Classify subcellular protein patterns in human cells

https://github.com/mohelhibouri

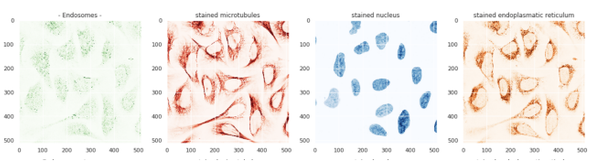
Mohamed El Hhibouri
UNI: me2641
m.elhibouri@columbia.edu

1. Introduction

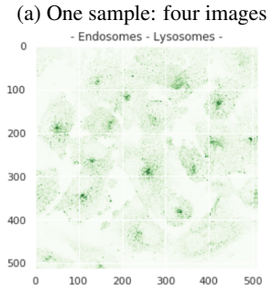
The problem at hand is the classification of proteins into their organelle localization labels. I will be differentiating between 28 different cell types and the input data will be microscopic images of the proteins of interest. The overall purpose of the project is for the submission into the Human Protein Atlas Image Classification Kaggle. Models submitted will contribute to The Human Protein Atlas; a smart microscopy system of identifying a proteins location from a high-throughput image. The goal of the project is to improve upon the baseline all classes implementation for the current protein classification given on the Kaggle competition website. [1]

2. Dataset and Problem Formulation

There are 31,072 unique entries in the data set. The entries are labeled with ids that represent their targeted labels for the given image. There are 28 labels with varying frequencies; with the most frequent being Nucleoplasm, being present in 12885 of the entries. All image samples are represented by four filters (stored as individual files), the protein of interest (green) plus three cellular landmarks: nucleus (blue), microtubules (red), endoplasmic reticulum (yellow). There are huge discrepancies between the most frequent label occurrences and the least frequent label occurrences. With the top three highest frequency label occurring as many times as the rest of the data combined. For this reason, I intend to use statistical sampling methods to carry out research on 10% of the dataset at a time. Co labeling occurs, but roughly half of the data consists of a single label for the image, and the large majority of the rest just have two labels for a given image. There does seem to be a slight correlation amongst some subcellular components, such as the endosomes and lysosomes; and this doesn?t come as much surprise as the two look very similar.



(a) One sample: four images



(b) Used to predict the label

Due to the high resolution of the images, I had to compress it to be able to feed it into our networks and run at a reasonable time. I also had to downsample it in order to train our hyperparameters sufficiently. My innovation, in dealing with compressed sampled versions of our data was to use data augmentation methods to enhance the usability of the data that I had. Upon developing a satisfactory model, I then transferred the same weights and parameters to the full dataset to achieve the same results.

2.1. Evaluation metrics

The primary evaluation metric of the Kaggle competition is the macro F1 score. As stated earlier, my first milestone is achieving a score of at least 0.5 on my model. The best loss function would be, of course the metric itself. However the macro is non-differentiable. Thus instead of using binary prediction, I will use probabilities in order to have a continuous loss function. Then if I have a probability p concerning 1 label, I'll get p true positive and 1-p false negative. If I have a probability p concerning 0 label, I will get p true negative and 1-p false positive. Also, a good trick could be to modify this loss function by rounding the value of probability and using it as an activation function for the last layer as proposed in comment in [4].

3. Features augmentation and data augmentation

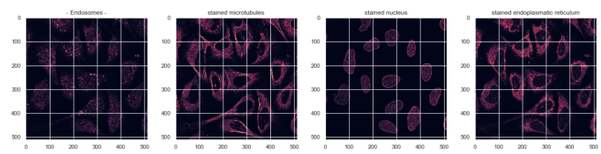
I carried out the below pre-processes using multiple workers running in parallel.

3.1. Data Augmentation

The pipeline I created read our images and transformed them randomly by one of the following methods: rotation of the images with 4 angles, two flips (up and down), cropping, Gaussian noises, Gaussian blur, contrast normalization and multiplication by a small coefficient all pixels.

3.2. Normed gradient using Sobel kernel

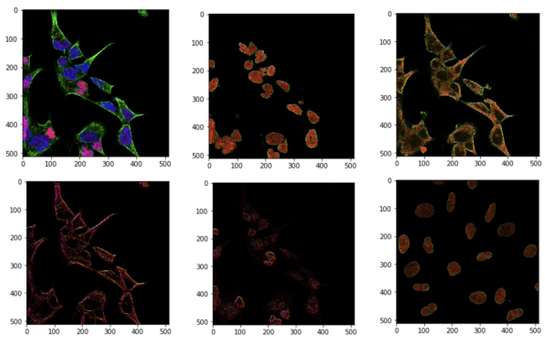
By convolving with a Sobel kernel my images, I was able to detect edges on the 4 different type of images as the following pictures. Furthermore, by comparing different types of proteins present in images, I noticed that some particular shapes are clearly visible and allow us to differentiate labels by eyes.



Edges detection using Sobel kernels

3.3. Directional gradient

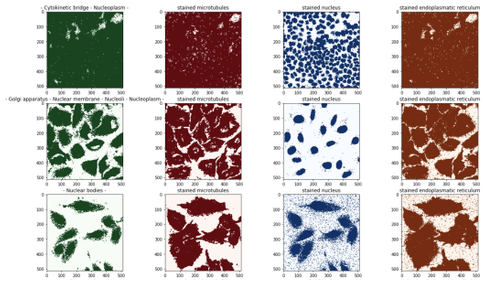
As each types of cells have different shape, I discovered that I can extract a great deal of information by using different gradient orientation. Indeed, this gives information about edges in multiple direction and then differentiated cells. Also, keeping pixels values only above 0 cleans the pictures and keeps only the edges of cells. In order to keep a repeatable pipeline, I concatenated those features augmentation as new channels: from (512,512, 4) to (512,512, 92). By plotting them in RBG, visually it is easy to see different combinations of channels and how different they are.



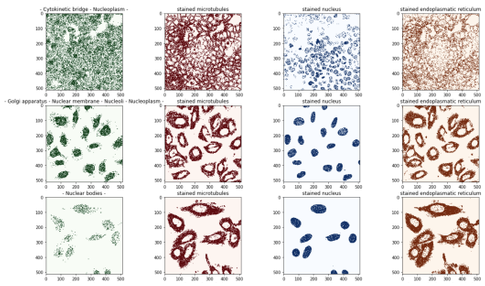
Multiple RBG channels combinaison

3.4. Normed gradient using Sobel kernel and threshold

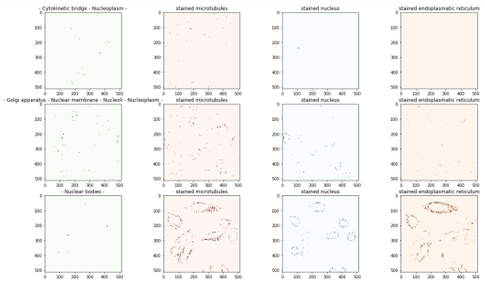
In order to detect those different shapes, I then add a lower bound threshold to detect if some objects could have outstanding pixels values. Indeed, for some particular labels like 'Endoplasmic reticulum', 'Nuclear bodies', and 'Nucleoli', I can observe on nucleus images that there are some particular colored pixels with different density repartition that are not visible for other labels. This is illustrated on the following pictures:



(a) Edges detection with threshold: pixels value above 0



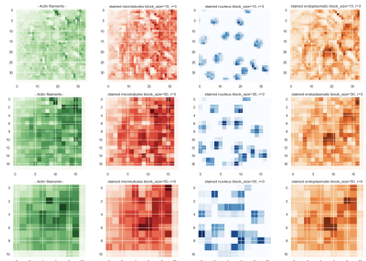
(b) Edges detection with threshold: pixels value above 10



(c) Edges detection with threshold: pixels value above 90

3.5. Blocks

Using blocks with different size and orientation of gradient, I was able to reduce the size of images but also keeping information about images. I sought to explore this type of preprocessing and find out if this type of data can keep enough information to have similar macro F1 score. I eventually did not keep this methodology as part of our image processing pipeline.



Different size pixels blocks of the mean of the gradient in multiple direction

4. U-net

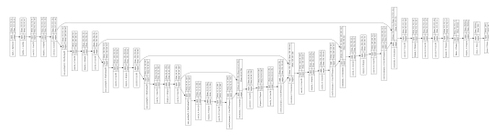
In researching work related to image classification, I came across the popular U-Net Architecture. I then implemented a U-net architecture on a sample set of the data. This was based on a recommendation from our project proposal and is derived from the paper by Ronneberger et al [4]. The advantage of this model is that it is able to identify localized labels within segments of a single image and work with smaller overall datasets. One difference in our implementation is that while the paper relies heavily on data augmentation to supplement its sparse amount of data, I was able to carry out a robust network with a sampling of our training dataset.

The basic architecture of our u-net consists of the below network of CNN layers, as well as a Flattening and Dense layer before the output.

We used an input shape of 512,512,4, preserving the initial 4 channels as well as the original size of the images,

Our U-net yielded less favorable results during the first few epochs compared to other models we had tried.. We did like the speed of the model and its ability to retain information over the layers.

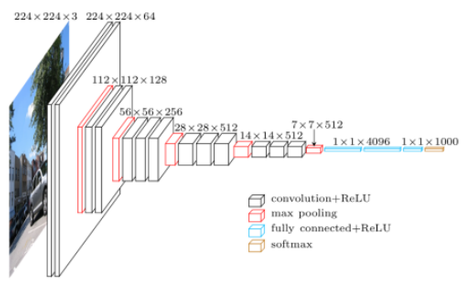
The size of our dataset and time constraints meant that I was not able to train the U-net on a fully augmented dataset.Instead, I randomly augmented images within our dataset as I trained our model. This still led to a bumper improvement in our predictions.



U-net architecture

5. VGG16

The third model in our three pronged approach was the VGG16 model, which was developed by the Visual Geometry Group at Oxford University. This particular implementation is the VGG16 used in an image recognition paper by Simonyan et al [9]. The model increases the representation depth of an image by putting it through successive CNNs on smaller and smaller sections of the image. The effect of this is to improve overall classification accuracy. One downside of this is that the increased depth leads to increases in complexity, which leads to much increased run times across the board. Below is a sample image from the paper I sourced for implementation[10]. Our image sizes vary and more detail can be found in the appendix.



VGG architecture

6. Spoon-net

Finally, I derived the Spoon Net as a combination of the Unet and the Depth enhancing VGG Models. I did this because I believed that I could improve the performance of the original Unet by adding more depth to the final concatenated images. The architecture of the model is shown below:

Spoon-net architecture

This model gave better outcomes than the U-net or the VGG.

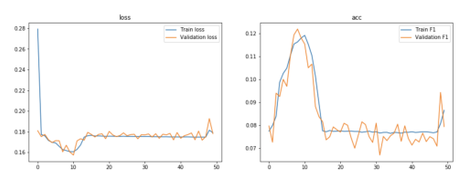
7. Results

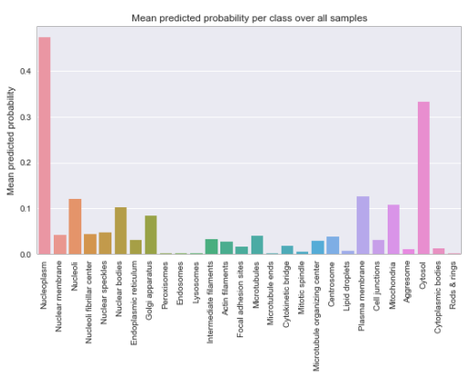
The Three Parallel Models achieved varying results over time. Due to the large dataset sizes, I ran only 10 epochs per model on our most recent dataset. In comparison, most entries on the leaderboards were based on runs of over 100 epochs. It is our hope that our final submission will include many more epochs and thus prove the efficacy of the models that I have put in place. Performance against validation set: The best performer against our validation set was the VGG16 Model after 10 epochs:
This result was obtained without data augmentation on the dataset as that proved to be very time consuming, given the complexity of the model.
Having settled on the best model, I found the best hyperparameters of our features? augmentation, and augmented the size of our data set.

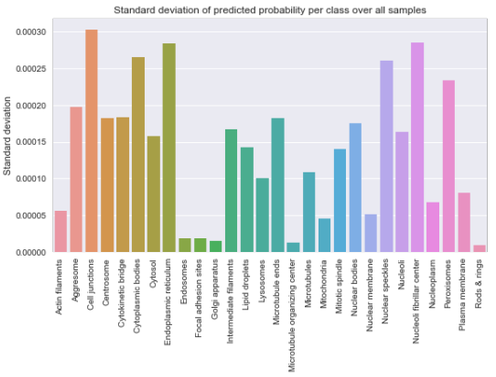
8. Results: Performance against validation set

8.1. U-net

The U-net performed the worst of the three models I settled on. A major component of error was the lack of information due to the massive downscaling of images that were fed in. I saw improvements in macro-F1 with bigger images but were hard pressed to keep image size at 128x128 for uniformity across models. One characteristic of the U-net was that there was a great deal of volatility in the macro-F1 score along the epochs.



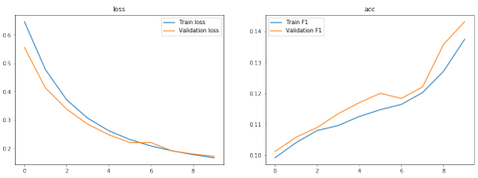





```
Individual F1-scores for each class:
[0.73139378 0.15753425 0.31086439 0.23152709 0.2533532  0.24495892
 0.16166282 0.25434783 0.0173913  0.00529101 0.00501253 0.16692427
 0.11715481 0.04180715 0.23024831 0.04878049 0.04854369 0.01529412
 0.08224299 0.16030534 0.1        0.41785714 0.18493827 0.34002677
 0.1509434  0.56630824 0.09375    0.        ]
Macro F1-score CV = 0.1806593611277381
```

8.2. VGG16

Below is the VGG16 Model performance on the validation set after 25 epochs:




```
Individual F1-scores for each class:
[0.80014276 0.73640167 0.65555556 0.55363322 0.6875     0.52988048
 0.45070423 0.5913371  0.11764706 0.10344828 0.07017544 0.46889952
 0.31515152 0.36842105 0.76683938 0.016      0.22058824 0.11764706
 0.29299363 0.32900433 0.16666667 0.62188516 0.43137255 0.71728595
 0.33766234 0.65504087 0.16666667 0.        ]
Macro F1-score CV = 0.4031625247514186
```

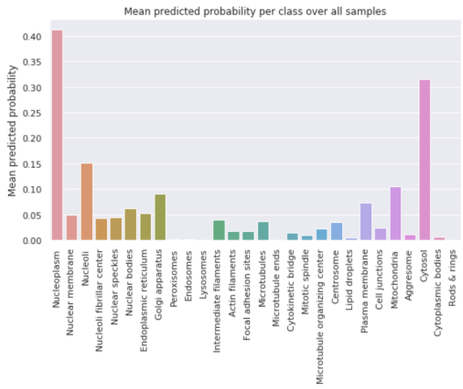
	Precision	Recall	F1
0	0.727922	0.877152	0.795600
1	0.745455	0.621212	0.677686
2	0.610553	0.680672	0.643709
3	0.513699	0.513699	0.513699
4	0.763314	0.675393	0.716667
5	0.538462	0.464945	0.499010
6	0.400000	0.390244	0.395062
7	0.655738	0.553633	0.600375
8	0.136364	0.333333	0.193548
9	0.023256	0.333333	0.043478
10	0.058824	0.500000	0.105263
11	0.342466	0.409836	0.373134
12	0.219298	0.328947	0.263158
13	0.234375	0.357143	0.283019
14	0.941860	0.743119	0.830769
15	0.001220	0.500000	0.002433
16	0.131579	0.178571	0.151515
17	0.277778	0.312500	0.294118
18	0.434783	0.333333	0.377358
19	0.411392	0.389222	0.400000
20	0.059701	0.500000	0.106667
21	0.547771	0.624697	0.583710
22	0.318471	0.568182	0.408163
23	0.617857	0.578595	0.597582
24	0.294118	0.357143	0.322581
25	0.583700	0.661673	0.620246
26	0.132353	0.236842	0.169811
27	0.000000	0.000000	0.000000

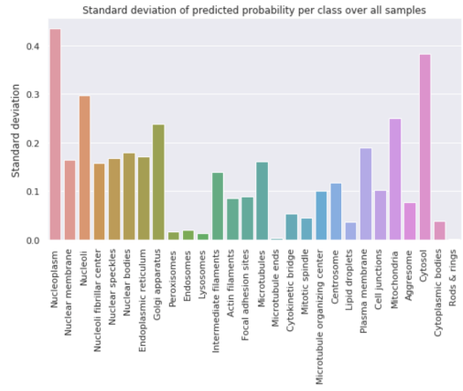
This result was obtained without data augmentation on the dataset as that proved to be very time consuming, given the complexity of the model.

8.3. Spoon Net

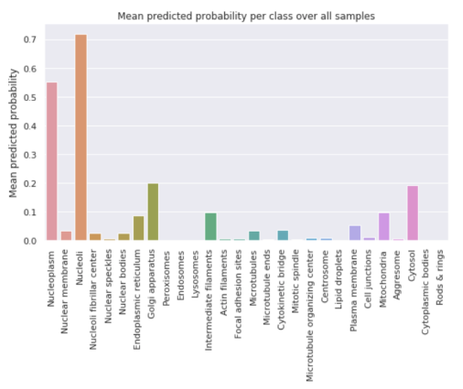
As discussed, the spoon net was an amalgamation of the Unet and the concepts behind the VGG model. It gave the best results on the validation dataset. I saw a great deal of fluctuation in the f1 of the validation set, even as the train set f1 continued to rise.

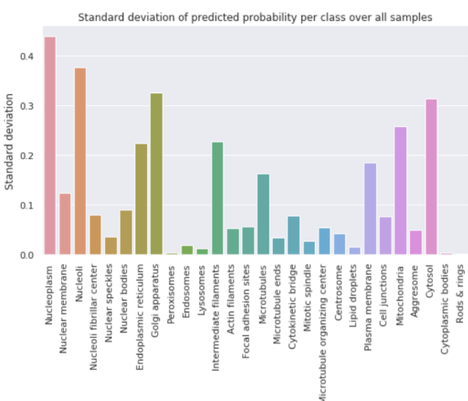
Without Data Augmentation:





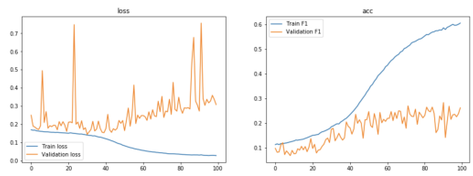
With Data Augmentation:





We can observe that with this model with a mix between stability from the VGG model and a lot of noise from the u-net during the training. I think that a deeper cascade of CNN would bring a better stability but it will take more time to train and would increase the complexity of the model and overfitting that can already observe in the following graph.

After the training, I used a brut-force optimization of the threshold in order to reach a better macro F1-score. This is



possible on both training set and validation set. Also, after the optimization of the threshold for each labels, we obtain the following results:

Results comparison:

	Precision	Recall	F1
0	0.990120	0.967003	0.978425
1	0.942857	0.823529	0.879163
2	0.956989	0.927083	0.941799
3	0.911439	0.872792	0.891697
4	0.866092	0.904019	0.884649
5	0.942317	0.888938	0.914850
6	0.965358	0.902808	0.933036
7	0.972808	0.946309	0.959376
8	0.781250	0.568182	0.657895
9	0.860465	0.880952	0.870588
10	0.960000	0.923077	0.941176
11	0.933921	0.873326	0.902608
12	0.936685	0.821895	0.875544
13	0.892057	0.884848	0.888438
14	0.975028	0.897597	0.934712
15	0.642857	0.473684	0.545455
16	0.735795	0.546414	0.627119
17	0.717949	0.721649	0.719794
18	0.892857	0.892857	0.892857
19	0.870170	0.856274	0.863166
20	0.856250	0.835366	0.845679
21	0.959294	0.952735	0.956003
22	0.869118	0.827731	0.847920
23	0.976097	0.934359	0.954772
24	0.863946	0.863946	0.863946
25	0.959385	0.966878	0.963117
26	0.896040	0.624138	0.735772
27	0.800000	0.363636	0.500000

	Precision	Recall	F1
0	0.902319	0.816942	0.857511
1	0.854768	0.607175	0.710005
2	0.903195	0.714614	0.797913
3	0.880396	0.612544	0.722442
4	0.893587	0.662418	0.760830
5	0.821423	0.611396	0.701016
6	0.848393	0.605832	0.706883
7	0.876404	0.708251	0.783406
8	0.666667	0.295455	0.409449
9	0.913978	0.505952	0.651341
10	0.976744	0.403846	0.571429
11	0.792143	0.534758	0.638488
12	0.768270	0.502451	0.607557
13	0.782383	0.533838	0.634644
14	0.937319	0.761755	0.840467
15	0.904762	0.250000	0.391753
16	0.480380	0.400316	0.436709
17	0.534483	0.399485	0.457227
18	0.798511	0.528325	0.635909
19	0.763553	0.540875	0.633207
20	0.695980	0.422256	0.525617
21	0.871631	0.639343	0.737632
22	0.793857	0.470588	0.590899
23	0.913132	0.714647	0.801789
24	0.786464	0.573129	0.663060
25	0.785454	0.763060	0.774096
26	0.793624	0.407759	0.538724
27	0.125000	0.068182	0.088235

Macro-F1 Score: without then with data augmentation

9. Key Takeaways

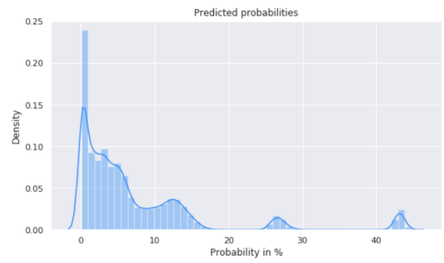
Our goal at the beginning of this project was to create an image processing pipeline that could improve outcomes for less frequently occurring labels in the dataset. I believed that this would give us an edge in the competition and boost our score. Our pipeline contained, amongst other things, an edge detection by computing the gradient using a Sobel kernel, compute this gradient in multiple direction and even combine those gradients with a threshold. All of those preprocessing contain hyper-parameters customizable like number of gradient orientation. Eventually, our pipeline augment the images for each simple by adding new channels with the previous methods before reducing the size of the image in order to keep as much information as possible. During our tests, I reduced the size by using the mean of blocks with a certain size which is customizable too. However, I found out that this method was not able to reach a macro-F1 score above 10 with multiple models (Resnet, Unet,VGG). I then decided to change our strategy by focusing our efforts on a new type of model by combining a U-net and the general idea behind the VGG16 (very deep convolutional neural net). This gave rise to the Spoon-net. Without any augmentation and using 4 channels with images size 128x128 I reached a macro-F1 score of 0.85. However, I quickly saw that this model overfitted the training set as the macro F1 loss started to diverge after 40 epochs and the score on the validation set never overpassed 0.30. To avoid this overfitting, I created a pipeline that read our images and transformed them randomly by one of the following methods:

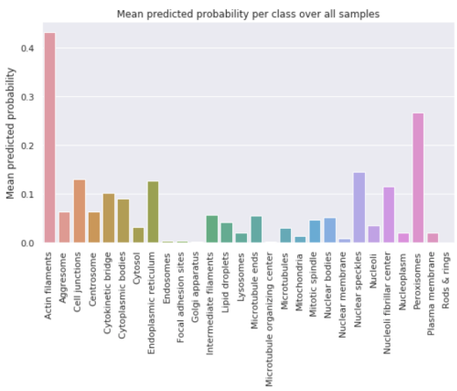
rotation of the images with 4 angles, two flips (vertically and horizontally), cropping, Gaussian noises, Gaussian blur, contrast normalization and multiplication by a small coefficient all pixels. In this way, I have been able to reduce the bias between the training and validation set. Indeed, both loss and macro-F1 seemed to be more stable on both set. The training however takes much more time due to the preprocessing. Due to the size and time constraints, I did not run more than 100 epochs on this new pipeline. One thing I could have done was use more suitable pre-trained models in our pipeline that would have allowed us to converge faster. I also would have liked to improve our pipeline through various means. For instance, adding new channels that contain the gradient in multiple direction. Finally, one thing that could have helped with our results would be to research the natural occurrence frequency of the protein labels and use those probabilities to optimize our training set.

10. Additional results

For the following graphs, I used a baseline model (four new channels: edge detection of of the four original channels using a Sobel kernel with images of size 64):
”first layer” : 3 CNN in series and one dense layer
”second layer”: 3 CNN in parallels and one dense layer
The ”first” and ”second layers” merged into one dense layer before the last layer of size 28 and using a sigmoid activation.

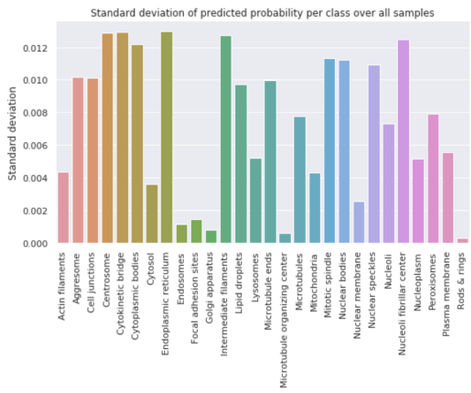
For this model, using edge detection helped to get a macro-F1score of 0.10 instead of 0.05 without edge detection after 20 epochs which proves that my work on the features augmentation might bring a reel improvement to the Spoon-net results if would be used.

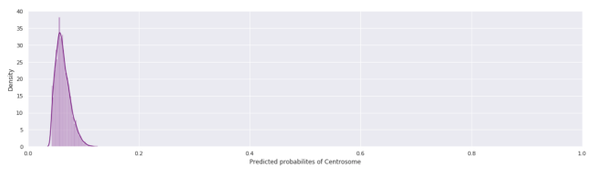




11. Previous work and references

[1] *NikitPatell* ”Best Tutorial for Beginner”
https://www.kaggle.com/nikitpatel/best-tutorial-for-beginner
Accessed Oct 25 2018
[2] *lafoss* ”Pretrained ResNet with RGBY”
https://www.kaggle.com/iafoss/pretrained-resnet34-withrgby-0-448-public-lb
Accessed Oct 25 2018
[3] *Tsung-Yi Lin et Al* ”Focal Loss for Dense Object Detection”
https://arxiv.org/pdf/1708.02002.pdf
Accessed Nov 1 2018





Example of Centrosome probability prediction in the validation set

[4] *O. Ronneberger and P.Fischer and T. Brox* ”U-Net: Convolutional Networks for Biomedical Image Segmentation”
https://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a/
Accessed Nov 1 2018
[5] *NikitPatell* IEtol ”Protein where art thou?”
https://www.kaggle.com/weegee/protein-where-art-thou
Accessed Oct 25 2018
[6] *Michal HaltufBest* ”Loss function for F1-score metric”
https://www.kaggle.com/rejpalcz/best-loss-function-forf1-score-metric
Accessed Oct 19 2018
[7] *NikitPatell* ”Four Img Combine in single img VGGModel”
https://www.kaggle.com/nikitpatel/four-img-combine-insingle-img-vgg-model
Accessed Oct 11 2018
[8] *Kevin Mader* ”Transfer Learning for Human Protein Submission”
https://www.kaggle.com/kmader/transfer-learning-forhuman-protein-submission
Accessed Oct 4 2018
[9] Karen Simonyan, Andrew Zisserman
Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015
Accessed Dec 5 2018
[10] *Davi Frossard*, VGG Implementation in Tensorflow
http://www.cs.toronto.edu/ frossard/post/vgg16/
Accessed December 5 2018