Adipocyte Cell Challenge
The HASTE Team

Philip J Harrison, Håkan Wieslander, Ankit Gupta,
Ebba Bergman & Erik Hallström

November 2020

1 High level description of submitted solution

In this report we detail our solution to the Adipocyte Cell Challenge hosted by AI Sweden and Astra Zeneca. The purpose of this challenge was to utilize machine learning to devise a method whereby a z-stack of bright field images can be transformed/translated to their corresponding single plane fluorescence images. This translation should be as faithful as possible to ensure that the downstream analysis from the generated images for each fluorescence channel (including morphological, intensity and textural measurements) is very close to that obtained from the real images. In our solution the prediction of the three fluorescent channels was done via two different approaches, although each approach shared some common underlying structures. The solutions for the lipids and cytoplasm channels employed generative adversarial networks (GANs) whereas the nuclei reconstruction solution employed a multi-task learning under privileged information (LUPI) solution.

Predicting the fluorescence image of the cell nuclei is the most challenging aspect of this challenge. The nuclei are not very apparent in the brightfield images. If there are significant errors in theses generated images the downstream segmentation of the nuclei will fail, a step that is crucial for obtaining subsequent measures of nuclear morphology, intensity and texture (and also for seeding the CellProfiler watershed algorithms for finding the cytoplasms). As an aid to the models we therefore provide a segmentation mask for the nuclei (derived from the fluorescence image) during training as privileged information. Learning using Privileged Information (LUPI; [17]) is a relatively new paradigm proposed by Vladimir Vapnik, the co-inventor of the support vector machine (SVM) learning algorithm. The key idea of the LUPI paradigm is that the machine learner, the "student", has an "intelligent teacher", that provides additional (privileged) information during the training phase, to improve and accelerate the learning process. Through LUPI, the learner gains a better concept of similarity between training objects and also receives hidden insights (explanations from the teacher) to guide decision rule formation. In our case we

apply LUPI through a multi-task learning framework based on a U-shaped architecture with one encoder and two decoders (one decoder for the fluorescence image and one for the segmentation mask). A loss weight map passed from the segmentation decoder to the image decoder is additionally applied to aid the nuclear fluorescence reconstruction [3]. Hence our solution predicts both the nuclei fluorescence image and its segmentation mask.

For the lipids and cytoplasm fluorescence predictions we utilized generative adversarial networks (GANs; [2]). In its basic form a GAN plays out a zero-sum game between two networks - a generator and a discriminator. The generator creates counterfeit images which it hopes to deceive the discriminator with, whilst the discriminator attempts to correctly classify the real and fake images. Convergence is reached when the discriminator can no longer tell the difference between the images. During the training of the generator the loss from this zero-sum game is added to the image reconstruction losses (such as L1, cross-entropy and gradient loss). The input to our discriminator includes both the brightfield z-stack and the fake or real fluorescence images and hence the discriminator not only asks whether the generated image looks real, but also whether or not it looks real conditioned on the bright field images. This conditional discrimination helps alleviate the problem of hallucinations that GANs can produce and was used in the popular Pix2Pix algorithm [6]. Furthermore, we also utilized relativistic learning for our GANs [7]. In traditional GANs the discriminator estimates the probability that an image is real, whereas in relativistic GANs the discriminator estimates the probability that a real image is more realistic than fake images, on average, and vice versa. This relativism passes more information to the generator and can significantly improve both generated image quality and training stability.

For the generators we utilized tiramisu dense U-Net [8] architectures. A basic U-Net [15] is a fully convolutional network (enabling per pixel predictions) that has an encoder downsampling path followed by an upsampling decoder path, with skip connections between the encoder and decoder. These skip connections aid the upsampling path in recovering finer grained spatial information from the downsampling layers. The tiramisu network extends this basic U-Net architecture through the use of dense blocks consisting of batch-normalization - ReLU - convolution - dropout operations that are densely connected (within a block) to one another in a feed forward fashion (i.e. there is a skip connection with the output of each quartet of operations to every subsequent quartet). On the downsampling path there are also skip connections that hop over each dense block. These dense blocks and additional skip connections result in a network that has multi-scale deep supervision with feature propagation and reuse, permitting the training of deeper networks but with fewer parameters than would otherwise be required.

See Figure 1 for an overview of the training schemes for the reconstructions of the three fluorescence channels. This figure will be further explained and should be referred back to during the reading of the coming sections in this report.

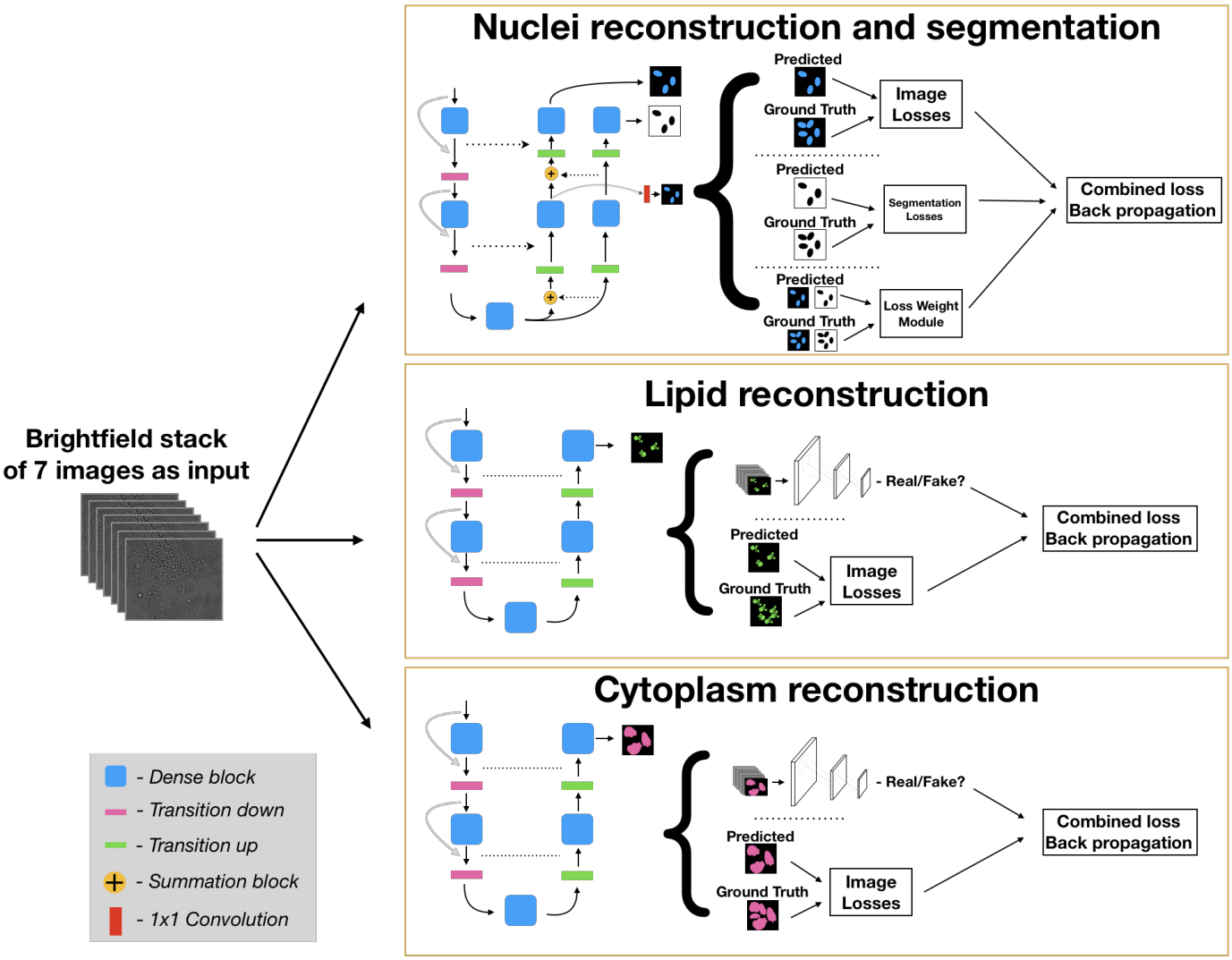

Figure 1: Overview of the training for the different channels

We explored a variety of different loss functions and in the end converged on the following five: log squared loss (for better handling of small values in the output of the network); median mean absolute error (for optimizing the evaluation score); image gradient loss (to better delineation between objects); and cross entropy for the segmentation model and relativistic least square loss for training the GANs.

We employed several data augmentations during training, including: random cropping; horizontal and vertical flips; rotations; random shifts; and scaling. More details of these augmentations, and when they were applied, are given in

the later sections of this report.

We trained the models for each channel and each resolution (20x, 40x and 60x) separately. We explored the prediction of all three resolutions simultaneously. A commonly applied data augmentation technique is to zoom-in on sections of an image to aid model training, hence having data at multiple resolutions may work well. In our case, however, we achieved better results by training on each resolution separately.

2 Data Processing

To get the data into a suitable range for training the neural networks the data was scaled to values between zero and one based on the minimum and the maximum values of the respective channels in the full dataset:

$$im\_scaled = \frac{image - minimum}{maximum - minimum} \tag{1}$$

Segmentation masks for the nuclei were extracted from the provided CellProfiler script. These masks were used to enhance the performance of the nuclei prediction network. For each segmented nucleus the center coordinate was extracted to make it possible to crop patches with each nucleus included.

Data augmentations were performed "on the fly" whilst training the network by first applying random crops then flipping left, right, or both, and adding 90 degrees rotations. For the nuclear prediction networks random shifts, scaling and rotations were applied around each nucleus. These additional transformations have proven beneficial in previous work [18] to improving segmentation results through removing the bias of centered nuclei in the input patches. As these augmentations are performed "on the fly", by the probabilities given in the configuration files, the size of the base dataset remains the same. The flipping and 90 degree rotations of the patches results in an eight fold increase in the variety of data seen by the model during training.

The scripts used for loading the datasets are provided in the *data* subfolder of our GitHub repository. This subfolder includes all the scripts needed for data normalization or standardization and the application of the augmentations as specified in the configuration files.

2.1 Data Processing Parameters

The maximum and minimum values for scaling the data to the range zero to one are presented in table 2.1

For the augmentations random crops were applied before inputting the images through the network. The crop size was chosen to be 512 x 512 pixels based on experiments showing that this largish crop size was beneficial for training all networks. Flipping and 90 degrees rotations were then applied with a probability or 0.5. Additionally, for the nuclei channel, shift, scale and rotate augmentations were applied with a probability of 1.0. The shift was set to a maximum of

Table 1: Minimum and maximum values per channel for normalizing the training data between zero and one.

		20x		40x		60x	
Channel	Z	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
01	01	0	24835	0	17794	0	15169
02	01	0	3467	0	2633	0	5328
03	01	45	65535	30	32504	23	11177
04	01	205	13522	1	5900	0	4796
04	02	183	14515	4	5547	0	4359
04	03	123	18461	7	5151	0	4166
04	04	109	19788	1	4703	0	4414
04	05	134	19200	13	4362	0	4417
04	06	144	17543	4	4888	0	4793
04	07	183	15875	23	4794	11	4225

0.2 times the crop size and rotations to a maximum of 180°. All augmentations was performed with the Albumentations library [1].

We converted the images into numpy arrays for faster data reading. Loading a numpy array with all 7 brightfield stacks is 5 times faster than loading each brightfield image separately with openCV. However, our code base can accept either the tiff images or numpy arrays as inputs, which is again simply specified in the config files. The script to convert the images to numpy arrays is available in the *utils* subfolder.

3 Model Architecture

Our models were based on the Dense U-Net architecture [8], as described in section 1. This architecture has the same shape as a regular U-Net, with an encoder and a decoder path with concatenating skip connections in between, but with the regular convolutions replaced by dense blocks [5], as shown in figure 2, and has shown impressive results in, for example, predicting cell traction forces [14] and image restoration [4]. The base model that we built upon was taken from [13].

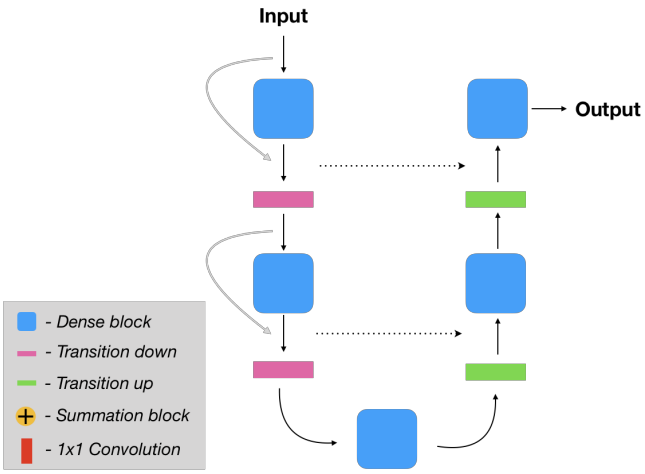

Figure 2: Example illustration of a DenseUNet

For the reconstruction component of the models we used pixel-shuffle followed by blurring to reduce checkerboard artifacts in the network outputs, as proposed in [16]. As the input and output of the network is normalized between zero and one we applied a scaled tanh function at the end of the network:

$$tanh\_scaled(x) = \frac{tanh(x) + 1}{2} \tag{2}$$

The tanh function has a steeper slope than the commonly applied sigmoid function and thus performs better at discerning differences at the margins of pixel intensities, i.e. those that are close to zero or one.

To better predict the nuclei channel we modified the network to have two reconstructive paths, one for predicting the nuclear stain (a regression decoder) and one to predict the segmentation map of the nuclei (a segmentation decoder). The segmentation decoder forces the attention of the nuclear image decoder into the areas where the nuclei are located. The segmentation decoder imposes attention to the nuclei locations by multiplying the output, before each transition up layer, to the regression decoder (see figure 3). This network also includes loss weight modules inspired by [3] and is illustrated in figure 4. The output after each dense block in the regression decoder goes through a 1x1 convolution before being passed to the loss weight modules. This reduces the number of feature maps and calculates the loss at each scale. The loss weight modules applies a weight mask created by calculating the difference between the segmentation output and the ground truth segmentation mask. This mask is then multiplied by a pre-defined weight value indicating how much influence the segmentation error will have on the reconstruction. For each of the different scale outputs from the regression decoder the loss weight module computes the reconstruction error between the output and the resized ground truth. This error is then multiplied by weight mask and the results at each scale is summed to produce

the final loss. In essence, the module focuses attention on areas that the segmentation decoder struggles with as these are also likely to be areas that are difficult for the regression decoder to accurately predict.

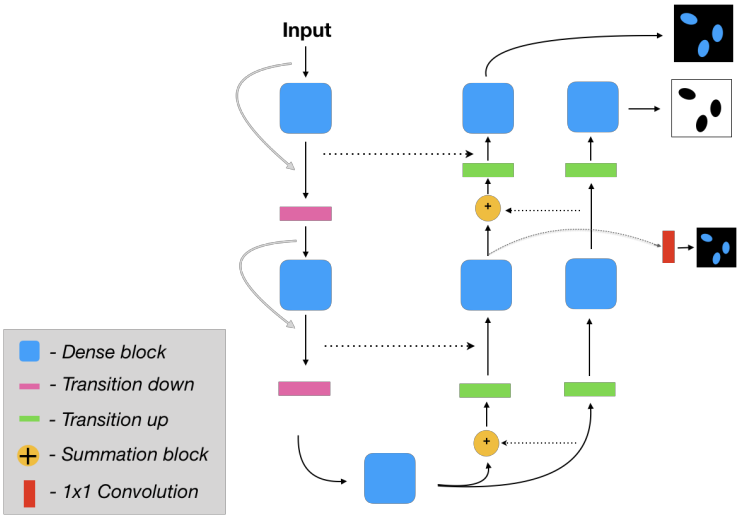

Figure 3: Proposed network for simultaneously learning the nuclei segmentation and the channel reconstruction

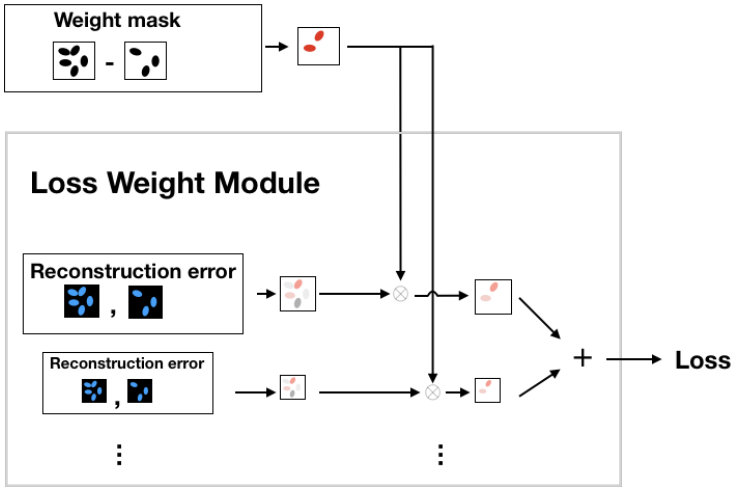

Figure 4: Illustration of how the loss from the loss weight modules are calculated

The regression decoder has an extra dense block in the end that is not connected to the segmentation decoder to give it extra freedom to predict the correct values.

For our GAN models we used a PatchGAN discriminator. This works by classifying patches in an image as real or fake instead of classifying based on the full image. This is achieved by limiting the receptive field of the network to smaller parts of the image. We used a four layer ConvNet (with batch-norm and leaky-ReLU [19]) to give the discriminator a receptive field of 70 x 70 pixels. This receptive field size was found to work well in the Pix2Pix algorithm [6]. Working with such patches promotes sharper outputs by better capturing local style characteristics and can be though of as introducing a form of style/texture loss.

3.1 Model Parameters

The tiramisu models are built with four dense blocks in the compression path and four in the reconstruction path. Each block has a size of four i.e four repeats of batch normalization, ReLU, convolution and dropout with skip connections in between. The dropout rate was set to 0.2 and the growth rate of the dense blocks was set to 12. In the config files these and additional hyper parameters can be seen and easily changed for exploring alternative configurations. We explored alternative values for these hyper parameters and found these to be the most suitable for the task at hand.

The final dense block of the regressor path of the nuclei prediction network was made smaller than than the other paths, with only 2 layers and a growth rate of 2.

3.2 Model Size

The base model for cytoplasm and lipid channels have 1.2 million parameters and the discriminator used has 1.7 million parameters. The nuclei model has 1.6 million parameters, the extra parameters comes from the the extra decoder that the nuclei network has.

The GPU memory usage for the GAN training (batch size of 4 with an input size of 512 x 512) takes approximately 19GB.

Training the Nuclei model with an input size of 512 x 512 and a batch size of 6 takes 37GB of GPU memory.

3.3 Loss Function

All models were trained with the Adam optimizer with beta1 = 0.9 and beta2 = 0.999, $\epsilon = 10^{-08}$ and weight decay=0. The learning rate were set to different values for the three fluorescence channels. These were determined by trial and error. For the lipid channel the learning rate for the generator was set to 10^{-4} and for the discriminator to 10^{-6}. For the the cytoplasm channel these same rate were used. The discriminators' learning rates were set much lower than the generators' to force it to learn more slowly. If the discriminator's rate of learning is too high it can too quickly become perfect at separating the real and fake images, thus passing no useful information to the generator. The weight

given to the adversarial loss, relative to the reconstruction losses, was set to 0.01 for both the lipid and cytoplasm channel.

In total we use five different types of losses: log squared loss (LsL); median mean absolute error (medMAE); gradient loss (gradL); relativistic average least squares (raLS) (eq (2) in [10]); and Binary Cross Entropy (BCE)

$$\mathcal{L}_{LsL}(x, y) = \frac{1}{N}\sum_{n}^{N}\left((log_e(\frac{y_n + \epsilon}{x_n + \epsilon})\right)^2 \tag{3}$$

$$\mathcal{L}_{medMAE}(x, y) = \frac{||x - y||_1}{median(y)} \tag{4}$$

$$\mathcal{L}_{gradL}(x, y) = ||\nabla x - \nabla y|| \tag{5}$$

Where x is the output and y is the ground truth. ϵ in eq. 3 is a small value (10^{-8} in our case).

4 Model Training

All models were trained with Pytorch version 1.7.0

4.1 Nuclei channel

The model for this channel is simultaneously trained with the nuclei channel and it's segmentation as the output. The training is done separately for different magnifications. The loss used for the output is:

$$\begin{aligned} loss = (\alpha_{lwm} * \sum_{k=1}^{n_{up\_blocks}} \mathcal{L}_{LWM}) + \alpha_{medMAE} * (\mathcal{L}_{medMAE} + \mathcal{L}_{medMAE_{weighted}}) \\ + \alpha_{medMAE} * (\mathcal{L}_{BCE} + \mathcal{L}_{BCE_{weighted}}) \end{aligned} \tag{6}$$

where $\mathcal{L}_{LWM}$ is loss weight module loss in the regression decoder blocks and is calculated by:

$$\begin{aligned} \mathcal{L}_{LWM} &= \sum_{k=1}^{n_{up\_blocks}} \sum_{i=1}^{W}\sum_{j=1}^{H} w_{ijk} * L^k_{medMAE} \\ \text{where } w_{ijk} &= \begin{cases} 2 & \text{pixel is misclassified} \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

and $\mathcal{L}_{BCE}$, $\mathcal{L}_{medMAE}$ are binary cross-entropy loss and median MAE on the output of the network respectively. $\mathcal{L}_{medMAE_{weighted}}$ and $\mathcal{L}_{BCE_{weighted}}$ are

calculated in similar way to $\mathcal{L}_{LWM}$ as:

$$\begin{aligned} \mathcal{L}_{medMAE_{weighted}} &= \sum_{i=1}^{W}\sum_{j=1}^{H} w_{ijk} * L_{medMAE} \\ \mathcal{L}_{BCE_{weighted}} &= \sum_{i=1}^{W}\sum_{j=1}^{H} w_{ijk} * L_{BCE} \end{aligned} \tag{8}$$

The 60x magnification is trained for 60 epochs with AdamW optimizer [12] and cosine annealing learning rate scheduler with warmup restarts [11] for 60 epochs with initial learning rate of 10^{-3} and $T_0 = 2$ and $T_{mult} = 2$ and then for 10 epochs with learning rate of 10^{-5} for the network to converge.

Since the number of nuclei are more in 40x and 20x magnification, they are trained for 30 epochs and 20 epochs respectively with same optimizer and learning rate scheduler as 60x and then trained for 10 epochs with lower learning rate for the network to converge.

4.2 GAN training

The generator network was pretrained, without the discriminator, with the LsL loss for 200 epochs with a batch size of 4. This pretraining puts the generator in a state where it is harder for the discriminator to differentiate between real and fake predictions. Subsequently, the discriminator is introduced and the generator is trained with the raLS loss as adverserial loss and gradient loss and median MAE loss for the image reconstruction losses. The total loss for the generator is thus:

$$loss = \alpha_{adv} * \mathcal{L}_{adv} + \alpha_{grad} * \mathcal{L}_{grad} + \alpha_{medMAE} * \mathcal{L}_{medMAE} \tag{9}$$

where α is the weight of each loss.

The discriminator is trained with only the raLS loss.

Instead of basing the GAN loss only on only new predictions we utilized an image "pool" training scheme which samples from previous discriminator predictions. The size of this "pool" was set to 50 i.e we saved a maximum of 50 previous predictions. Every time we get a new predication we replace an old one in the image pool with it. The discriminator thus learns both from new and old examples. This training scheme has shown to make the discriminator loss more stable.

As detailed in section 1, our GAN is a conditional GANs, i.e. we show the discriminator both the input to the generator and the output of the generator [6].

4.2.1 Lipid Channel

The learning rate for the pretraining is set to $10^{-}4$.

For the lipid channel we put the different loss weights to: $\alpha_{adv} = 0.01$, $\alpha_{grad} = 10$, $\alpha_{medMAE} = 1.0$. The values of the respective α is tuned by the

values of the corresponding loss and the amount of effect we want from each type of loss.

During the GAN training the learning rate for the generator is set to 10^{-4} and for the discriminator 10^{-6}, and the training was done for 1000 epochs with a batch size of 4.

4.2.2 Cytoplasm Channel

For the cytoplasm channel the hyper parameters were set to the same values as for the lipids channel with the exceptions that $\alpha_{grad} = 100$ and, due to faster convergence, only 600 epochs were required to reach convergence.

4.3 Consistency in training results

For reproducibility all models are trained with a fixed seed for python, random, numpy, cuda and pytorch. This makes it so that the models are initialized the same way, the data sampling is the same and augmentations are identical between consecutive runs. The seed is set to 42 for all instances.

Due to time limitations we did not fully test the consistency over multiple training runs with alternative random seeds. However, we have used 2 train/test splits and the results were consistent between those splits.

5 Model Testing

All networks are built to take a brightfield stack of 7 images as input. The inference is then done on overlapping tiles to reduce GPU memory consumption and to have similar receptive field at training and inference. For each tile a pyramidal weight mask is calculated to reduce edge effects that otherwise can occur. The weights are created such that the central pixels in the image has a bigger weight than the pixels on the image boundary. The pyramidal weight calculations are from the implementation in [9].

6 Analysis of Model Performance/Output

During model development we held out one of the eight wells for testing. For the final models that we submitted we reran our best models with all wells included in training but with a random selection of images from across all the wells (3 images for 20x, 4 images for 40x and 6 images for 60x, i.e. the equivalent of half a well in the numbers of test images).

For the lipid channel, training only with the LsL loss did not make it possible to detect any of the defective lipids, as became aware to use when we ran our output test images through the CellProfiler pipeline. Adding the adversarial training and the gradient loss made these defects clearer and gave a better score from the CellProfiler pipeline.

For the cytoplasm channel the main errors in the CellProfiler pipeline came from faulty detections of the nuclei (which were then used by the pipeline to seed the watershed algorithms to find the cytoplasms). We therefore put a significant amount of effort into reducing false positive and negative predictions in the nuclei channel. This was mainly achieved by improving upon the segmentation results of the segmentation decoder. The inclusion of the loss weight modules, passing information from the segmentation decoder to the image decoder, improved our test results from the CellProfiler pipeline.

For the image based evaluation, the median MAE loss was used by all networks to push the intensity values closer to the median of the ground truth, hence giving a better score for this metric in the final evaluation.

7 Model Execution End-to-End

7.1 Training

Our code base includes the option to train across multiple resolutions and imaging channels and can be easily explored by adapting the configuration files. In our GitHub repository we provide the configuration files used to create our final results. Through these configuration files the user can specify: (1) basic training info, such as the number of epochs to train for and the learning rates and schedulers to apply; (2) the losses to use, including their hyper parameters and the weights to apply to them when used in combination; (3) the models to train and their hyper parameters, such as the number of layers and filters and whether to use batch normalization; and (4) which dataset to use, it's location on the system, whether to use standardization or normalization, and which data augmentation techniques to apply during training. Should the user wish to train a model or apply a loss function not available in our code base these can be easily added to the corresponding model and loss folders and then called in the .json configuration file. The model is then run simply in the terminal by typing e.g.

```
python3 -m ai_haste -c my_config.json
```

A schematic diagram of the code base in our GitHub repostory is shown in Figure 5. Our code base was written in python and utilizes PyTorch for all the neural network components.

```
.
├── README.md
├── ai_haste
│   ├── data                : python scripts for loading the datasets required
│   ├── loss                : for loss functions not included in PyTorch
│   ├── model               : python scripts for the various neural networks
│   ├── tester              : scripts requied when running the models in test mode
│   ├── trainer             : scripts requied when running the models in train mode
│   └── utils               : utility functions (e.g. for converting the images to numpy arrays for faster data loading)
├── config
│   ├── train               : .json files for running the models to reconstruct the three fluorescence channels
│   │   ├── C1              : nuclei
│   │   ├── C2              : lipids
│   │   └── C3              : cytoplasm
│   └── test                : .json files for running the models in test mode for the three resolutions
│                             (across all three fluorescence channels)
├── docker                  : files for Docker image construction and running
└── exp_stats               : .csv files for image statistics and train/test splits
```

Figure 5: Schematic diagram of our code base on GitHub

In the python script __main__.py in the ai_haste directory all the random seeds are set and hence the user can easily reproduce our results. As only the GAN's generator is needed for testing (as with the non-GAN-based models for the nuclear channel) only one command line execution per resolution is required, e.g.

```
python3 -m ai_haste -c C2_20x_test.json
```

7.1.1 Lipids/Cytoplasm GANs

For running the GAN based models (for the lipids and cytoplasm channels) two config files run in succession are required:

```
python3 -m ai_haste -c C2_20x_pretrain.json
python3 -m ai_haste -c C2_20x_train_GAN.json
```

The first execution pretrains the generator for channel C2 (the lipids channel) and the second trains this generator alongside the discriminator (see section 4.2 and 6.1.1). Before beginning the GAN training the path to the pretrained model need to be specified.

```
train
├── load_pretrainde:true
└── pretrained_model_path:"" <--- path to pretrained model here
```

7.1.2 Nuclei

To rerun, for example, a replicate version of our final model for the nuclear channel (C1), at the 20x resolution, the user simply needs to execute the following command in the terminal:

```
python3 -m ai_haste -c C1_20x.json
```

7.2 Inference

To run the inference, run ai_haste as a module while specifying which config file to use. All config files for inference are located in the *configs/test* folder for the different magnifications. The module will run inference on all models (1 model per channel) at the same time and write the result in a pre-specified output folder.

To begin, download the different models and put them in the *models* folder. Then edit each config file in *configs/test* and specify the data path and the output path.

```
config
├─ "exp_folder" <--- path to output folder here
├─ "data"
   └─ "folder":""<--- path to data folder here
```

The inference is then executed by:

```
python3 -m ai_haste -c configs/test/test_20x.json
```

for inference on 20x,

```
python3 -m ai_haste -c configs/test/test_40x.json
```

for inference on 40x and

```
python3 -m ai_haste -c configs/test/test_60x.json
```

for inference on 60x.

8 Future Directions

A benefit with our nuclei prediction network is that we obtain both the predicted nuclear channel and the segmentation prediction simultaneously. Having the classification output (i.e the segmentation) we can also obtain the certainty in the prediction based on conformal prediction [18]. This has the advantage that we can make the downstream analysis only in the parts of the image where we are most confident. Another way of using this is that the models usually have a high confidence in the center of the nuclei and reduced towards the edges. Placing a high threshold on the confidence output will thus give us the location of the cells we are most confident in and which can later be used as seeds for a watershed based segmentation. Due to time restrictions for the competition we did not apply this uncertainty quantification, but leave it as a direction for future work.

References

[1] Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. Albumentations: Fast and flexible image augmentations. *Information 11*, 2 (2020).

[2] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]* (June 2014). arXiv: 1406.2661.

[3] Gu, Z., Niu, L., Zhao, H., and Zhang, L. Hard Pixel Mining for Depth Privileged Semantic Segmentation. *arXiv:1906.11437 [cs]* (Mar. 2020). arXiv: 1906.11437.

[4] Guan, S., Khan, A. A., Sikdar, S., and Chitnis, P. V. Fully dense unet for 2-d sparse photoacoustic tomography artifact removal. *IEEE Journal of Biomedical and Health Informatics 24*, 2 (2020), 568–576.

[5] Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks, 2018.

[6] Isola, P., Zhu, J., Zhou, T., and Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pp. 5967–5976. ISSN: 1063-6919.

[7] Jolicoeur-Martineau, A. The relativistic discriminator: a key element missing from standard GAN. *arXiv:1807.00734 [cs, stat]* (Sept. 2018). arXiv: 1807.00734.

[8] Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., and Bengio, Y. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. 1175–1183. ISSN: 2160-7516.

[9] Khvedchenya, E. Pytorch toolbelt. https://github.com/BloodAxe/pytorch-toolbelt, 2019.

[10] Kupyn, O., Martyniuk, T., Wu, J., and Wang, Z. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. *CoRR abs/1908.03826* (2019).

[11] Loshchilov, I., and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).

[12] Loshchilov, I., and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[13] Pielawski, N. npielawski/pytorch_tiramisu: Better tiramisu 1.0, feb 2020.

[14] Pielawski, N., Hu, J., Strömblad, S., and Wählby, C. In silico prediction of cell traction forces. In *2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)* (2020), pp. 877–881.

[15] Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention MICCAI 2015* (Cham, 2015), N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 234–241.

[16] Sugawara, Y., Shiota, S., and Kiya, H. Super-resolution using convolutional neural networks without any checkerboard artifacts, 2018.

[17] Vapnik, V., and Izmailov, R. Learning Using Privileged Information: Similarity Control and Knowledge Transfer. *Journal of Machine Learning Research 16*, 61 (2015), 2023–2049.

[18] Wieslander, H., Harrison, P. J., Skogberg, G., Jackson, S., Friden, M., Karlsson, J., Spjuth, O., and Wahlby, C. Deep learning and conformal prediction for hierarchical analysis of large-scale whole-slide tissue images. *IEEE Journal of Biomedical and Health Informatics* (2020), 1–1.

[19] Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network, 2015.