Protein Localization from High-Resolution Microscopy using Deep Neural Networks

Manuel Madeira (2018403425)
Dept. of Computer Science and Technology
Tsinghua University
Beijing 100084, China

Maria Teresa Parreira (2018403422)
Dept. of Computer Science and Technology
Tsinghua University
Beijing 100084, China

Abstract

We use a neural network for a multilabel classification task that intends to localize proteins within samples of several cells, given 28 available locations. The dataset is provided by the Human Protein Atlas and we perform image augmentation on underrepresented labels. Our model is a CNN with Global Average Pooling Layers and outperforms other models meant for tasks similar to that of this project, when considering the Macro F_1-Score. After fine-tuning our model, the best version achieves a Macro F_1-Score of 0.604469. We point out some limitations of our approach and of the project itself.

1 Introduction

The homeostasis of any organism cannot be accomplished without the homeostasis of its basic units: the cells. Within a cell, proteins are probably the most important component as they are included in every single mechanism and task of even the most simple kind of cell. The malfunction on the cycle of any protein can have a great impact on a unicellular or multicellular being - therefore, criteria such as a protein's composition, structure, shape and location are used to assess poor functioning. These last two parameters can be evaluated through Microscopy.

With ever increasing spatial and temporal resolution, Microscopy and Imaging are fast-developing fields, and this fact greatly influences the current techniques for diagnosis and study of the generated image data: so far, it has mostly been dependent on human evaluation; however, with the increase in resolution and on the amount of images generated, this becomes impractical and that is where the need for specialized software and hardware for data handling comes. Automated image analysis, which needs to be developed or adjusted for each specific task, is currently one of the biggest challenges in this field. [1].

This is where Machine Learning can play a pivotal role. By producing an abstract representation of the task, which includes an assessment of the relevant features, it tackles the issue of the variety of cell appearance. Moreover, it accounts for a far greater amount of samples than a human expert could ever witness. This perspective is very exciting and would revolutionize clinical imaging applications as well as several fields of study related to Cell Biology.

The problem mentioned above has been approached via several forms of deep neural networks (particularly beneficial because they overcome the feature selection problem). This type of approach is popular among some of today's biggest challenges in Biology and Health, including drug discovery and genomic data processing [2]. With Deep Learning outperforming humans in object recognition [3] and being able to process big amounts of data in an efficient way, this paradigm seems suitable for solving the protein localization problem.

Indeed, neural networks have been proven to perform well in this particular task [4, 5, 6]. Architectures based on Recurrent Neural Networks (RNNs) such as Long Short Term Memory (LSTM)

networks are more fitting for sequential data (*e.g.* predicting the localization of a protein based on its sequence of aminoacids)[7] and in dynamic biomedical signals, and less frequently used in static biomedical images. On the other hand, Convolutional Neural Networks (CNNs) are most commonly used in biomedical image analysis due to their outstanding capacity in analyzing spatial information.[8].

The specific task we set to complete (image recognition and localization) seems to be a perfect candidate for the use of a CNN-based model. Regarding the specific architecture of the net, sets of convolutional filters have been proven to perform very well in this kind of task, if combined with three fully connected layers [9, 5], resulting in a satisfactory extraction of features.

The complexity of the task increases, of course, when analyzing eukaryotic rather than prokaryotic cells, mostly due to the existence of a wider variety of subcellular structures, and even when compared to networks meant to locate proteins within yeast cells [5] we find that the problem needs to be tackled differently if the targets of the analysis are animal tissues. Additionally, we must take into account that any set of images derived from microscopy techniques is subjected to the existence of artifacts, which can further complicate the localization/classification task.

This project was a part of the "Human Protein Atlas Image Classification - Classify subcellular protein patterns in human cells" competition, hosted by *Kaggle* and *Human Protein Atlas*. The evaluation metrics and the dataset were defined according to the rules of the competition, and the final Macro F_1-Score was calculated by submitting a .csv file on the platform.

2 Handling the Dataset

Before designing the skeleton of the network that would allow for the computation of the desired task, it is rather useful to analyze the dataset, available at *Kaggle.com* (Human Protein Atlas). The training set consists of 31072 samples, each formed by four PNG files (*Fig 1*). Each image consists of several cells, of the same type but displaying different morphologies. There are 28 different locations available and, given that this is a multilabel classification task, within a sample we can have proteins in different organelles. As a way to evaluate performance, we are using macro F_1-Score, the unweighted average of the F_1-score for each one of the 28 labels.

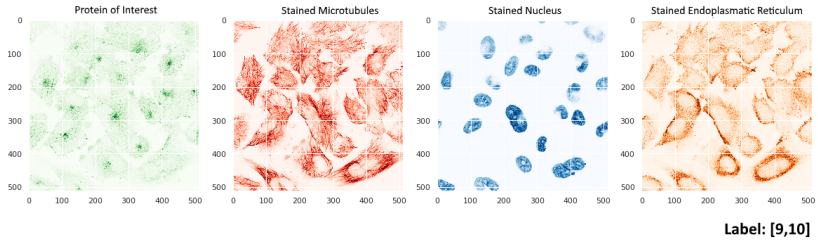

Figure 2.1: Visualization of the dataset. As mentioned, it consists of 512x512 PNG files, where each sample is represented by four different filters: one highlighting the protein of interest (green) and three regarding cellular landmarks: nucleus (blue), microtubules (red) and endoplasmic reticulum (yellow); additionally, each sample is associated with one or more labels, regarding the proteins location.

Figures A.1, A.2 and A.3 summarize the analysis of the dataset, regarding, respectively, absolute frequency of each label, correlation between labels and an assessment of the amount of labels per sample . The dataset is very clearly not uniformly distributed, as we can observe a larger incidence of certain locations (*e.g.* nucleoplasm) whereas some locations are seldom present within the training set labels (*e.g.* rods&rings, with an absolute frequency of 11). Drawing samples at random from the dataset revealed it consists of not only different cell types but also different levels of staining quality - some images have very good contrast whilst others are barely distinguishable.

Both these facts make our dataset the perfect candidate for oversampling, in particular through the technique of image augmentation.

2.1 Preprocessing

A fair amount of the most successful models meant for tasks similar to this one achieve such high levels of accuracy through a heavy load of preprocessing: Liimatainen[6] trains their model with samples containing only one label, excluding every sample within the training set that has two or more labels; in the *GapNet* paper[10], the authors exclude all the samples with at least one corrupted filter; Pärnamaa[5] achieves great results in the classification task by first thoroughly preprocessing the dataset, including a step where a random forest is trained to discriminate between cells and non-cells; Pärnamaa[5] and Godinez [11] build their model from high-throughput images but perform single-cell crops before feeding the samples to the network. However, we wanted our model to be as convenient and applicable on real life as possible - ideally, a model able to receive images directly from Confocal Microscopy (a highly standardized way of acquiring the images by itself). Training our model on single-cell samples, with high quality, wouldn't bring great advantage as the time saved on automation of classification would be spent by the machine on preprocessing. Moreover, that kind of preprocessing would be a cumbersome chore and wouldn't fit this course's program.

Concerns about overfitting behaviour were raised *a priori*. Taking into account that each sample carries a fair amount of information (4 512x512 PNG files), our dataset is not very large. Furthermore, considering the highly unbalanced distribution of labels (*Fig. A.1*) and the fact that the low F_1-Score of one class can be caused by low incidence of the class labels on the dataset, we decided to perform oversampling to the poorly represented classes. To achieve this, we performed image augmentation on the samples which had at least one label whose absolute frequency was lower than 500. As a result, we increased the size of our dataset from 32072 to 34098 creating slightly modified copies of these samples. We used different techniques of image augmentation so that we could ensure that our model wouldn't overfit those rare examples, but had at least 500 samples with a determined label to train on. A side effect of this procedure was the fact that we increased the absolute frequencies of the labels that were not rare but were present in the same samples as the rare labels. However, this point can be seen optimistically since, that way, our model will still train on a dataset with a labeled distribution similar to the one initially available (which we assume to be analogous to the ones available on real-life applications). The variation of absolute frequencies for each label is shown on Figure A.1, in red.

The effectiveness of using data augmentation on deep learning for tasks related to image recognition has been proven [12]. In our case, we decided to apply on each copied sample one of the following procedures:

- Mirroring horizontally input images with probability of 0.5;
- Applying gaussian blur with random sigma between 0 and 0.5 with probability of 0.5;
- Randomly cropping away pixels at the sides of the image;
- Strengthening or weakening the contrast (normalize contrast by a factor of 0.5 to 1.5 sampled randomly per image);
- Adding white (gaussian) noise - we sample the noise once per pixel or we sample the noise per pixel and channel (this can change inclusively the color of the pixels);
- Making images brighter or darker, by multiplying all pixels in an image with a value between 0.8 and 1.2. Moreover, with probability 0.2, we sample the multiplier once per channel, which can end up changing the color of the images as well;
- Applying affine transformations (scalling, translating, rotating and shearing).

3 Methods

3.1 Our Approach: GapNet

As it was already referred, the biggest limitation from the reported previous work on tasks of this kind was the fact that the inputs were preprocessed before being fed to the neural network - for instance, Pärnamaa [5] uses as input a 64x64 cropped image of a centralized single cell. However, given the heterogeneous nature of our data set and of some of the observed samples, this is not a very viable alternative, as some of the cells might get partially cropped if their shape is elongated; the labeled targets might also not be present in every cell but rather in only some of the cells of the

image. Consequently, the cropping could become a prominent source of misclassification. For these reasons, new models have been suggested that consider images with multiples cells [6] and the results seem to be satisfying, with F_1-Ccores of 0.51.

The authors of [10] suggest a different model - GapNet-PL. Instead of building high-level representations of each cell, it builds low-level representations by convolutional layers and these hints are then collected by layers where global-average pooling is performed. As mentioned in the article, it consists of a two-step approach: firstly, abstract features are captured by convolutional layers at different spatial resolutions (due to the max-pooling layers present). Secondly, three global-average pooling layers collect the information from the previous sequence, concatenate the features, and pass the vector to a fully-connected network.

One of the main advantages of this model is the fact that it allows the learning to take place with full-resolution inputs, instead of losing information in preprocessing downscale steps. Additionally, the global-average pooling steps at different levels of abstraction and spatial resolution will produce features statistics.

For our network, we mimicked that structure, building it from scratch and changing some of its characteristics (*Figure A.4*), in order to minimize the validation loss obtained . The inputs preserve the dimensions of the samples. Instead of SELU, we used ReLU as the activation function. Batch Normalization (BatchNorm), contrarily to what was suggested on the article that introduced this technique [13], was applied before the convolution (Conv) instead of being placed between the Conv layer and the activation layer (providing a structure BatchNorm-Conv-ReLU). Both these decisions were made based on experimental results and the performance of the network seemed to improve.

The input is normalized and followed by a Conv layer with 32 features and stride 1. A MaxPooling layer (stride 2) is preceded by BatchNorm. This structure is repeated with an increasing number of features in the Conv layers. The output of each of the two MaxPooling layers and of the last layer of this convolution chain is collected, suffers a dropout (dropout rate: 0.4) and passes through a global-average pooling (GAP) layer. The three vectors are concatenated and inserted into a fully-connected net with two hidden layers; the dropout rates are 0.4 for both. The last layer consists of only 28 features, correspondent to the number of labels, and uses the sigmoid function as activation (since we're dealing with a binary classification problem for each label).

The loss function which produced the best results was Binary Cross-Entropy with the default parameters. For the same reason, for optimization, we used Adam optimizer with a learning rate of 0.001. We used F_1-Score and Accuracy as metrics. As a consequence of having a small dataset, we decided to use k-fold validation, dividing our dataset into 3 non-overlapping partitions.

Since the results of our model are not integers, we had to apply a threshold to define the value which separates a positive (1) from a negative (0) prediction of our model for each class. This threshold is defined through the same process for all classes: on the validation set, we calculate the F_1-Score for each class, using a threshold which ranges from 0 to 1, incrementing it by 0.001 at each step. Then, for each class, we select the threshold which maximizes the F_1-Score. As a consequence, this set of different thresholds will maximize the Macro F_1-Score (average over the F_1-Scores for each class) of the prediction of our model (on the validation set).

This method of selecting the threshold may become a source of complications for classes whose labels are not present on the validation set - since any threshold will always yield an F_1-Score of 0, the default value (0) is going to be selected as the threshold for that class and thus the model will assign that label to every single sample. One solution would be to set the default threshold to 1, a strategy in which we're assuming that if the label isn't in the validation set, then it is a rare label. As a consequence, our model will always have negative predictions for that class. However, this is obviously an excessively restrictive condition and the F_1-Score for that particular class will be impaired. Because our dataset isn't large, we decided to proceed with a K-Fold validation strategy. This way, because we are covering the entire dataset, we will always have at least one incidence of each label (otherwise, it would imply that the class wasn't present on the dataset). We used K=3.

3.2 Alternative Models

In order to compare the performance of our model with other models designed for tasks of localization and classification, we built adapted versions of DeepYeast[5], M-CNN[11] and DenseNet121 [14].

DeepYeast The model was adapted for the task at hands. The shape of the input was reduced from 512x512x4 to 192x192x4. (Fig. A.5). Regarding the specific architecture of the net, sets of 3x3 convolutional filters have been proven to perform very well in this kind of task, if combined with three fully connected layers [5, 9], resulting in a satisfactory extraction of features - the small dimension of the filters is a trade-off for a larger amount of convolutional layers. The MaxPool layers have stride 2 and 2x2 windows. The dropout rate was set to 0.5 and the learning rate to 0.01, decreasing by half after 3 epochs if no improvements were observed. We used Stochastic Gradient Descent, momentum 0.9, weight decay of 0.005, batch size of 128.

M-CNN The M-CNN model (Fig. A.6) is designed for phenotype classification of human HTI data and was benchmarked on 8 separate datasets. The main idea of the architecture is to combine features extracted from the input at several spatial resolutions. These scaled versions of the input are processed by three convolutional layers and the feature maps of the last layer, respectively, are downscaled via pooling to the smallest resolution. Then, the feature maps are concatenated and combined via 1x1 convolution and passed on to a fully connected layer and the output layer. For this model, we determined a batch size of 22 and an initial learning rate of 0.01. The shape of the input was kept at 512x512x4.

DenseNet The basic idea of DenseNet is to reuse features learned on early layers of a network, containing fine-grained localized information, on higher layers which have a more abstract representation of the input. This type of architecture reflects on the model not having to relearn features through the network, which means that the individual convolutional layers have a relatively small number of learned filters. We picked DenseNet-121 based off on open source code [1]. The growth rate is 32, with 4 dense blocks, and we used a batch size of 32 and an initial learning rate of 0.001. We did not use the pre-trained version of this model, as we wanted to fully understand how a version of this architecture would capture the features from scratch.

4 Results

4.1 Fine-Tuning of Hyperparameters

Figure 4.1 displays two examples of how some of the hyperparameters of our model were defined. Performance was evaluated through the Macro F_1-Score for a total of 25 epochs of training.

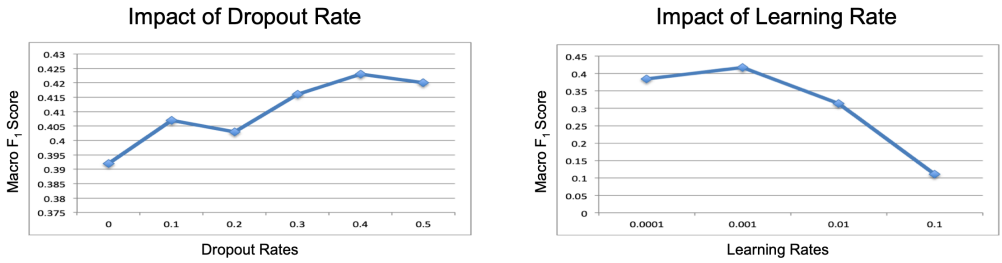

Figure 4.1: Some examples of fine-tuning. Left: Dropout Rate, with the best performance corresponding to a value of 0.4. Right: Learning Rate, with the best performance corresponding to a value of 0.001.

The training loss and accuracy over a total of 50 epochs, during the process of 3-fold validation, are shown on Figure A.7. From the top two plots, an overfitting behaviour is evident (the validation

[1] https://github.com/flyyufelix/DenseNet-Keras/blob/master/densenet121.py

loss ceases to follow the decreasing of the training loss after only a few epochs of training). The bottom plot (third run), however, does not display this phenomenon. We assign this fact to different characteristics on each part of the dataset used for validation: while on the two first cases, validation loss gets stuck on local minima, the partition of the dataset on the third case allows the model to continue decreasing its validation loss.

We picked three different models (one from each run of the K-Fold validation strategy): from the first one, we chose the model correspondent to epoch 35; from the second one, the model correspondent to epoch 23; from the last one, the model correspondent to epoch 46. This choice was made considering the best training and validation losses of the epochs in which the models didn't display a typical overfitting behaviour. The F_1-Scores per class for these models are shown below (Fig. 4.2):

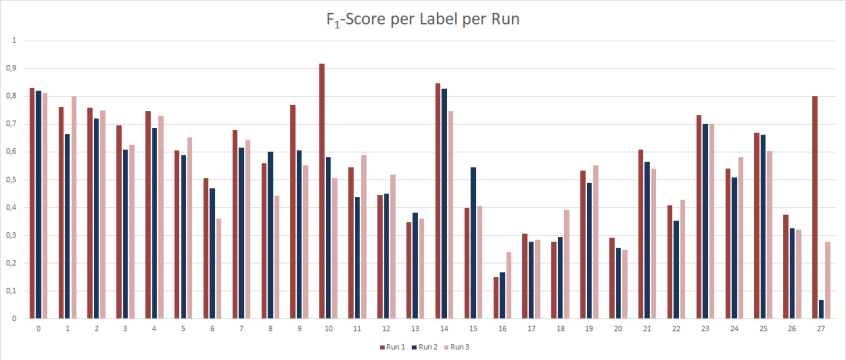

Figure 4.2: F_1-Scores per label obtained for each run of the K-Fold validation method

There are obvious differences on the F_1-Scores distribution per run. This fact is explained by the different incidence of each class on each run. Taking into account the Macro F_1-Score (unweighted average), we obtain:

- First model: 0.575345
- Second model: 0.509465
- Third model: 0.523693

Because the values above are fairly similar, we can consider the performance of each run to be consistent and it proves that our model is able to capture the right features even in such an unbalanced dataset.

With this in mind, we added a fine-tuning stage (still on the training set) after the training period. In it, we freeze all the layers that are part of the "convolution chain" and work only on the fully-connected network. The idea behind this decision is the fact that, after training, the first few layers have successfully captured the simpler features; however, because we are inputting images of high complexity and there are many labels (locations) available, it is essential that the last few layers are able to successfully capture subtle differences that might be pivotal for the classification task. Our solution is an attempt to avoid overfitting the model while still getting some performance improvement. For the fine-tuning process, we used the model which displayed highest Macro F_1-Score from the ones presented above (correspondent to the first run, epoch 35). Fig. 4.3 illustrates the loss values after a different number of fine-tuning epochs.

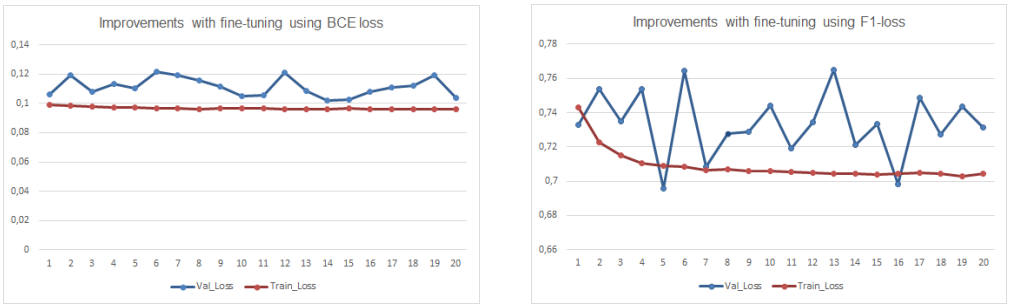

Figure 4.3: Training loss and validation loss over epochs of fine-tuning. Left: using Binary Cross Entropy (BCE) as the loss function. Right: Using F_1 loss as the loss function.

As expected, since our model has already been through a training stage using Binary Cross-entropy as the loss function, the results for that case are typical of an overfitting situation. For the case in which we use the F_1 loss, we can observe a situation of apparent learning (training loss is decreasing), even though the validation loss shows an unstable behaviour. The high values obtained for the F_1 loss will be further clarified on Section 5.1.

Again, we picked two models from the two fine-tuning runs mentioned above: the model correspondent to epoch 5 when using F_1 loss and the model correspondent to epoch 14 when using Binary Cross-entropy (BCE). This choice was made based on minimizing the validation loss. Surprisingly, the Macro F_1-Score obtained for the fine-tuning with BCE (0.604469) was higher than the one obtained with F_1-loss (0.580270), even though both of them are above the Macro F_1-Score of the model without fine-tuning (0.575345). Considering the best result obtained, which made use of the Binary Cross-Entropy, we hypothesize that the fine-tuning allows the model to better approximate the distribution of labels - an effect of using this particular loss function - without a representative improvement on the validation loss, but with consequences on the Macro F_1-Score. This experiment further confirmed that the Binary Cross-entropy is a better loss function for this task, as had been concluded for the training stage.

The F_1-Scores per label were also evaluated as a way to understand which locations represent a more troublesome classification task (Fig. 4.4).

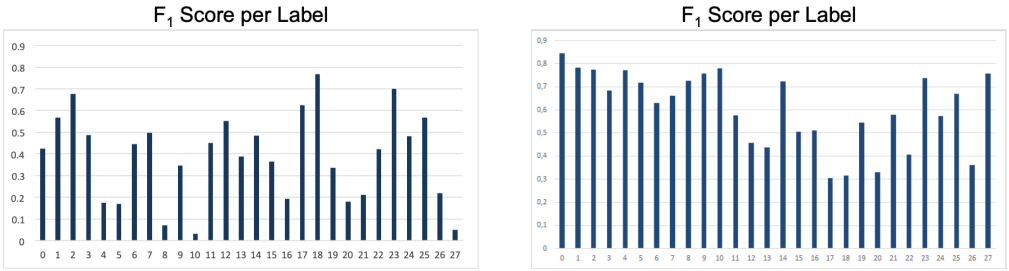

Figure 4.4: F_1-Score per label. Left: after 25 epochs of training, on the original dataset, using 1 epoch of fine-tuning with F_1 loss function (plot presented during poster session). Right: after 35 epochs of training, on the augmented dataset, using 14 epochs of fine-tuning with Binary Cross Entropy as loss function (model with the highest Macro F_1-Score).

4.2 Evaluating Performance

Table 4.2 displays a comparison of the best performances for each of the models built, when tested in the original dataset (before oversampling via image augmentation). Our model, a version of GapNet, showcased the best Macro F_1-Score out of all the models.

Table 4.1: Macro F_1-Score of the best-performing version of each model tested.

Performance	
Model	Macro F_1-Score
DeepYeast [5]	0.145
M-CNN [11]	0.235
DenseNet[14]	0.319
GapNet[10]	0.433

The first submission of the model on *Kaggle*, which evaluated the model on a private test set, consisting of the basic structure of our model with parameters similar to the ones on the original *GapNet* paper, produced a Macro F_1-Score of 0.227. After tuning it to our task, which involved optimizing dropout and learning rates, changing the optimization function (from Stochastic Gradient Descent to Adam Optimizer),as well as the number of epochs of training, stride of Convolutional Layers and activation function (from SELU to ReLU+Batch Normalization) we achieved a Macro F_1-Score of 0.433 by the time the competition ended.

Once the deadline for submissions ended, we were unable to evaluate our model through the same process. However, we still proceeded with major changes to the project, including oversampling, k-fold validation, optimization of fine-tuning. This leads us to believe that a submission with the final version of the model would result in a considerable increase of the Macro F_1-Score on the test set (see Fig. 4.4, right plot - the distribution of the F_1-Scores per label displays a very clear improvement).

5 Discussion

5.1 Loss Function and Performance

On the poster session, one of the points we had yet to investigate was the behavior of the model in regards to the choice of the loss function. When training the model, using Binary Cross-Entropy as the loss function displayed better results than directly using the Macro F_1-Score loss, which seems contradictory since the evaluation criteria is, in fact, the Macro F_1-Score. Reviewing some theoretical material provided insight on the subject and brought us closer to an explanation. Firstly, we should refer that the Macro F_1-Score can be the source of problematic performance, as it is not differentiable, which means we can't use it as a loss function in its "pure" form. However, we can modify it slightly in order to fulfill that requirement, for example by changing its inputs from 0/1 integer predictions to probabilities. In other words: if the ground truth is 1 and the model's prediction is 0.4, we account 0.4 as true positive and 0.6 as false negative; if the ground truth is 0 and the model's prediction is 0.4, we account 0.6 as a true negative and 0.4 as a false positive. Since our main objective is to maximize the Macro F_1-Score, value which is bounded by 0 and 1, the loss function should then be formulated as [1 - Macro F_1-Score]. Therefore, by minimizing this loss function, we'll be maximizing the evaluation metrics.

After changing the calculation of the F_1 loss function to one that was logically sound, the point that confused us the most emerged. As was mentioned above and as can be seen on Figure 5.1, the values of loss when using the F_1 loss function are incredibly higher in comparison to those obtained for the case in which we use Binary Cross-entropy as loss function.

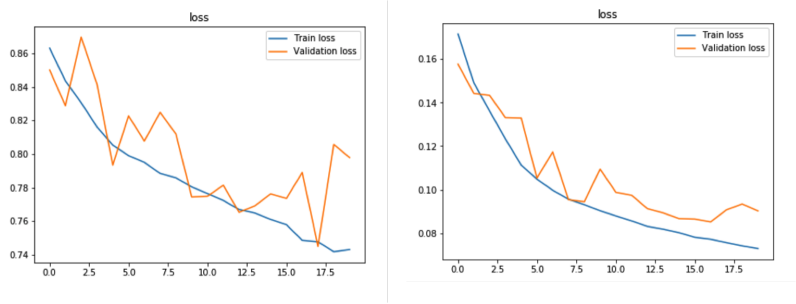

Figure 5.1: Example of validation loss obtained while using Macro F_1−Score loss function (left) and while using Binary Cross as loss function (right), for an early version of our model.

It is known that the F_1−Score is assumed to be 0 if there are no true positive samples for a determined class. Therefore, if we choose a small batch size (as is our case, given that we used a batch size of 20 as a consequence of memory limitations due to the size of the samples in our dataset), there's a high probability of rare labels not being present in it and thus the F_1−Score of those classes will be calculated as 0. This will, of course, have an impact on the Macro F_1−Score of each batch (unweighted average of F_1−Score calculated for each class in the batch), decreasing its real value. Since our loss function is calculated as [1 - Macro F_1−Score] and accounting for the fact that the already mentioned uneven distribution of the labels in our dataset will cause a decrease on the calculated Macro F_1−Score value, it becomes clear why the values obtained for validation loss were so high (the validation loss calculated by *Keras* for each epoch is calculated by averaging the Macro F_1−Scores obtained for each batch). This problem could easily be solved by a different *Keras callback*. Moreover, we can now explain the unstable decrease of the validation loss when in comparison with the subtle one shown in the case of Binary Cross-entropy as loss function: unlike the F_1−Score loss, the Binary Cross Entropy loss is not highly dependent on the batch constitution (which is randomly changed in every epoch).

5.2 Lowest F_1−Scores per label

Regarding Figure 4.4., from the plot on the right, we can see that the classes with the lowest F_1−Scores include the locations: Mitotic spindle (label 17), Microtubule organizing center (label 18), Lipid droplets (label 19) and Cytoplasmic bodies (label 20). The Human Protein Atlas[2], provider of the dataset used, gives some insight on how easy to detect (to a human) these locations are. The mitotic spindle can exclusively be detected during cell division, as it is a structure that attaches to the chromosomes and pulls them apart - this means that, if the dataset has few examples of cells in division stages, or if these samples have poor staining, the classification is a hard task; the microtubule organizing center is difficult to define as it is located in different parts of the cell, depending of its life stage. Besides, it can vary in shape and size if the cell is isolated or inserted into a cell line; the staining of the cytoplasmic bodies, on the other hand, can easily be mistaken with vesicle staining (which includes the staining of the lipid droplets) - meaning that the model probably does not clearly distinguish both labels and is missclassifying some samples.

5.3 Evaluating our Approach - Limitations

Given the choice of topic for our project, as well as the time and resources available, there are limitations to point out on our approach that had a cost on the final performance of our model.

Firstly, we chose to build a model from scratch, instead of using a pre-trained model and fine-tuning it to our specific task. This decision was made taking into account that this is our first project on the field of Machine Learning and thus we wanted to boost our knowledge regarding every stage of building a successful model. This decision represents a trade-off between a better

[2]https://www.proteinatlas.org/learn/dictionary/cell

insight on the subjects presented in class and the performance (evaluated through the Macro F_1-Score) of our model.

Then, the nature of our project constituted a limitation in terms of computation: working with high-resolution samples, our model was heavy and we had to adjust both its structure and that of the alternative models (used for comparison of performances), as well as batch sizes. We trained using a standard NV6 Microsof Azure VM, with a Tesla M60 GPU, 6 cores and 56GB of memory. Still, the last version of our model took us 1h per epoch of training (if not using cache memory) and the machine would often blow up after 20 epochs, if using cache. This slowed down our progress as it would take a fairly long time to test hypothesis.

The dataset represented an obvious limitation on the performance of our model. The poor-quality staining and the underrepresented labels jeopardized the learning stage, making it harder for our model to capture the relevant features. We must refer that there was an unofficial, external dataset, twice the size of the official one (more than 60k samples). However, the samples were not provided under the same format as our input - each sample was one file alone (instead of four, each representing one filter) and was composed of the superposition of 7 different filters. Naturally, a robust model (prevented from overfitting) will perform better if the training process involves a larger amount of samples. However, the download of this data and its conversion to the intended format would be slow and time consuming, without great pedagogical intake. It is given this context that the image augmentation option gained more preponderance, since we'd be able to increase our dataset without having to go through the referred cumbersome tasks (even though we're not exposing our model to new and different samples, just adding some noise to the already known ones).

The behaviour of our model seemed very prone to overfitting. Image augmentation was used as a technique to prevent this effect. However, if augmentation is performed on bad-quality samples, its benefits are inhibited. In order to further tackle the overfitting behaviour, after some investigation, we put forward some strategies which seemed promising but we did not have time to implement. The training could have been done on different filters, individually, rather than considering all the channels at once; this is a viable, interesting option because the relevance of each channel for the classification task differs, as the structures being stained are more or less important for the localization of organelles within the cell; then, the model could be assembled as a conjugation of the models trained on each individual filter. Furthermore, we could have used an autoencoder for the training, with the classification head being connected to the same point as the decoder; after the classification training, the decoder would be removed. This way, we would guarantee a more abstract representation of our inputs, which could remove irrelevant information that is causing the model to overfit - a sort of attention mechanism. We could also have assemble our model with pre-trained models as backbone, such as ResNet, DenseNet or Inception.

6 Conclusion

The task we proposed to solved seemed rather complex and as the project progressed we realized that, indeed, it poses a challenge. The format of the samples and their quality - a conjugation between images with a lot of information (many cells) and cases in which this information, due to poor staining or damaged cells, is not properly conveyed - as well as the amount of labels (28) on this multilabel classification problem, made this project a arduous one.

The work of localizing a protein on a tissue/cell sample is, in itself, not a straightforward task by any means, and even human experts struggle to accurately classify samples [10]. As of now, the perspective of a commercialized software for this task seems unlikely, as it would have to be thoroughly tailored for the specific task at hands (*e.g.*, for a specific type of cell or protein). Furthermore, the key for high-level performances, as mentioned, is a cumbersome stage of pre-processing the samples fed to the model for training, which in itself takes up a lot of resources (such as hiring human experts to personally approve each sample in terms of quality and labelling. However, as new Machine Learning strategies are developed, as well as with an increase on computational power and decrease of its costs, an automated version of this task is surely a reality in a near-future.

References

[1] Pepperkok, Rainer and Ellenberg, Jan (2006). *High-throughput fluorescence microscopy for systems biology*. Nature Reviews Molecular Cell Biology,7: 690–696.

[2] Webb, Sarah(2018). *Deep Learning for Biology*. Nature,554: 555—557.

[3] Kaiming, He *et al.* (2015). *Deep Residual Learning for Image Recognition*. Nature Reviews Molecular Cell Biology,7: 690–696. CoRR. Available at:
`http://arxiv.org/abs/1512.03385`

[4] Boland, Michael V. and Murphy, Robert F.(2001). *A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of HeLa cells* Bioinformatics, 17(12): 1213-1223.

[5] Pärnamaa, Tanel and Parts, Leopold (2017) . *Accurate Classification of Protein Subcellular Localization from High-Throughput Microscopy Images Using Deep Learning*. G3 (Bethesda, Md.),7(5): 1385–1392.

[6] Liimatainen, Kaisa *et al.* (2018). *Cell organelle classification with fully convolutional neural networks*.

[7] Kaae Sønderby, S. *et al.* (2015). *Convolutional LSTM Networks for Subcellular Localization of Proteins*. ArXiv e-prints.Available at:
`http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf`

[8] Cao, Chensi *et al.* (2018). *Deep Learning and Its Applications in Biomedicine*.
Genomics, proteomics & bioinformatics, 16(5):17–32.

[9] Simonyan, Karen and Zisserman, Andrew(2015) *Very Deep Convolutional Networks for Large-Scale Image Recognition*. CoRR. Available at:
`http://arxiv.org/abs/1409.1556`

[10] Anonymous authors (Paper under double-blind review)(2018). *Human-level Protein Localization with Convolutional Neural Networks*. Available at:
`https://openreview.net/pdf?id=ryl5khRcKm`

[11] William J Godinez, Imtiaz Hossain, Stanley E Lazic, John W Davies, Xian Zhang (2017). *A multi-scale convolutional neural network for phenotyping high-content cellular images*. Bioinformatics, 13(33):2010–2019.

[12] Jason Wang, Luis Perez (2017). *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. CoRR. Available at:
`http://arxiv.org/abs/1712.04621`

[13] Ioffe, Sergey and Szegedy,Christian(2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. CoRR. Available at:
`http://arxiv.org/abs/1502.03167`

[14] Gao Huang, Zhuang Liu, Kilian Q. Weinberger(2016). *Densely Connected Convolutional Networks*. CoRR. Available at:
`https://arxiv.org/abs/1608.06993`

A Appendix

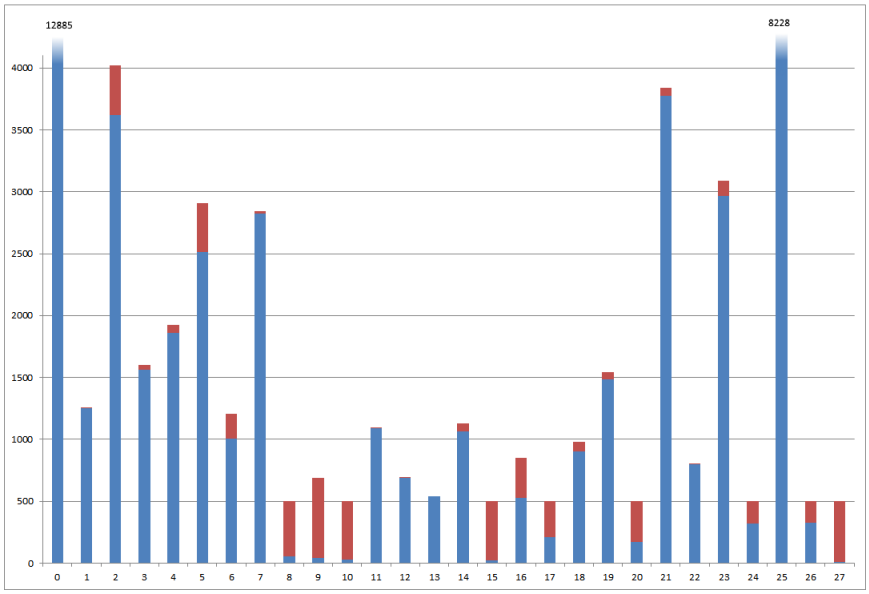

Figure A.1: Absolute frequency of each label in the training set - original (blue) and augmented (blue + red).

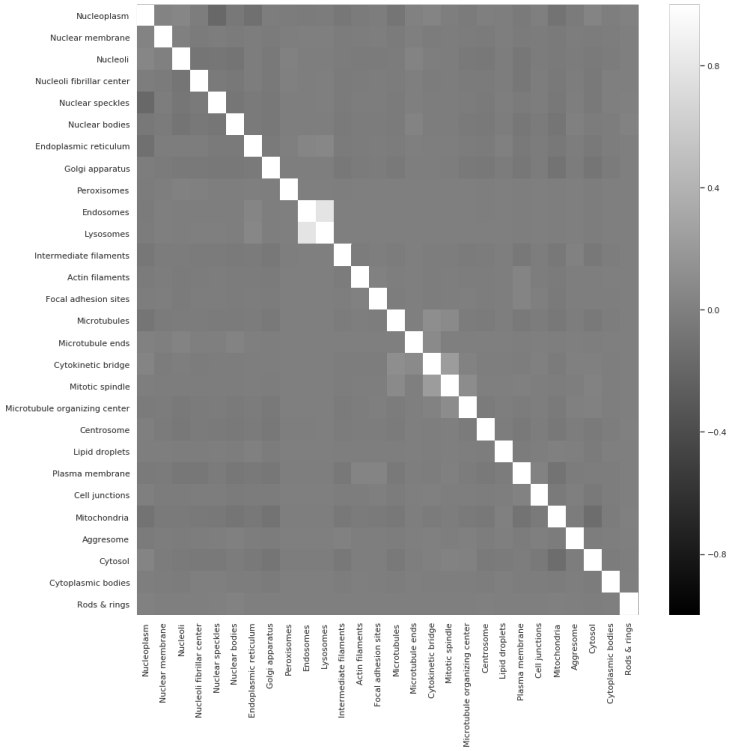

Figure A.2: Correlation between labels. It seems that there's very little correlation between most targets, although some relations can be clearly established between Lysosomes and Endosomes and also between structures usually present during cell division (Cytokinetic brigde, Mitotic spindle).

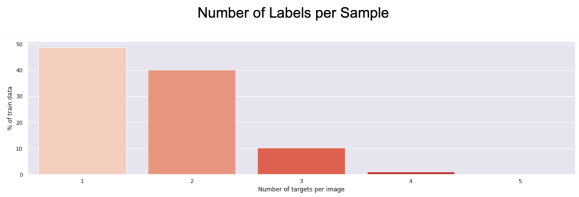

Figure A.3: Number of labels per sample. The great majority of samples only has one label, a significant amount has two labels but only a residual part has three of more locations.

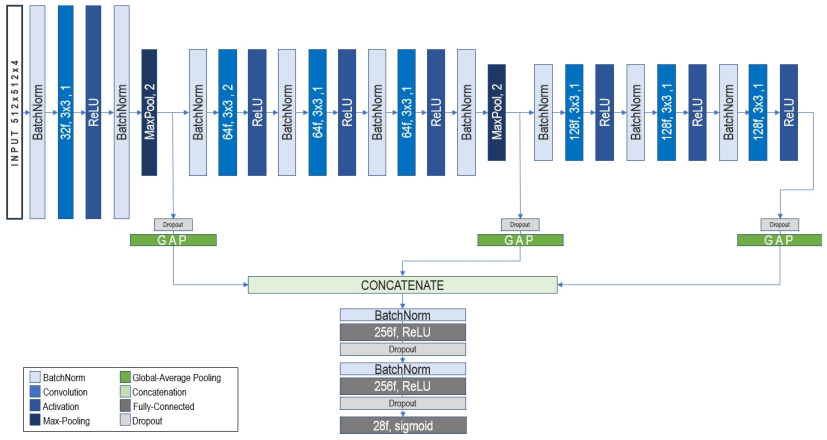

Figure A.4: Structure of the model (based on *GapNet-PL* [10]). The inputs have the original dimensions (512x512x4). Convolutional layers are described by the number of features per filter, the size of their windows and stride. The MaxPooling layers are described by stride. The Fully-Connected layers are described by number of features and activation function.

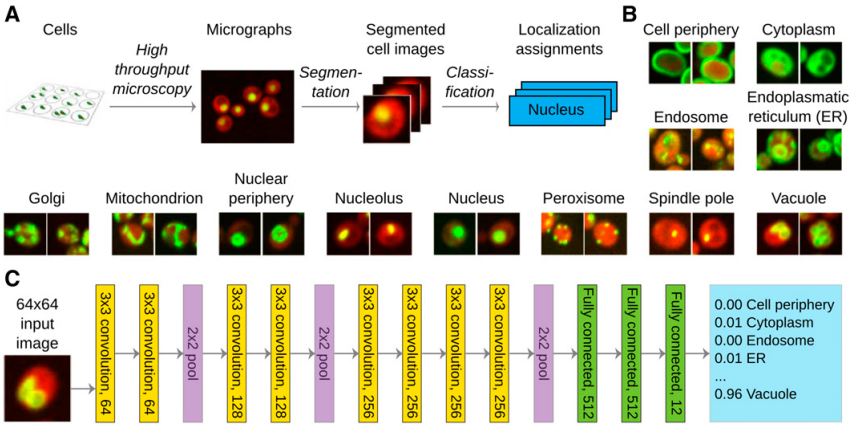

Figure A.5: DeepYeast. (A) Outline of the data generation and classification workflow. (B) Example pictures from each of the 12 classes. (C) Architecture of the *DeepYeast* convolutional neural network. Eight convolutional layers (yellow) are succeeded by three fully connected ones (green), producing the prediction(blue). *Image from [5].*

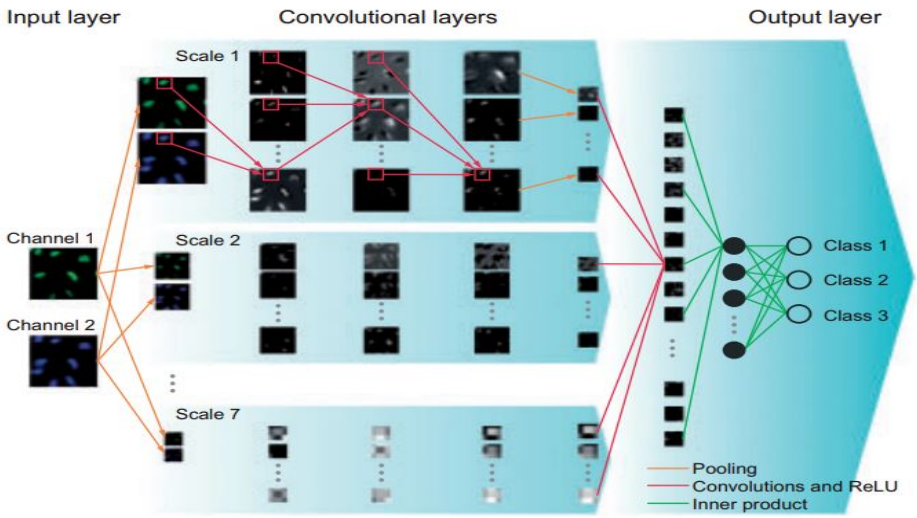

Figure A.6: Multiple CNN. *Image from [11].*

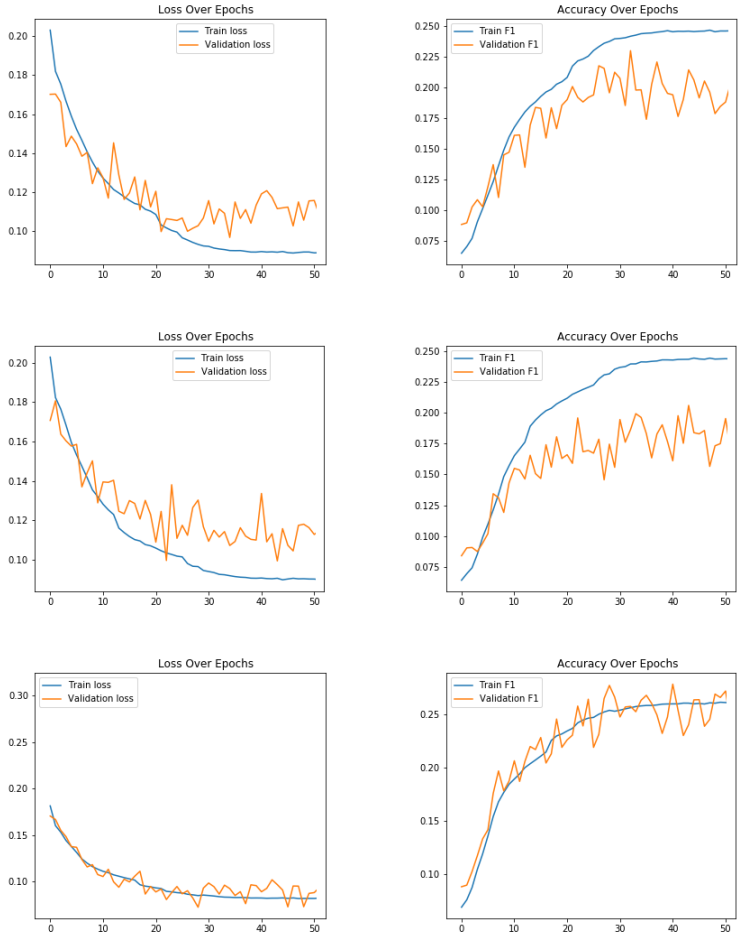

Figure A.7: Loss and Accuracy over epochs of training, for the 3-fold validation technique. Top to bottom, using as validation set the first, middle and final 1/3 of the dataset,respectively.