Deep Galaxies

The project was developed by Marco Di Francesco on 2 different platforms, initially on Kaggle then moved to GitHub, in the report there are the different versions of the tests linked.
Final GitHub repository (https://github.com/MarcoDiFrancesco/DeepGalaxies)

First steps

The project was built with the goal of understanding each piece of the process of a deep learning algorithm, so everything started really simple and then complexity was added step by step in order to understand what really made the difference in performance.
Everything started building a dataset with the 32x32 pixels in the middle of each picture, creating a really simple network consisting of 2 operations, each one containing a convolution and a max pooling operation.
This model ran with 1000 epochs and got around 62.4% of accuracy (notebook). [1]

Network

The network was then rebuilt taking inspiration from VGG, using its main structure and adapted it to the given dataset and later added a few improvements. More in detail it was built in such a way that the outputs were the 10 float values normalized from 0 to 1, equivalent to the likelihood of each image belonging to a galaxy type.
In detail the first part of the network [2] is built by 4 sequential operation, each containing: a convolution, with kernel 3x3, stride 1, padding 1, so that it's possible to exactly halve the dimension of the image; the batch normalization, used to normalize the inputs of the layer by re-centering them, to get a faster training. After that there are 2 fully connected layer [3] with RELU that take the input tensor of 512 * 8 * 8 down to 4086, and then down to 10 channels, where we get the final predictions. Moreover, the fully connected layers had random dropout with 50% chance, meaning that around half of the neuron (feature detectors) were set to 0, allowing to reduce overfitting.

Parameters and metrics

Cross Entropy Loss was chosen as the loss function, it uses the log soft max allowing to get the 10 values and normalize them from 0 to 1, and having an exponential nature so that it's possible to have a heavy penalty for being more wrong.
As optimizer SGD was used so that it was possible to slow down the training when we are getting to the local minima. Later weight decay was added to avoid overfitting, in order to penalize large weights and effectively limited the freedom of the model.
The parameters of the models were then tweaked to see the initial performance. The first training was of 100 epochs with a 10^-2 learning rate, getting the best accuracy of 63.3% but got 2 things that were looking wrong. First thing was the swing of the accuracy, between the epochs there was a huge gap (going for example from 63% to 55% back to 61%), so the learning rate was lowered. Another problem was getting 99% accuracy in training [4], so an L2 regularization was added to the optimizer (notebook).

Dataset and data augmentation

Dataset runs were not based on the same dataset, initially it was separated in 3 sets: train, validation and test, with the latter used to check for an unbiased evaluation of the model fit. Later the ratio of the dataset splits was changed because the validation-test accuracy in the first part of each run were really similar (<1% of difference), opposed to the training-validation sets that were clearly different (>25% of gap), so the ratio was updates from 70 / 20 / 10 to 80 / 20 to avoid wasting images. This

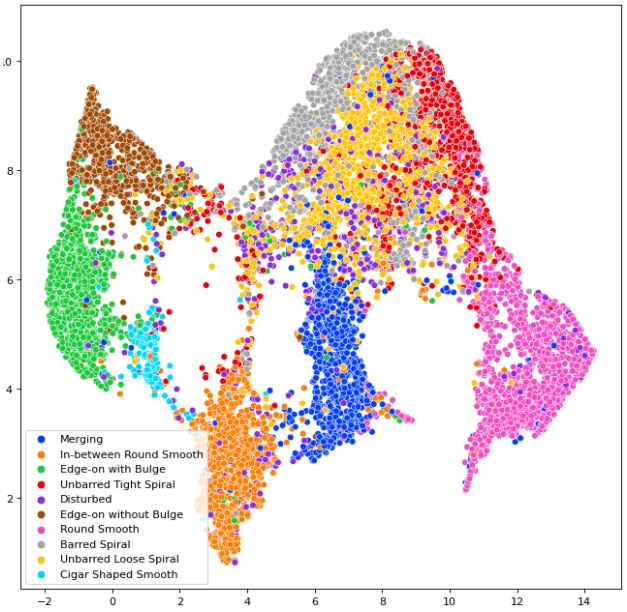
was also confirmed to be effective by the really similar values obtained by the validation set compared to the official unlabeled test set of the last training, having an accuracy difference <0.01%. Data augmentation tools were applied in the dataset and compared the performance improvements. To get consistent results the network was trained over 50 epochs looking at validation accuracy and more importantly overfitting, the former judged by the validation loss. All data augmentation techniques were only applied only to the training dataset, keeping the test dataset untouched.
The bare network scored 58% of accuracy on the validation set and 73% on the training set [5]. The pictures were then visualized to see if there were channels without useful information, but all channels contained some important information. (notebook) [6]
The first augmentation applied was random rotation, where the image was rotated by a random angle.It was tested and in this run the loss decreased way slower, achieving a 52% of accuracy in validation, and improving a lot overfitting, getting training accuracy down to 57%. [7]
Jitter was another technique tested where brightness, saturation, and other properties of an image were randomly changed (transform representation). It was able to achieve a 49% accuracy in validation and 51% accuracy in training, and because this difference did not look meaningful other than slowing down training it was then compared with the rotation one in a 170 epochs run. Because comparing them in 2 different charts was difficult they were plotted in the same one, and instead of using the accuracy loss was used. Here it is possible to see that not only the model did not improve, but the loss was even higher and took more time to get to it's lower point, it was not used in order to avoid using it in the final train. [8]
The Normalize function was tested in another run. It normalizes the values of each channel of the image (RGB) using a normal distribution. In the end, despite expecting both training accuracy decreasing and validation accuracy increasing only the latter happened [9], but given the improvement it was used in the final training.
The final training of the custom network was done using all the improvements discussed above. The best accuracy in validation was 77.5%. [10]
The custom model was then replaced by the torchvision pretrained versions of Resnet 50 and Resnet 152 using the same parameters. Both networks got better results, the Resnet152 achieved 79.5% in accuracy [11] and Resnet50 85.0% [12]. This improvement is probably due to the skip connection in resnet, adding them in the custom network would have allowed make it way deeper while avoiding the gradient descent problem.

Dimensionality reduction

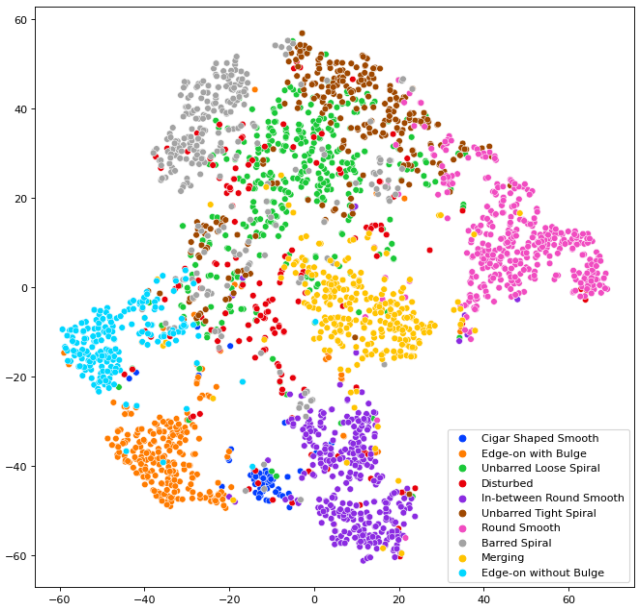
Dimensionality reduction techniques were used to visualize the image correlations. Weights of the network were saved every 5 epochs and chose the best weights, looking not only at the best accuracy but also seeing that the saved weights were not overfitted. The features were extracted from both the custom and the resnet network, and analyzed using 3 dimensionality reduction methods: UMAP, TSNE and PCA in both training and validation for both the networks. UMAP and TSNE had the most defined cohorts and were the ones used in the analysis, all of them can be seen in the GitBub notebooks (CustomNet) (Resnet). Analyzing the Resnet UMAP train [13] and TSNE validation [14] it's possible to notice patterns in the different galaxies, "Disturbed" was one of the most confused by almost all the other galaxies, another one is "Unbarred Loose Spiral", being confused in "Barred Spiral" and "Unbarred Tight Spiral". Analyzing the custom network UMAP and TSNE validation [15] [16], it was possible to notice some differences from the resnet distribution, the "Disturbed" class was not as confused, but it confused "Cigar Shaped Smooth" and "Merging" with all the other galaxies. An improvement to the network can be to change the ratio of images given in input to the network and increase the amount of the 4 galaxy's images it currently has difficulties to recognize.

Dimensionality reduction

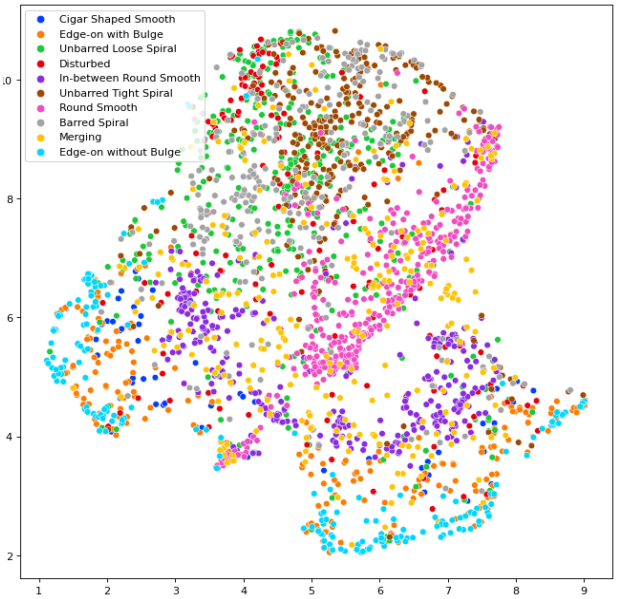
UMAP RESNET TRAIN



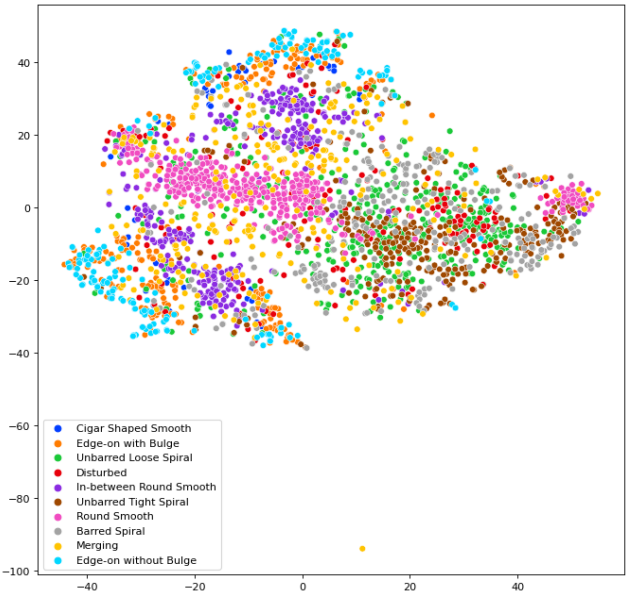
TSNE RESNET VALIDATION



UMAP CUSTOM NET VALIDATION

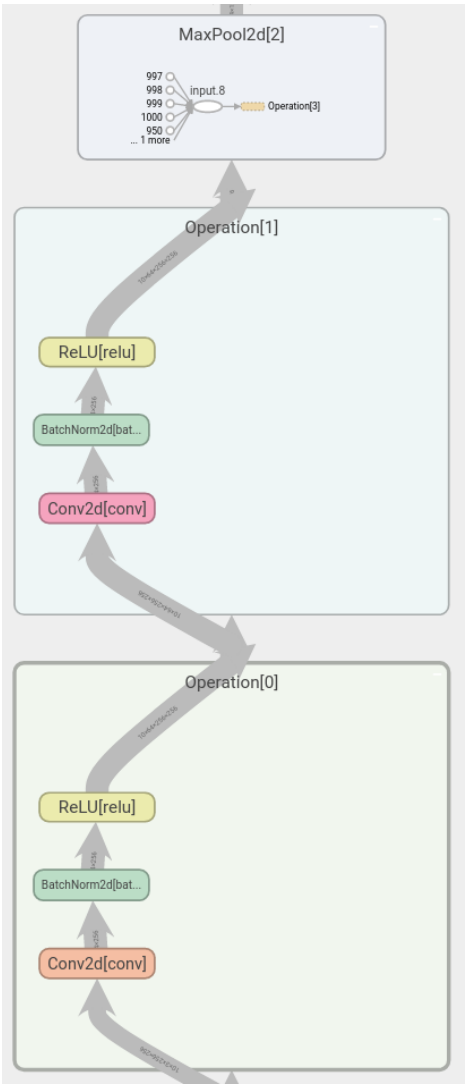


TSNE CUSTOM NET VALIDATION

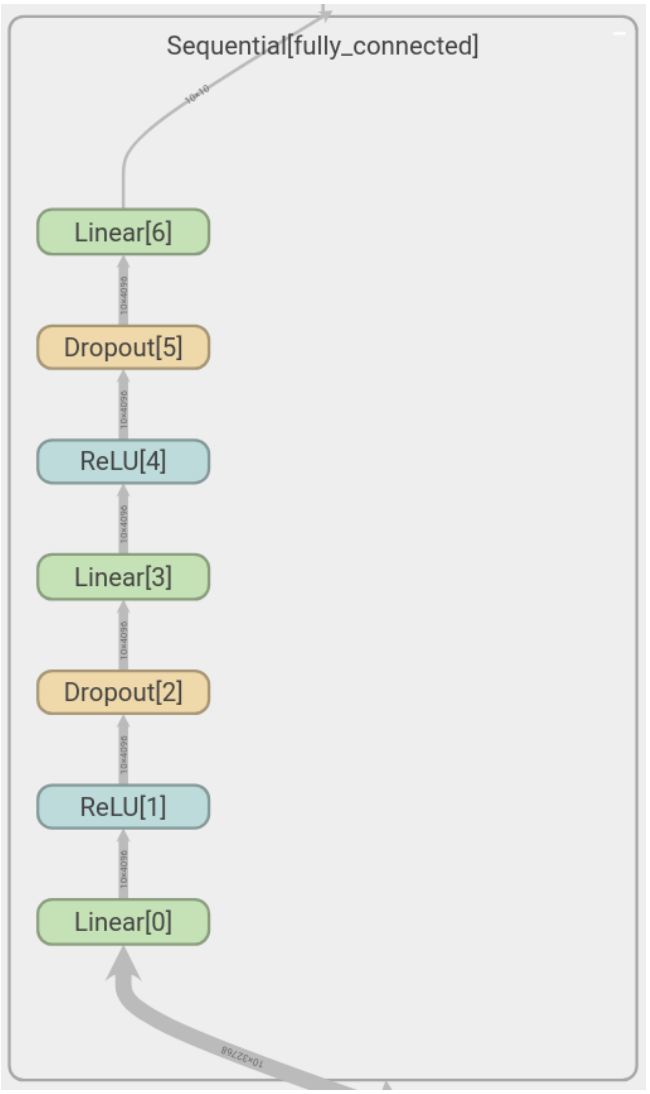


Network structure

OPERATION IMAGE

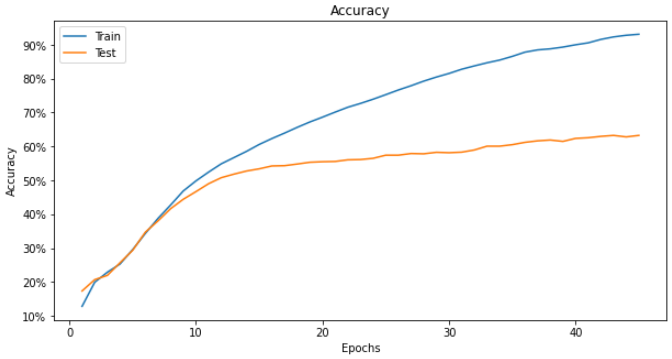


FULLY CONNECTED

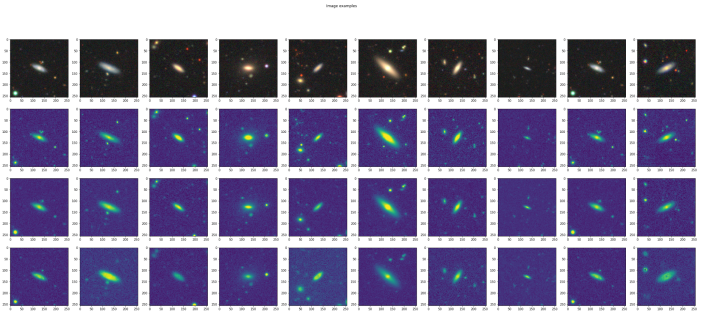


Plots

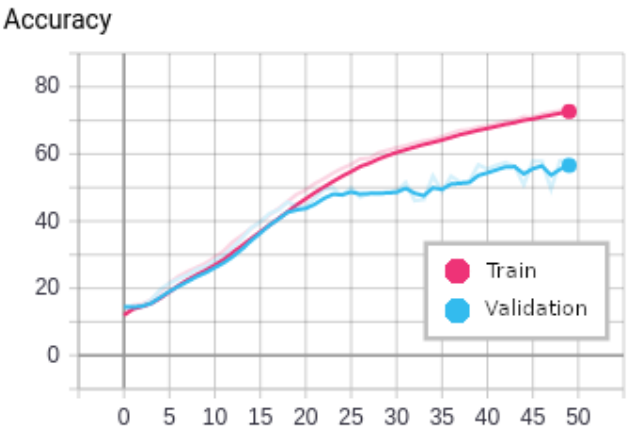
SMALL NET RUN

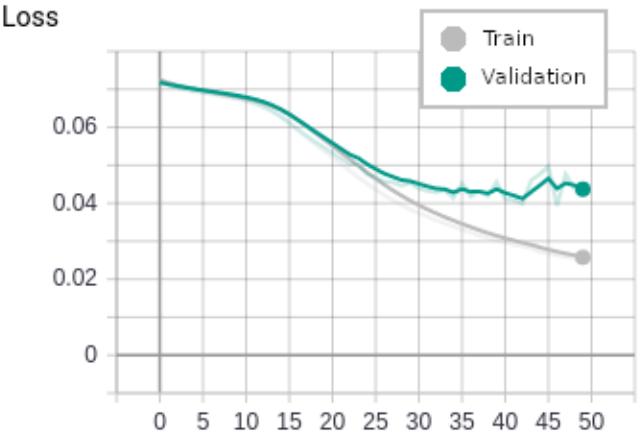


3 CHANNELS IMAGE



BARE NETWORK





RANDOM ROTATION

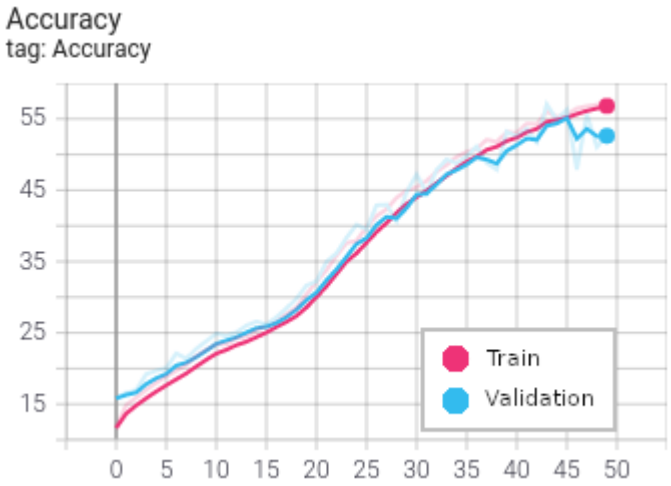
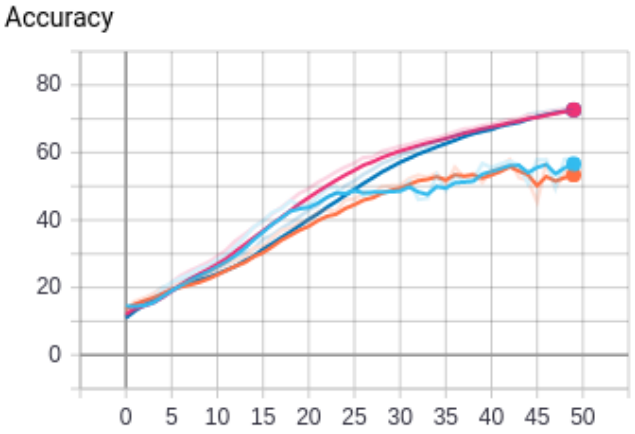
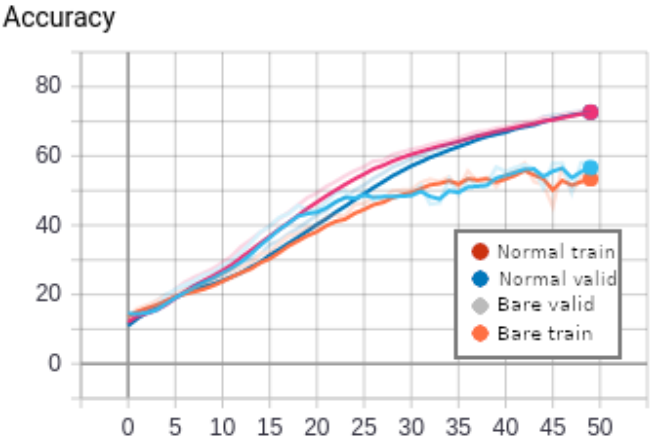
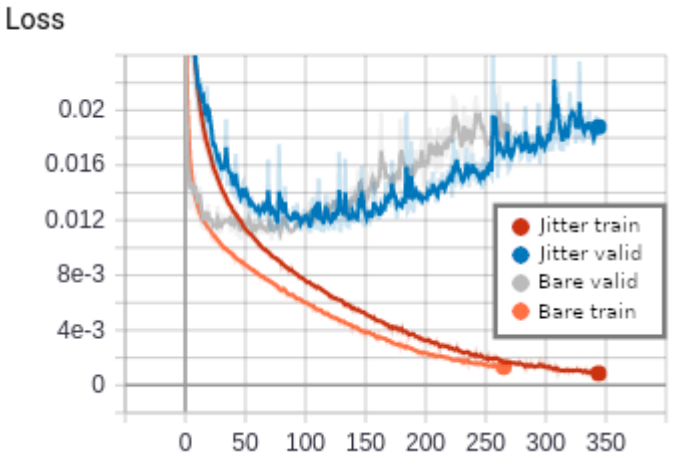


CHART NORMALIZATION VS BARE

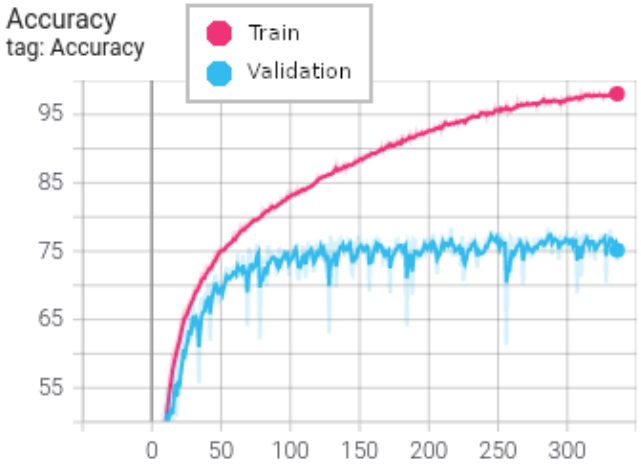


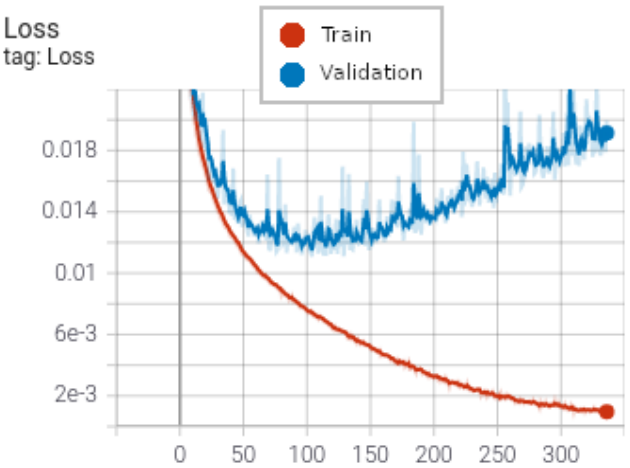


JITTER

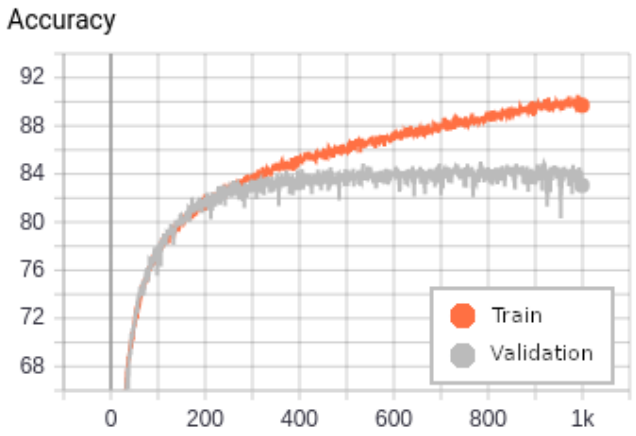


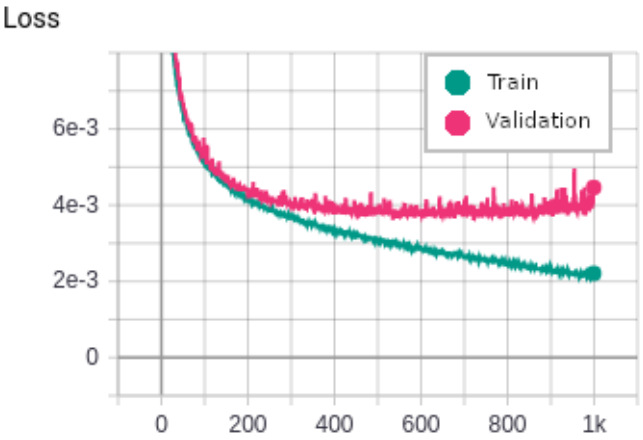
FINAL TRAINING





RESNET 50





RESNET 152