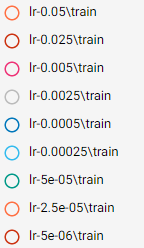
The results are the following

These graphs shows just the training accuracy because the assumption is that after using all the dataset the validation value will be close to the training value.

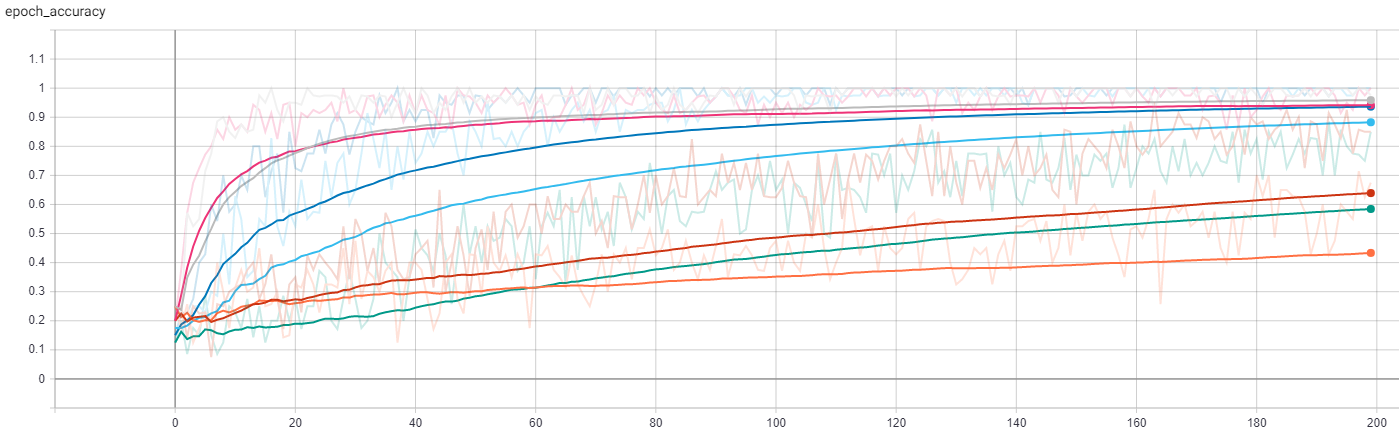
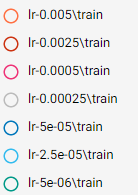
Resnet152



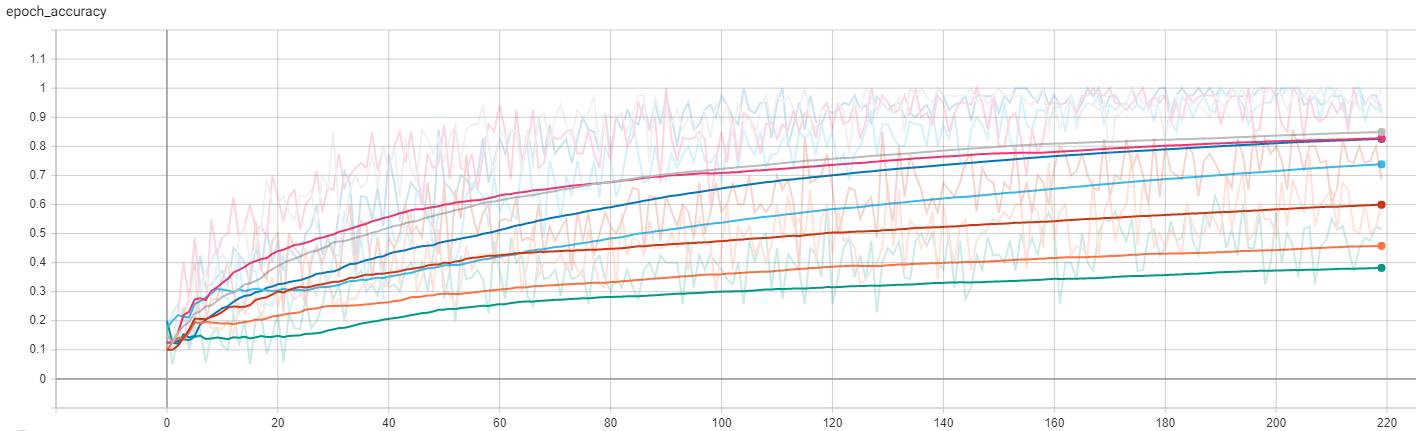
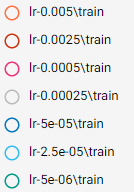
InceptionV3



Xception



Densenet161



|  |  |
| --- | --- |
| Model | Best LR |
| Resnet152 | 0.00005 |
| InceptionV3 | 0.00005 |
| Xception | 0.00025 |
| DenseNet161 | 0.00025 |

Accuracy

Metrics

Resnet152

Train

F1 Score: 0.9934928041549645

Hamming Loss: 0.006472491909385114

Jaccard Score: 0.9875417909099123

log loss: 0.024156280111023325

Validation

F1 Score: 0.20485367075515398

Hamming Loss: 0.7868852459016393

Jaccard Score: 0.13806384520081713

log loss: 5.692915327122016

InceptionV3

Train

F1 Score: 0.8879849109216635

Hamming Loss: 0.11542610571736785

Jaccard Score: 0.8091911196298816

log loss: 0.39418005801246914

Validation

F1 Score: 0.1373008546820092

Hamming Loss: 0.8557377049180328

Jaccard Score: 0.08992359250592097

log loss: 7.058589225571957

Xception

Train

F1 Score: 0.8187631969460836

Hamming Loss: 0.18015102481121897

Jaccard Score: 0.7065833162696866

log loss: 0.6532672608083165

Validation

F1 Score: 0.14215408247666314

Hamming Loss: 0.8491803278688524

Jaccard Score: 0.09127099006131263

log loss: 6.8693384256494685

DenseNet

Train

F1 Score: 0.8827789013863031

Hamming Loss: 0.1186623516720604

Jaccard Score: 0.7984909354221321

log loss: 0.4880079954175008

Validation

F1 Score: 0.08976616145269721

Hamming Loss: 0.898360655737705

Jaccard Score: 0.05340519695358405

log loss: 11.916597437710259