



ADIPOCYTE CELL IMAGING CHALLENGE

# Team: The Bug Hunters

*Authors:*
Andrés Martínez Mora, Blanca Cabrera Gil,
Osheen Sharma, Blanca Bastardés Climent

November 15, 2020

# Contents

# Chapter 1

# High level description

The proposed solution to the Adipocyte Cell Imaging Challenge by the Bug-Hunters Team consists in the use of an Attention U-Net with Residual Connections that is trained on the Maximum Intensity Projection (MIP) images generated from the raw brightfield images. Different models were trained on different magnifications and against different fluorescence channels, using the Mean Squared Error (MSE) loss as loss function and the Structural Similarity Index Measurement (SSIM) and the Peak Signal-to-Noise Ratio (PSNR) as metrics. The diagram below shows a summarized description of the above:

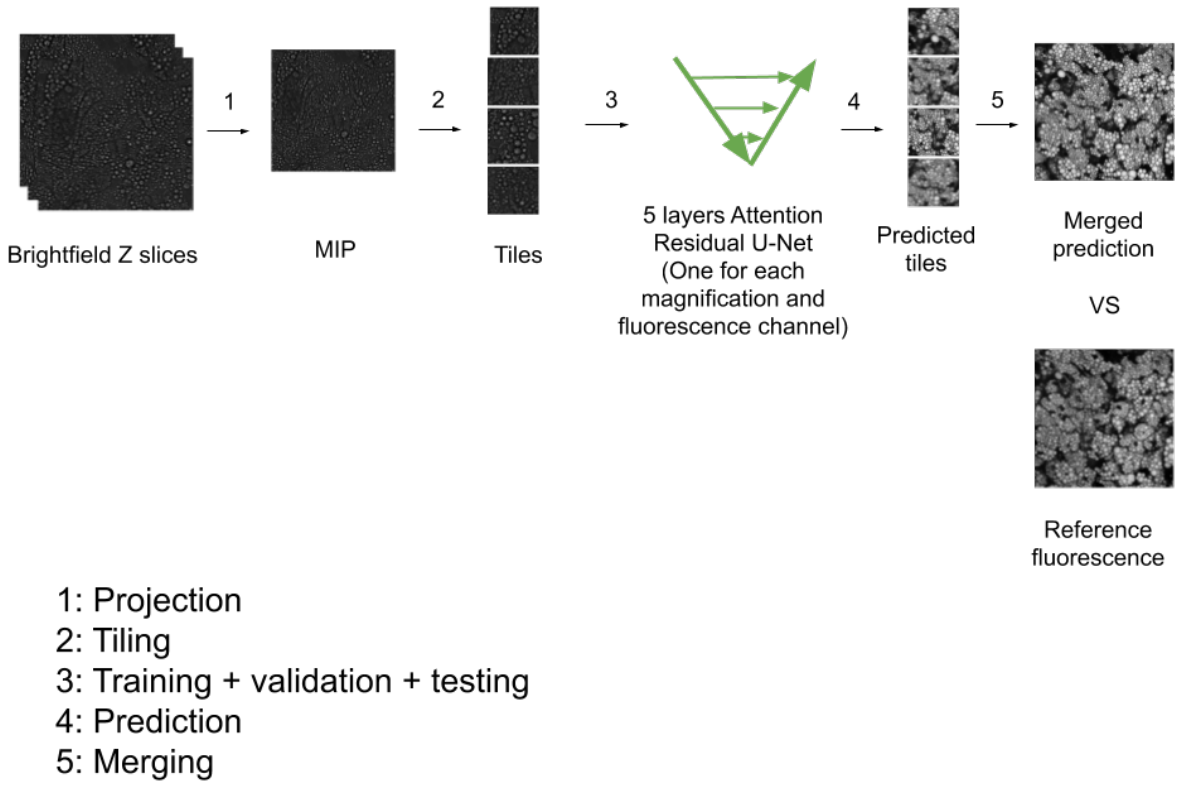



Figure 1.1: High level description of the solution from Team Bug-Hunters

# Chapter 2

# Data Processing

## 2.1 Data Processing Parameters

The given brightfield images were inputted into a preprocessing pipeline that enabled a robust analysis during later steps. The following steps were included:

1. Reading the raw image files, while distinguishing between brightfield images and the different fluorescent channels, according to their *A0x* and *C0x* tags.
2. Creating Maximum Intensity Projection images (MIPs) from the seven Z-stack brightfield images and saving them in a separate folder. It was preferred to do this rather than preprocessing *on the fly* to allow for a faster data loading. MIPs were cropped with 2048x2048 size that enabled a power of two processing. This step allowed the formulation of a 2D problem, which for a Deep Learning Architecture is easier to solve. Data variation along the Z axis was not so large, so a projection image could retain a large percentage of the original 3D information.
3. Data splitting into train, validation, and test. A random 10% was assigned to the test set, while from the remaining dataset, 80% was assigned as train and 20% to validation.
4. Converting the high resolution images into small patches by tiling the input images (MIP brightfield channel) and corresponding targets (fluorescent channel) for both train, validation, and test sets.
5. Creating NumPy array from the tiled inputs and targets before training the model.

Steps 1 and 2 were coded in the script **preprocessing.py** of the team's repository, step 3 was carried out in the beginning of the script **main.py**, and steps 4 and 5 were completed in the script **merge_and_tile.py** (called from **main.py**).

# Chapter 3

# Model Architecture

Given the nature of the image translation problem, a generative approach was selected to predict the fluorescence channels from bright-field images. The skeleton of the implemented architecture is a UNet model [1] as it is proved to provide state-of-the-art results for image-to-image translation for life science images [2]. Further, residual connections have been added per convolutional block to ease information propagation throughout the network and ease the training [3]. Additionally, attention gates were added to the architecture as they provide the network with the ability to focus on target structures of varying shapes and sizes [4], easing the network detection of the nuclei, lipocytes, and cell matrix. Therefore, we will refer to the implemented model as AttentionResUnet.

The code for this architecture is in the script **model.py**.

The diagrams for the architectures are shown in figures 3.1 and 3.2

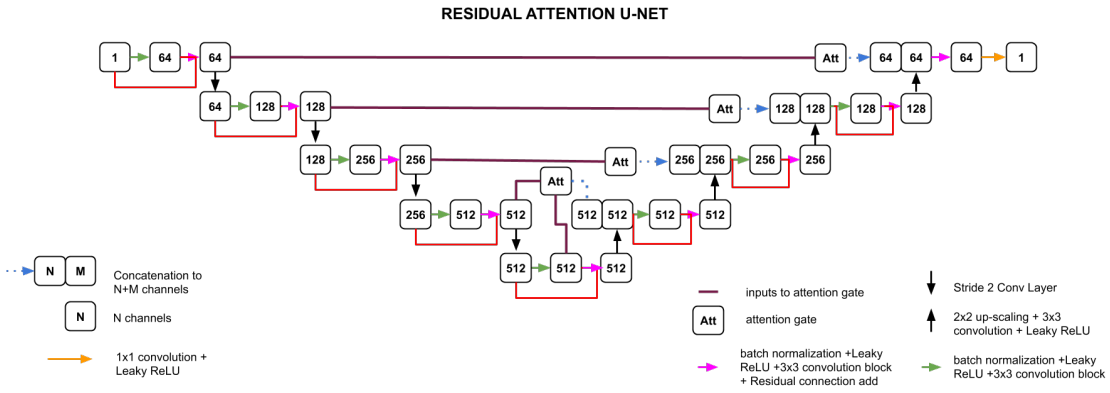



Figure 3.1: AttentionResUNet, the implemented architecture by the Team Bug-Hunters

## 3.1 Model Parameters

The neural network architecture has five layers of depth with 64, 128, 256, 512 and 512 feature maps respectively. The input size is defined as the size of the tiles divided by a down-sampling factor in case the training images need to be down-sampled, and the number of input channels. In the case of this challenge the input shape was [512,512,1] in

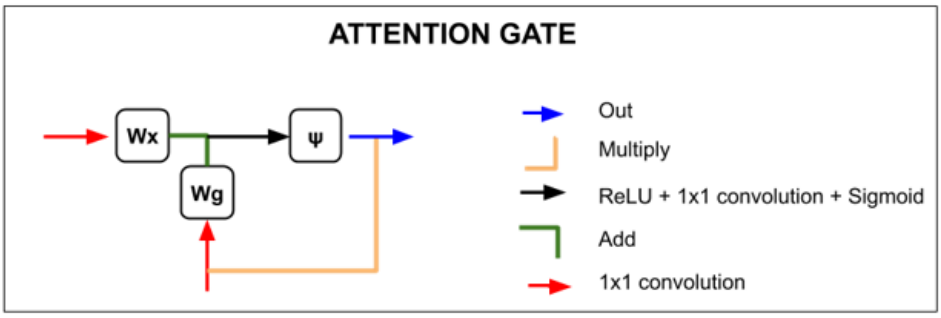



Figure 3.2: Detailed diagram of an attention gate, input from the left comes from the downsampling section, and input from below comes from the upsampling section

channel last format. The input has only one channel as we are only feeding the MIPs of the brightfield images. Additionally, the network activation functions were set to Leaky ReLU to avoid the dead neuron problem.

## 3.2 Model Size

The total amount of network parameters scales up to 34,953,797, having 34,931,141 trainable parameters and 22,656 non-trainable ones. Summing up to a maximum of 420 MB.

## 3.3 Model Output

The model outputs a predicted tile with shape corresponding to the inputted tiles, in our case [512,512,1]. As the images provided by AstraZeneca have been tiled and fed to the network, the predicted tiles are merged back in groups of 16 tiles to rebuild the original image. An example of predicted tile can be found in the Figure 3.3, together with an example of an image composed by merging the predicted tiles in Figure 3.4.

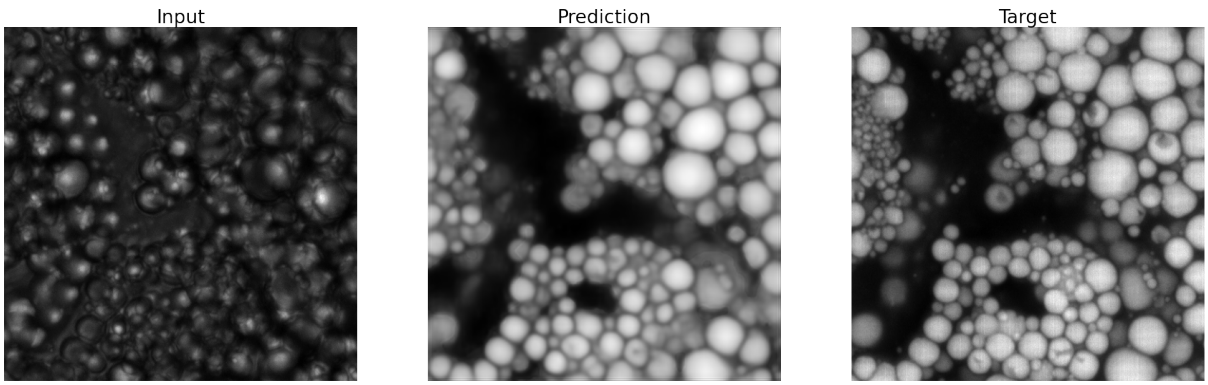



Figure 3.3: Comparison between the input, model prediction, and target for a 512x512 image tile of channel 2 and magnification 60x.

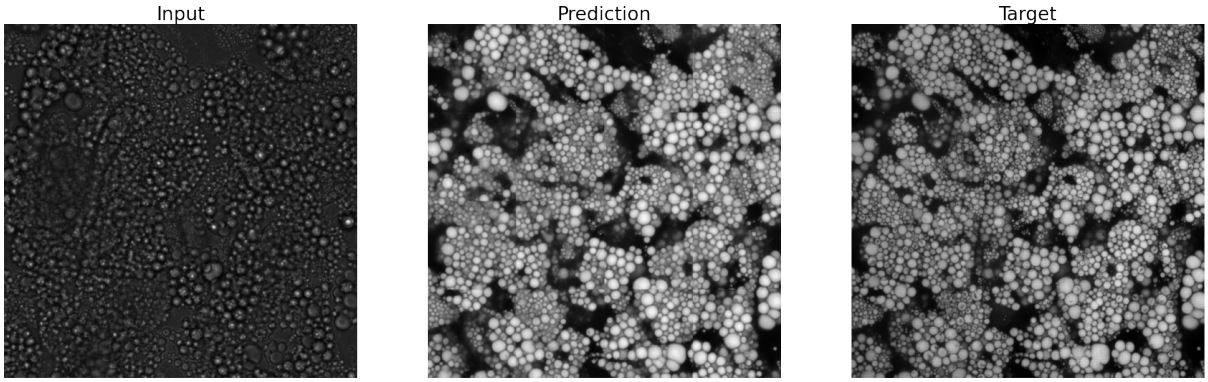



Figure 3.4: Comparison between the input, model prediction, and target for the merged image of including the tile showed in figure of channel 2 and magnification 60x.

# Chapter 4

# Model Training

As the challenge proposed a very complex problem having to solve an image-to-image translation of brightfield images of different magnifications to the corresponding images of three different fluorescence channels, a divide-and-conquer approach was selected. Therefore, nine different models have been train, which is a different model per magnification and fluorescence channel. In this way, each model becomes quite specialized in a magnification and in a fluorescent setting, being easier for it to learn, at the drawback of having to load more models in the inference phase.

## 4.1 Training Scheme Overview

The training scheme is divided in the following main sections: Data Augmentation, Model Prediction, and Mean Squared Error (MSE) Loss computation as observed in Figure 4.1.

The data augmentations parameters consisted of horizontal and vertical flips, and in occasions rotations of a maximum of 20 degrees where added. Moreover, the tile samples were shuffled at each epoch to ensure an unequal feeding order while training. MSE was selected as loss function as it proved effective for optimizing the architecture and it was one of the evaluation metrics. Additionally, Structural Similarity Index Measure (SSIM) and Peak Signal-to-Noise Ratio (PSNR) were selected as accuracy metrics to monitor model learning. The implementation of these metrics can be found in *utils.py*

A learning rate schedule was applied while training using exponential decay schedule. Therefore, the learning rate decreases exponentially from the 10th epoch onwards. The method implementing this is called *scheduler* and it is located in *main.py*

The hyperparameters tuned for this model were the learning rate and tile downsampling factor. These final values were selected after performing a grid search and selecting the ones that provided a better validation score.

Finally, given the amount of memory space and time needed to train each network an early stopping callback was also added. This was monitoring the evolution of the validation loss and had a patience of 20 epochs.

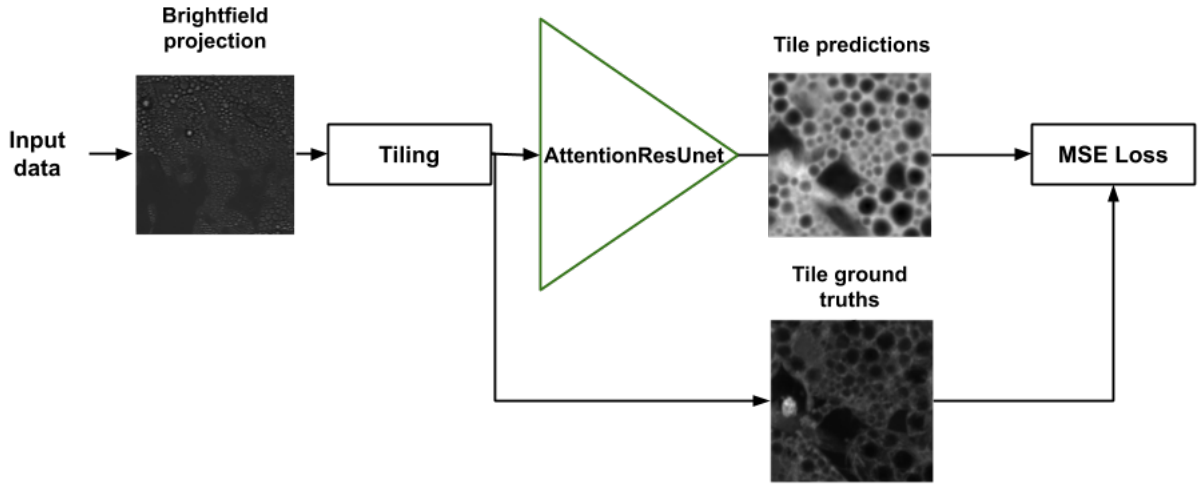



Figure 4.1: Training diagram. The input MIPs of the brightfield image stacks as well as the corresponding fluorescence targets are tiled, and then fed to the network. Mean Squared Error (MSE) is used as the loss function to compare the similarity between the predicted and target fluorescence tile.

## 4.2 Pre-Training

The models trained for this challenge did not use any pre-training schedule or used weights from any other model. Yet, in the provided code, if a previous checkpoint is found for the same magnification level and fluorescence image, then the saved model is loaded and the training continues from the last saved checkpoint.

## 4.3 Training Parameter

The final hyperparameter configuration can be found in Table 4.1

| **Hyper-parameters** | **Settings** |
|---|---|
| learning rate | 1e-4, with exponential decay after 10 |
| epochs 100 with early stopper | |
| optimizer | Adam, $\beta 1 = 0.5, \beta 2 = 0.999$ |
| tile resolution | 512X512 |
| batch normalization | True |
| batch size | 1 |
| augmentations | vertical flip, horizontal flip, rotation = 20° |
| depth of U-Net | 4 downsampling, 1 bottleneck, 4 upsampling convolutional layers |
| activation function | ReLU, LeakyReLU |
| downsample factor | 1 |
| batch size | 1 |

Table 4.1: Hyperparameter setting

## 4.4 Consistency in training results

The different training curves show that all models experienced similar training trends. Firstly, all models seem to have converged around 40 epochs given that the validation loss becomes flat at this point for further epochs, and is therefore easy to reproduce the training and validation results. In all augmentations the models trained to predict fluorescence channel 1 have a higher MSE, and lower levels of SSIM and PSNR regardless of the magnification. The reason for this behaviour resides in the fact that channel 1 fluorescence images have a higher degree of background and cannot be directly extracted from purely structural features of the input image. Yet, this difficulty to predict cell nuclei becomes smaller the higher the magnification parameter. For channel 2, the validation curves of SSIM, PSNR, and MSE for magnifications 20x and 40x look very similar, depicting a great optimization of the model. Nevertheless, its curves for magnification 60x show a slight learning from the network compared to other channels. Finally, channel 3 is the channel that shows a greater optimization for all 3 augmentation levels, having higher SSIM and PSNR final values, and low MSE. The training plots for 20x, 40x, and 60x magnifications can be found in Figures 4.2, 4.3, and 4.4 respectively.

## 4.5 Training time

The model was trained in the NVIDIA A100-SXM4-40GB GPU provided by the challenge organizers. The input images had a consistent training size after the pre-processing step, the trainings had a duration of 300s per epoch, which sums up to 3 hours and 20 minutes to train for 40 epochs.

The accumulated inference time for the different magnifications can be found in 4.2.

Table 4.2: Time for final testing implementation

| **Magnification** | **Time (in secs)** |
|---|---|
| 20x | 138.9822 |
| 40x | 192.8459 |
| 60x | 284.3814 |
| **Total** | **616.2094** |

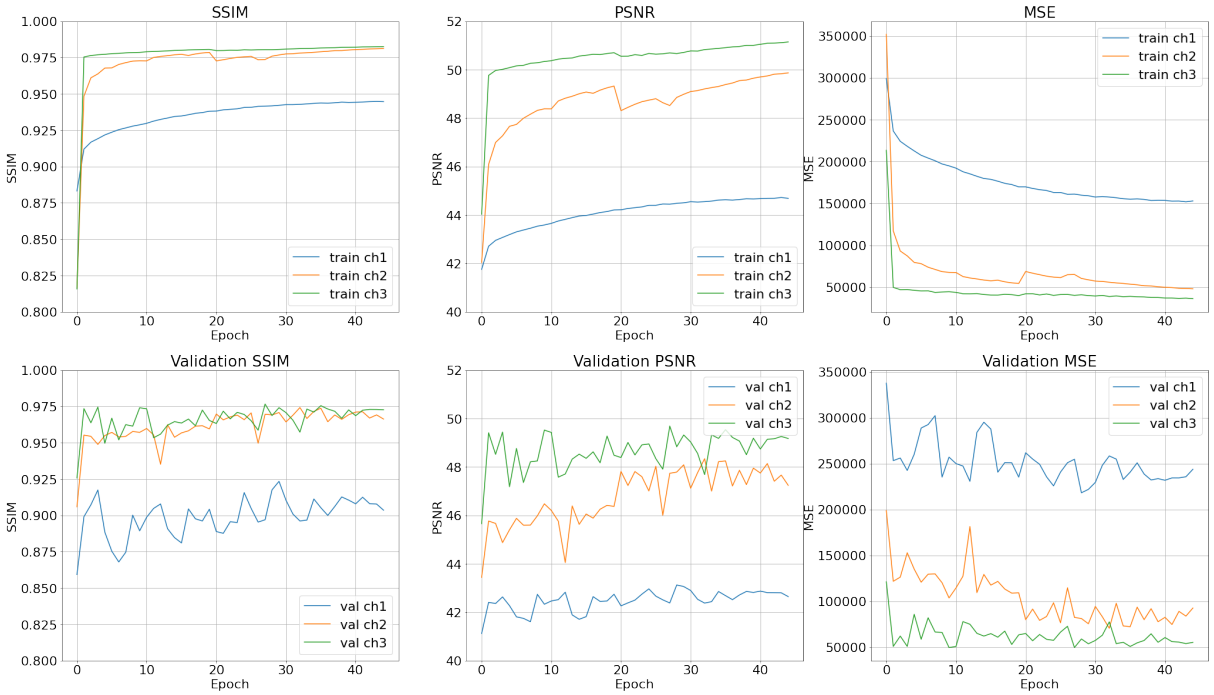



Figure 4.2: Learning curves for the 3 channels of the models trained on images of 20x magnification.

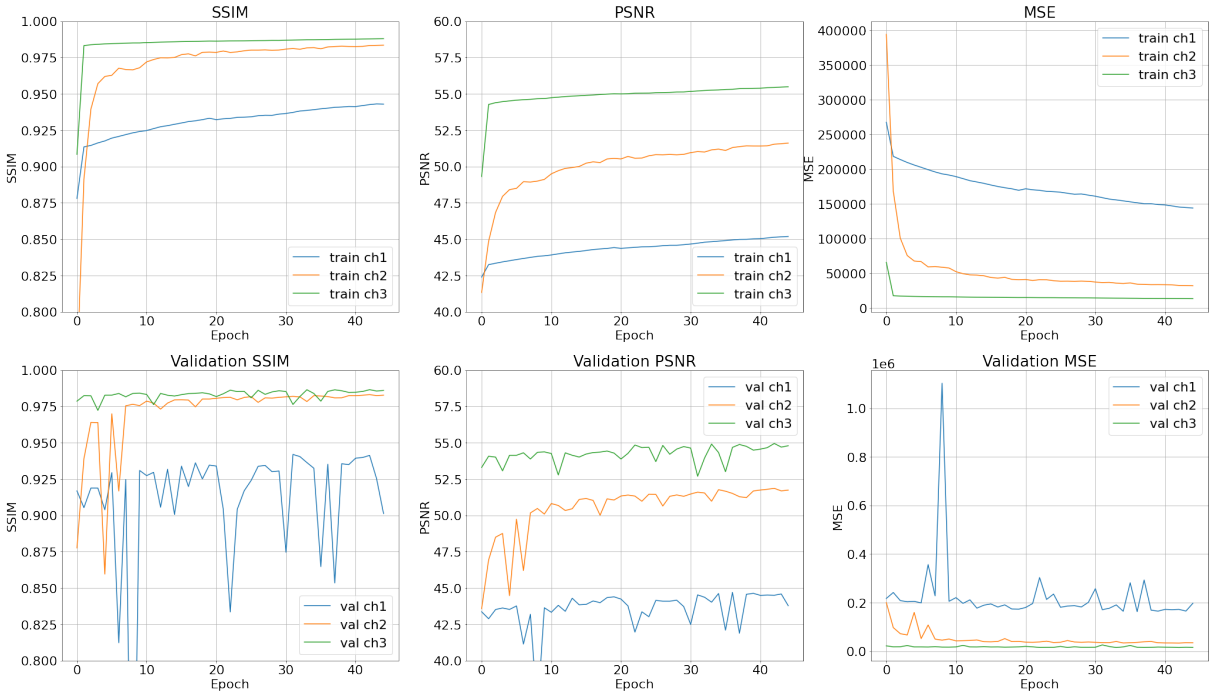



Figure 4.3: Learning curves for the 3 channels of the models trained on images of 40x maginfication.

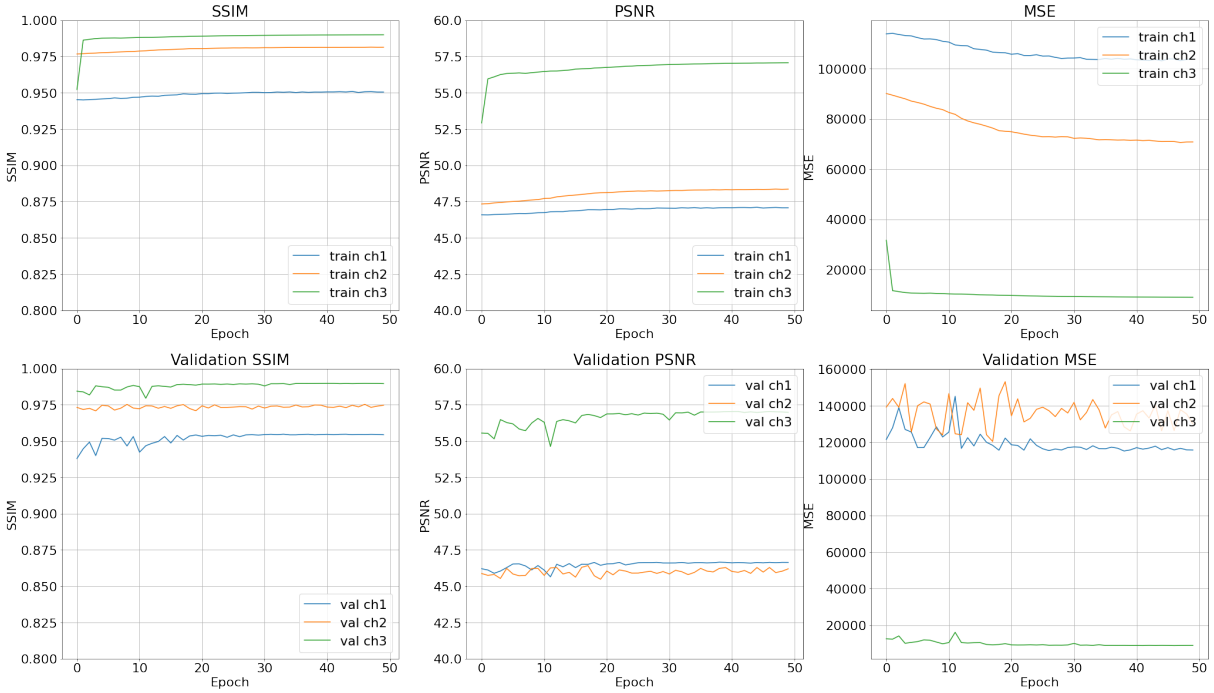



Figure 4.4: Learning curves for the 3 channels of the models trained on images of 60x magnification.

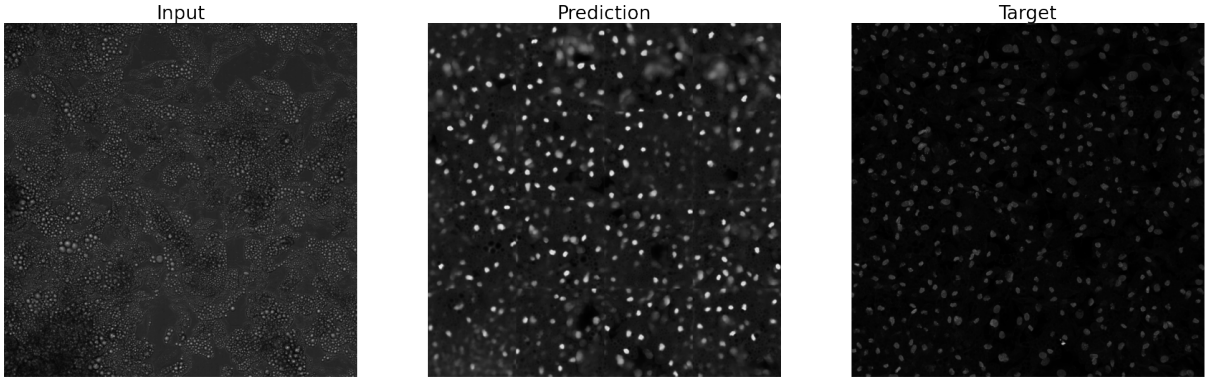
# Chapter 5

# Analysis of Model Performance

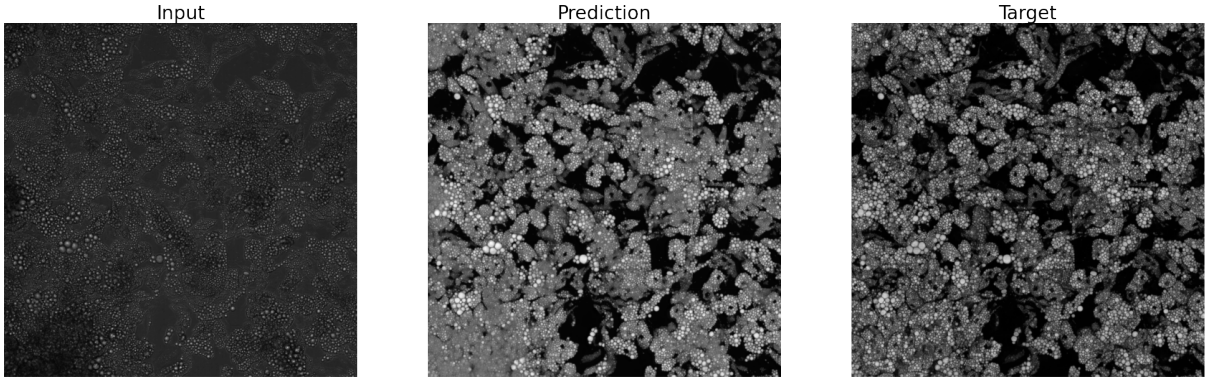
Once the models were trained, they were tested with a small set of images reserved to test its generalization capability. To make sure the model output could be linked to the evaluation pipe line the predicted tiles were merged back to reconstruct the original image and resized to match its initial shape. Additionally, for testing purposes, the MSE, SSIM and PSNR of the merged predictions has been calculated. These metrics obtained for each model can be found in Table 5.1. In the same way, it can be observed that the test results are within the range of values obtained for training and validation. Examples of resulting images per magnification can be found in Figures 5.1, 5.2, and 5.3.

| **Magnification** | **Channel** | **MSE** | **SSIM** | **PSNR** |
|---|---|---|---|---|
| 20 | 1 | 209804.21 | 0.92 | 43.26 |
| 20 | 2 | 62144.125 | 0.97 | 48.73 |
| 20 | 3 | 49114.16 | 0.97 | 49.50 |
| 40 | 1 | 178709.90 | 0.93 | 44.18 |
| 40 | 2 | 34349.26 | 0.98 | 51.37 |
| 40 | 3 | 14157.81 | 0.98 | 55.12 |
| 60 | 1 | 247270.67 | 0.91 | 42.69 |
| 60 | 2 | 334827.15 | 0.81 | 41.41 |
| 60 | 3 | 10384.38 | 0.98 | 56.50 |

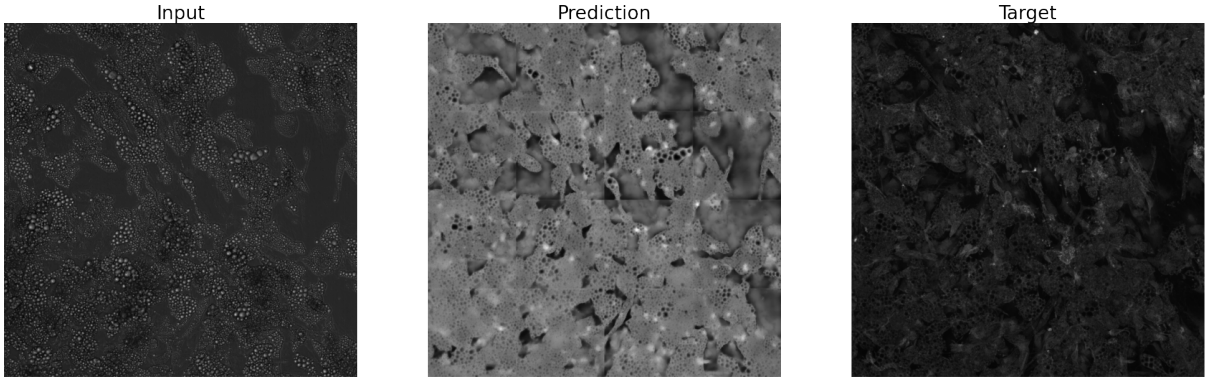
Table 5.1: Comparison of test metrics obtained per each model




(a) Channel 1.

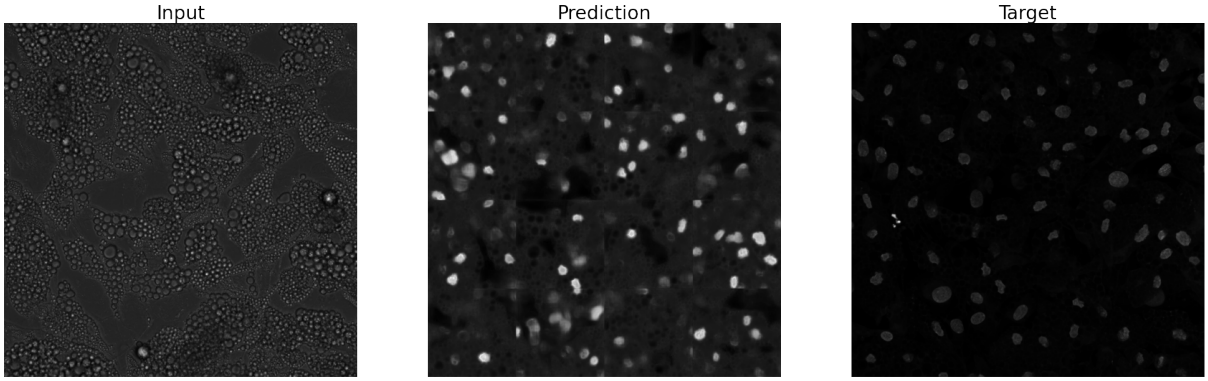



(b) Channel 2.

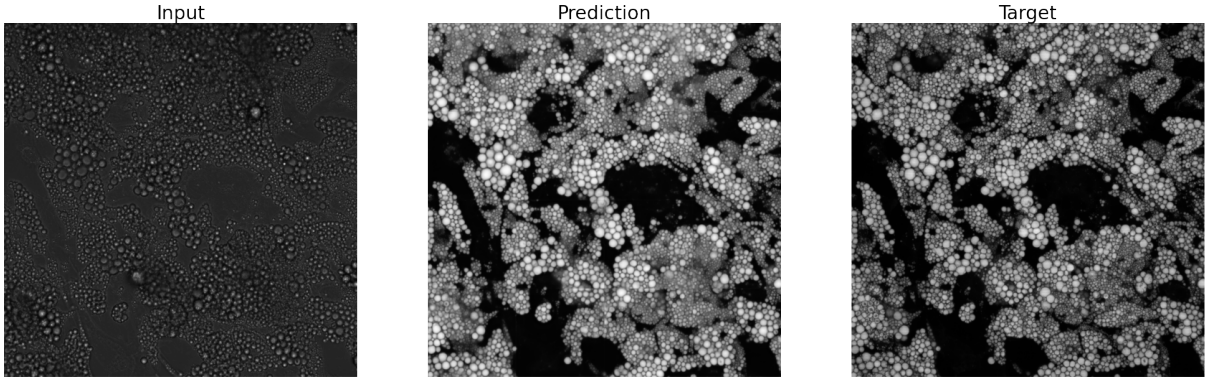



(c) Channel 3.

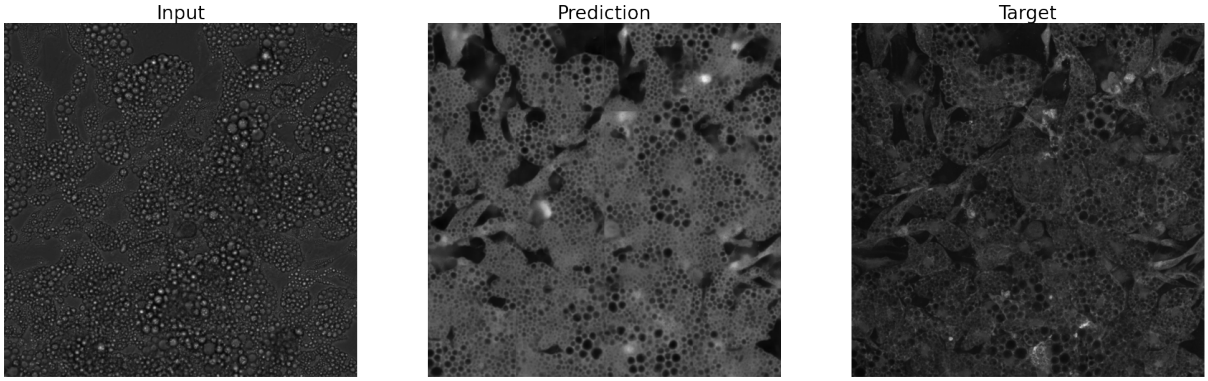
Figure 5.1: Prediction-Target comparison for 20x test images.




(a) Channel 1.

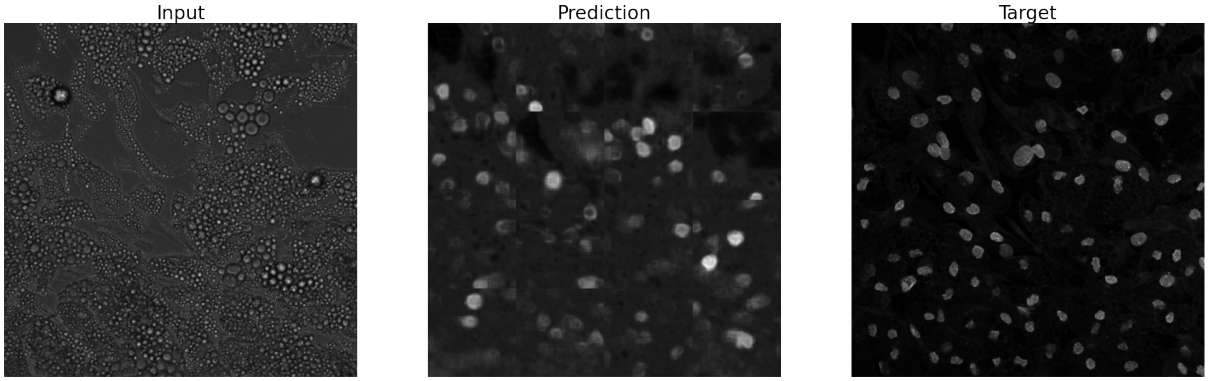



(b) Channel 2.

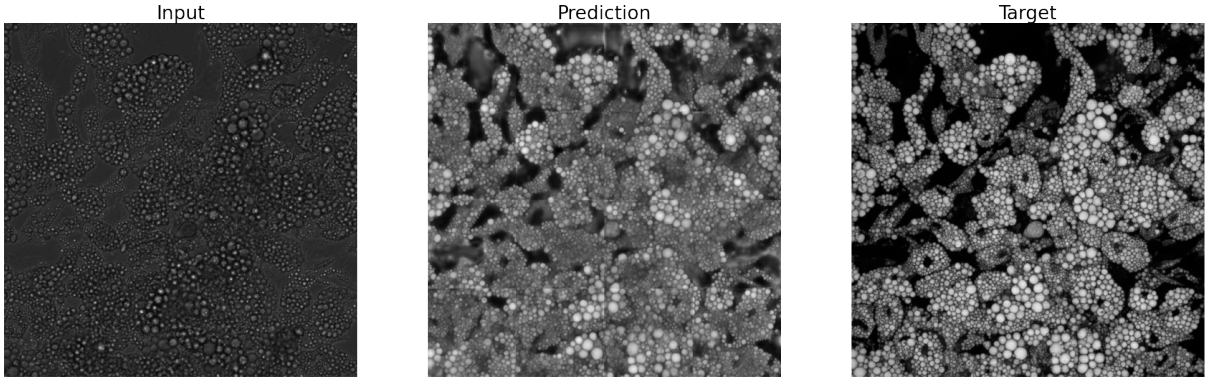



(c) Channel 3.

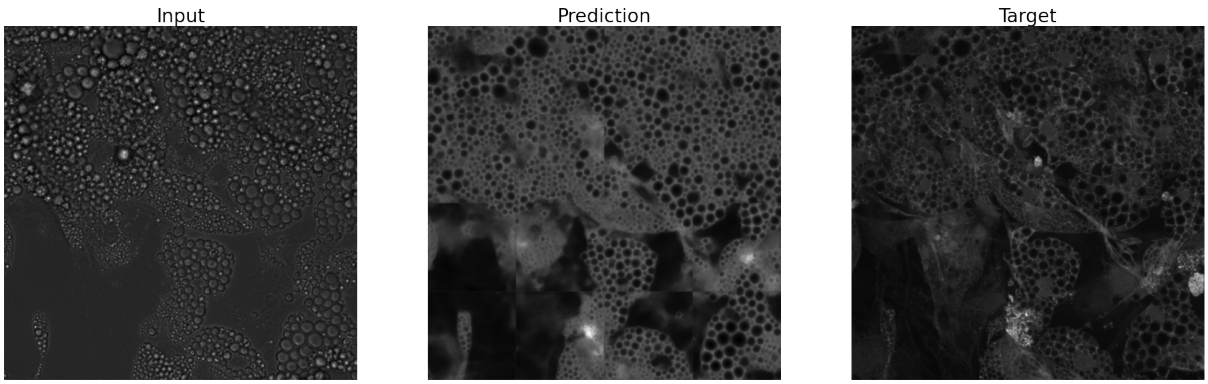
Figure 5.2: Prediction-Target comparison for 40x test images.




(a) Channel 1.




(b) Channel 2.




(c) Channel 3.

Figure 5.3: Prediction-Target comparison for 60x test images.

# Chapter 6

# Model Execution End-to-End

The provided code for the hackathon is composed by the following files. To train the proposed neural network from scratch for three different magnification levels and fluorescent channels, the **preprocessing.py** python file needs to be executed which arranges the dataset in the desired format and then only the **main.py** script should be run which calls all the required definitions.

- **preprocessing.py**: The file consist of cropping definition which is designed to crop the input images and targets in a square shape of 2048x2048. Additionally, code to access the dataset is written which takes the brightfield images and performs the MIPs. Further the MIPs and the corresponding fluorescent images are saved in structured folders.
- **data_loading.py**: Once the dataset is saved in desired format the next step is to load the preprocessed data, create tiles of size 512x512 for each input image and its corresponding target.
- **merge_and_tile.py**: The script takes numpy array as input and creates tiles. To evaluate the models performance on the whole resolution image, merging code is also implemented which merges the arrays of the tiles back into whole resolution image.
- **utils.py** The file consists of metrics, SSIM and PSNR which were used to evaluate the model performance.
- **model.py**: The Residual U-Net model with Attention Connections is implemented in this python file which takes the tiles from MIPs in form of array along with corresponding targets.
- **params.py**: The parameter list is prepared to carry out the preprocessing and training tasks. Along with the different parameter settings, the file also consists of variables *fluor_ch* and *magnification* which are set as 1 and 20 respectively by default. To train the network for other channels and magnification, only these two variables need to be altered without making any changes in the **main.py** script. Where *fluor_ch* set as 1 corresponds to Nuclei; 2 and 3 for Lipids and Cytoplasm, respectively. Similarly *magnification* set as 20 corresponds to 20x and 40, 60 corresponds to 40x and 60x, respectively.

- **main.py**: To train the network from scratch, only this script needs to be executed which calls all the definitions and gives the prediction on the merged tiles. The trained model is saved in hdf5 format for each channel and magnification level and the checkpoints are also saved in the script. If the user wants to train the network from saved checkpoints then the saved models should be located in a directory named as *training_ checkpoints*, and the checkpoint should have a name of the format: *checkpoint_ attUNet{magnification number}_fluor{number of fluorescence channel}*.

- **test_TeamBugHunters.py**: this function performs the inference of our process from a given input folder. The library *getopt* was used to allow the user to input a folder where images for testing are located. Optionally, the user can provide an output folder name for saving the images there. The input to be provided in the terminal is:

  ```
  python3 test\_TeamBugHunters.py -i input\_folder -o output\_folder
  ```

  The code should be in the same folder as **utils.py**, **params.py**, and **merge_and_tile.py**. All the models saved as .hdf5 should also be in the same folder as this script and the other scripts mentioned.

  The code examines all the .tif files under the input folder, checks the magnification each file belongs to, and sorts all the .tif filepaths for magnification (getting only brightfield images with the tags *A04* and *C04*). The image stack files are recognized by the beginning of the filename up to *A04*. When all files from the same magnification and stack are identified, the MIPs are obtained, being down-sampled to a size of 2048x2048. Then, the MIPs are tiled with 512x512 size and inputted to the models. All images from the same magnification are processed at once, having to load only the models that work with that magnification. The predictions are then generated and saved in a different subfolder for each magnification as *uint16 datatype*.

# Bibliography


[1] Brox T Ronneberger O. Fischer P. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. MICCAI 2015.* Lecture Notes in Computer Science 9351 (May 2015). DOI: 10.1007/978-3-319-24574-4_28 (page 3).

[2] Chawin Ounkomol, Sharmishtaa Seshamani, Mary M. Maleckar, Forrest Collman, and Gregory R. Johnson. "Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy". In: *Nat. Methods* 15.11 (2018), pp. 917–920. ISSN: 15487105. DOI: 10.1038/s41592-018-0111-2. URL: http://dx.doi.org/10.1038/s41592-018-0111-2 (page 3).

[3] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. "Road Extraction by Deep Residual U-Net". In: *IEEE Geoscience and Remote Sensing Letters* 15.5 (May 2018), pp. 749–753. ISSN: 1558-0571. DOI: 10.1109/lgrs.2018.2802944. URL: http://dx.doi.org/10.1109/LGRS.2018.2802944 (page 3).

[4] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. *Attention U-Net: Learning Where to Look for the Pancreas.* 2018. arXiv: 1804.03999 [cs.CV] (page 3).