Application of Variational Autoencoders in Image-Based Analysis of Cellular Response Profiles

Jesús Muñoz Alloza
Màster universitari en Bioinformàtica i Bioestadística

Area 5 – Machine Learning

Consultors: Ferràn Reverter Comes & Esteban Vegas Lozano
Prof. Responsable: Alexandre Sánchez Pla

22/06/2020

FITXA DEL TREBALL FINAL

Títol del treball:	*Application of Variational Autoencoders in image-based analysis of cellular response profiles*
Nom de l'autor:	*Jesús Muñoz Alloza*
Nom del consultor/a:	*Ferràn Reverter Comes*
	Esteban Vegas Lozano
Nom del PRA:	*Alexandre Sánchez Pla*
Data de lliurament (mm/aaaa):	*06/2020*
Titulació o programa:	*Màster universitari en Bioinformàtica i Bioestadística*
Àrea del Treball Final:	*Machine Learning*
Idioma del treball:	*English*
Paraules clau	*Variational Autoencoders, Deep Learning, Generative Modelling*

Abstract:

Cell images reconstruction from a subset of the MCF7 image repository is the primary goal of this work. It is implemented through a variational autoencoder: a generative, unsupervised learning paradigm whose architecture consists of an encoder that reduces the dimensionality of the input space by obtaining a distribution over the latent space and a decoder, that rebuilds the inputs from the encoding.

This work is completed with the description of the activities associated with the primary goal, like image segmentation and processing and infrastructure setup, the latest driven by automation tools and performed in a cloud environment.

to my wife, Pilar

and my son, Daniel

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and justification

The use of multiple measurements of cell morphology from images has been performed to characterize effects of drugs or the function of genes[4]. Anyway, the task of generating representations of single cells to summarize their properties in a *profile* to represent the population is still an open problem[5]. Training of a variational autoencoder (VAE) is a tool that can address this issue, modeling a latent variables *space* to characterize single cell imaging.

Other benefits of the use of generative models to produce synthetic images are to train discriminative models and also the simulation of biological processes using techniques like *latent variable walkthrough*.

Some other efforts have been made to adopt generative techniques to deal with cellular autoencoder reconstructions [5]. The novelty of the approach of this work is 1) to describe the whole process at technical level 2) to discuss the value settings, architectures and the mechanisms that influence the final result and 3) to test public cloud infrastructure as a technological framework for training models.

1.2 Project objectives

1. **GO1. Analysis of the theoretical framework:** in order to establish a theoretical background for the study, it is necessary to analyze prior works in this field and identify possible issues and limitations. The theoretical aspect must be also be worked to support decisions and to be described in the final document.

   1.1. Identify and review of related work.

   1.2. Study of already implemented VAE architectures.

2. **GO2. Design and implement a VAE:** the core of the project is the implementation of a VAE. Technical aspects include image treatment,

tools integration and model generation. Conclusions must be supported by a measurable aspect beyond subjective similarity between images.

2.1. Cell segmentation

2.2. Arrange cellular images by colour/channel, image enhance.

2.3. Image choosing based on image cell size and group control ownership.

2.4. Adjust images to a common size.

2.5. Propose and justify model decisions: architecture, neural network deepness and latent variables space dimension.

2.6. Implement a variational autoencoder based on the design.

2.7. Define or use an already defined metric to compare the generated data with the original data. Analysis of the results.

3. **GO3. Study the factors that determine VAEs performance:** the design of a second VAE based on the first one has to confirm or challenge theoretical assumptions. The use of the already defined metrics will allow to compare results among this, the first one and the raw images.

3.1. Based on the theoretical analysis and the implementation, identify which factors may improve VAE model performance.

3.2. Propose and implement a new model and check results with the first model and the original data.

1.3 Approach and working methodology

Both the scientific and the technology aspects of the project require different approaches. The **scientific aspect** is mainly related with the analysis of the theoretical framework, the identification of the factors that are going to improve the VAE model performance at second stage and the final conclusions of the project. This aspect has to be worked in a linear fashion, supporting the technical decisions whenever are to be taken.

Regarding **technology aspect**, the use of ML standard frameworks like **Keras/Tensorflow,** tools like **Jupyter** notebook and programming languages like **Python** reinforce the idea of consistence at technology point of view. Other tools like **CellProfiler**[6] are a common choice to perform tasks like cell segmentation. The image treatment at file level is done using **Python** and its supporting image libraries (Python Imaging Library by Fredrik Lundh, PIL).

To avoid resource limitations, the use of the **public cloud infrastructure** is an approach that is worth to be considered. The pay-per-use price

modeling and the discounts for idle computational power[1] makes *apparently* affordable the use of a ML instance for Deep Learning training. Moreover, there are some virtual machines in **Amazon Web Services** with preinstalled frameworks and tools (Keras/Tensorflow, Jupyter notebook) that perfectly fit the needs of the project makes AWS the chosen infrastructure provider.

The use of a tool[2] to describe the **infrastructure components as a code** allows 1) to avoid delays between computing components creation and model training, 2) to agile the second model training described in Project objectives section and 3) to facilitate the study to be reproducible.

Considering the aspects described above, and taking into account uncertainly over model training performance, it is more suitable to assume an iterative methodology with reference to technology aspects. Assumptions and objective accomplishments were reassessed at the end of the work phases.

1.4 Work plan

1.4.1 Tasks

1. **GO1. Analysis of the theoretical framework (20 h).**
    - Collect different material about the topic both at theoretical and implementation perspectives (10 h)
    - Study of already implemented VAE architectures. (10 h)
2. **GO2.1. Image processign (30 h).**
    - Cell segmentation (10 h)
    - Arrange cellular images by colour/channel, image enhance. (5 h)
    - Image choosing based on image cell size and group control ownership. (10 h)
    - Adjust images to a common size. (3 h)
    - Upload control group images to the cloud. (2 h)
3. **GO2.2. Design and implement the VAE (125 h).**
    - Propose a model design for the VAE. (20 h)
    - Set a pipeline to segment cellular images. (15 h)
    - Image treatment: check integrity, adjust size, enhance images (if necessary), and select control group images. (25 h)

[1] Amazon Web Services (AWS) spot instances: https://aws.amazon.com/ec2/spot/
[2] Terraform: https://www.terraform.io/

- Create a script using Terraform to generate the cloud environment. (20 h)
- Create Python code to implement the VAE. (20 h)
- Execute model and grab the results. (5 h)
- Define the metric to compare the results with the original data. Calculate over results. (20 h)

4. **GO3. Study the factors that determine VAEs performance (25 h)**
    - Identify parameters that can improve VAE performance and change model parameters. (5 h)
    - Execute the new model and grab the results. (5 h)
    - Calculate the new metric with the new results. Compare with the first model. (15 h)

Related with the PECs, the assessment (120 h) is:

- PEC0: Proposal (4 h)
- PEC1: Work plan (11 h)
- PEC2: Work Phase 1 (10 h)
- PEC3: Work Phase 2 (10 h)
- PEC4: Project report (50 h)
- PEC5: Presentation (20 h)
- PEC5: Public defense (15 h)

1.4.2 Milestones

Project milestones are:

Objective	**Milestone**	**Deadline**
GO1	Collect a set of references to support hypoteses	5-6 Mar
GO2	Enhanced segmented images persisted in a cloud environment	21-22 Mar
GO2	First VAE implemented	15-16 Apr
GO4	Second VAE implemented	10-11 May

1.5 Deliverables

The deliverables for this project are:

- Project report, including a budget report to evaluate methodology feasibility in other projects.
- CellProfiler pipelines used to segment MCF7 image set.
- Python code used for file and image processing, including source code to enhance/modify segmented images and code used to create RGB images from different image channels.
- Terraform scripts to ease environment creation and steps to setup (in Markdown).

Source code is in a public version control repository (GitHub). See **Appendix A** for more details.

1.6 Brief description of other sections

After a brief **introduction** to generative models and the **fundamentals of variational autoencoders**, loss function components for VAEs are depicted. Afterwards, basic elements of the Deep Networks that have been trained in this work are also described.

Materials and methods chapter describes, step-by-step, the process from segmentation to analysis, including the components of the cloud environment. A final section details the budget for each trained model analyzed.

Results chapter describes the outcomes of each model and makes a comparison among them. A sampling of one of the models is also performed, showing the images produced.

The **last chapter** depicts the final ideas and possible improvements to this work.

Chapter 2

Background

2.1 Introduction to generative models

The distinction between **discriminative modelling (DM)** and **generative modelling (GM)** is a major division in machine learning [7]. DM attempts to, given an observation $\mathbf{x}$ and a category y, estimate the probability of $p(y|\mathbf{x})$. The idea that relies under GM is to establish the probability of observing $p(\mathbf{x})$. Given a sample set $\{x_i\}$ with some attributes or characteristics like image pixels, the generative model *maps* the attributes of the training set to a *mainfold* or subspace of a more general space, enabling *representation learning*.

Historically, discriminative modeling has been the driving force in machine learning. The aplicability in multiple business fields as user segmentation or sentiment analysis and the ability to easily measure performance in discriminative models have boosted the development of DM methods.

Some studies [8, 9] have relied in discriminative methods using neural network analysis to monitor cellular responses. On the other hand, some efforts have been made to adopt generative techniques to deal with cellular autoencoder reconstructions [5].

2.2 Fundamentals of variational autoencoders

An autoencoder is a type of artificial neural network that consists of two steps or phases. In the first one, the observation $\mathbf{x}$ is reduced or **encoded** to a low dimensionality expression through a deep neural network. In the second phase, another neural network called **decoder** reconstructs the initial observation.

The standard Autoencoder, similarly to **principal component analysis (PCA)**, performs a dimensionality reduction of the features of the data, but PCA preconditions are not applicable in autoencoders: several basis can describe the same subspace, and orthogonality of the resultant principal

components is not required, so new features in the latent space (LS) can be dependent among them.

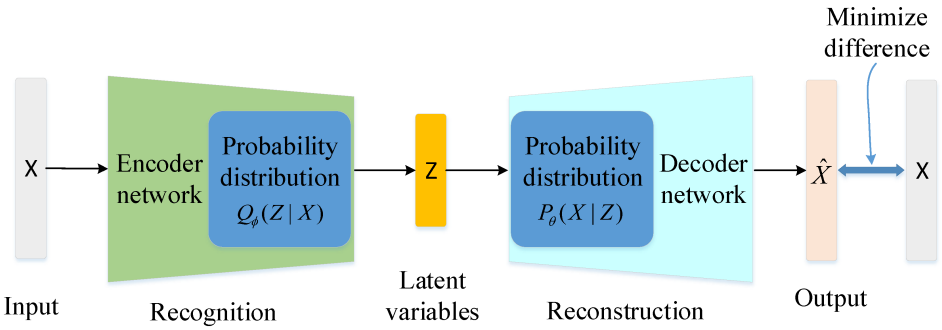

Figure 2.1: Variational autoencoder architecture [1].

A variational autoencoder is an autoencoder that, instead of mapping one element **x** to one point of the latent space, maps the element to a **multivariate distribution** (in general, a multivariate normal distribution). Thus, the LS becomes **continuous**, allowing sampling and interpolation. Furthermore, the points around the codification of **x** are likely to be similar to this element, so a sampling near this point is likely to be decoded as a well-formed image.

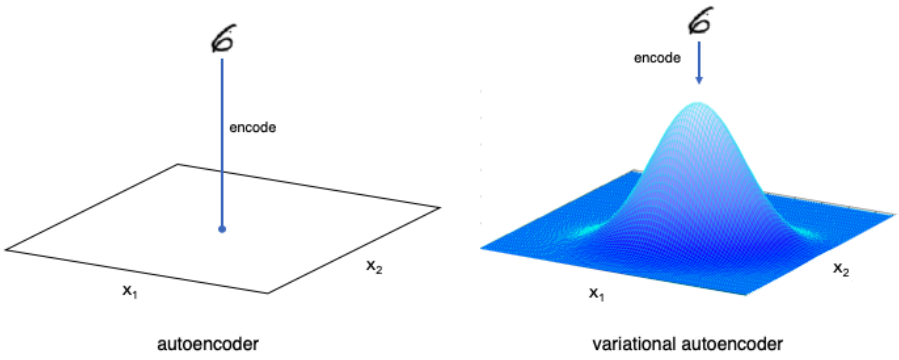

Figure 2.2: The difference between the encode in an autoencoder and a variational autoencoder [2].

Reconstruction and Regularization

The method to evaluate performance of the model is the loss function, that in variational autoencoder is composed by two terms:

- a *classic*, generative factor, which compares the input and the output of the model. In this case the least square errors (L2) function has

been implemented:

$$L2 = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

- Kullback–Leibler (KL) distance, that compares the latent vector distribution with a normal with $\vec{\mu} = 0$ and $\Sigma = I$; this term penalizes the variational autoencoder if latent vectors are not from the distribution. In its closed-form expression [10]:

$$\mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})] = -\frac{1}{2}\sum_{k=1}^{K}\{1 + \log\sigma_k^2 - \mu_k^2 - \sigma_k^2\}$$

The value of the loss function

$$lossfunction = L2 + KL[q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})]$$

is equalized via `R_LOSS_FACTOR` to balance the relation between $L2$ and KL during the training process to avoid a preeminence of one of these factors.

2.3 Deep learning layers

Convolution and convolutional layer

A **convolution** (Figure 2.3) is an mathematical operation on two functions (x and y) that produces a third function expressing how the shape of one is modified by the other. In equation form:

$$\mathrm{x(t)} * \mathrm{y(t)} = \int_{-\infty}^{\infty} x(\tau)y(t-\tau)d\tau$$

A convolutional neural network is a class of neural network that is mainly composed by convolutional layers. In convolutional neural network (CNN) terminology, the function x is referred to as the **input**, and the second argument y as the **kernel**. The objective of a kernel is to extract features from an input image. Thus, a kernel could, for example, enhance vertical lines in an image. In machine learning applications, a CNN learns the values of the kernel on its own during the training process.

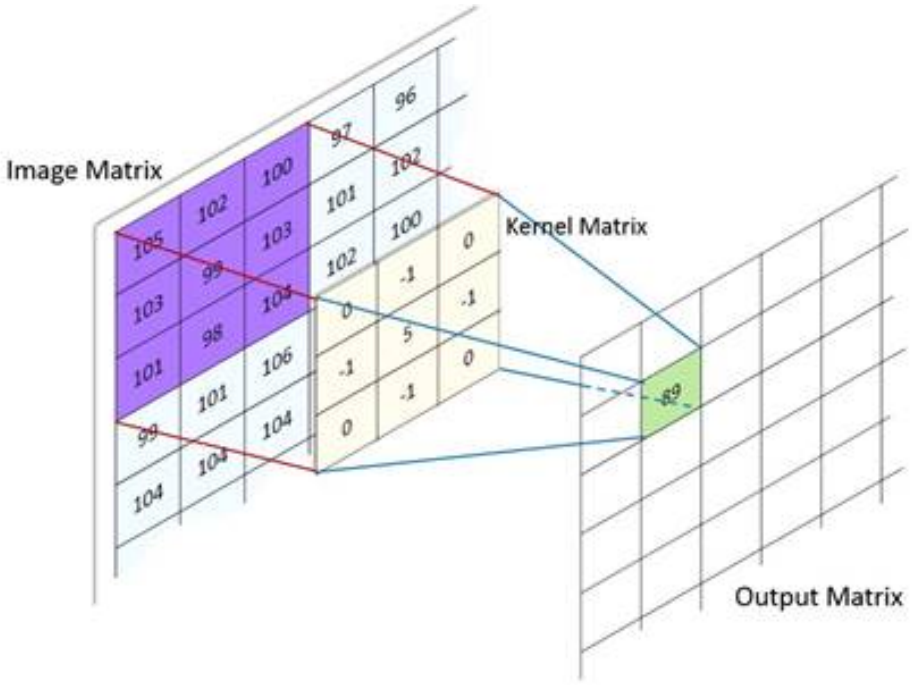

Figure 2.3: A kernel that implements a general 2D convolution operation [3]

Batch normalization

We define **internal covariate shift** as the change in the distribution of network activations due to the change in network parameters during training [22]. As backpropagation mechanism propagates the errors in the network, and because initial values in layers have been randomly assigned, the gradient in early layers can grow exponentially (overflow).

To improve stability of neural networks, batch normalization layer calculates the mean and standard deviation of its input layers and normalizes by subtracting the mean and dividing by the standard deviation.

LeakyReLU

An **activation layer** that uses leaky rectifier linear unit (LeakyReLU) (see Fig. 2.4), a function that implements a small, positive gradient when the unit is not active.

Dropout

A **dropout layer** (see Fig. 2.5) is a *regularization* technique to avoid *overfitting*. The idea is to avoid the dependency of an arbitrary group of neurons that remember the observations in the training set. Thus, in the training process some units are disconnected (the number is configurable) to avoid a

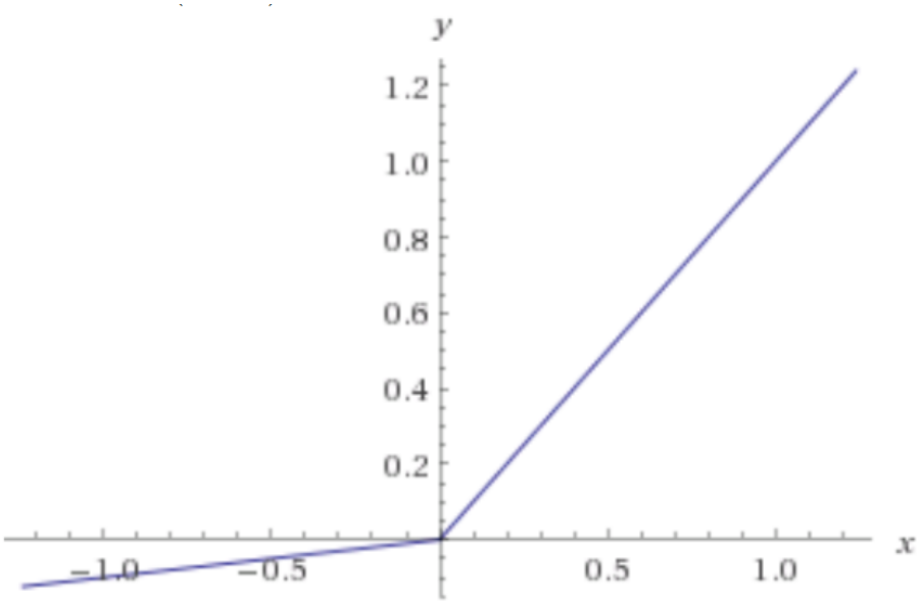

Figure 2.4: The Leaky ReLU function

dependency with the training set. Dropout layers are parametrized through a rate p that specifies the fraction of the input units to drop.

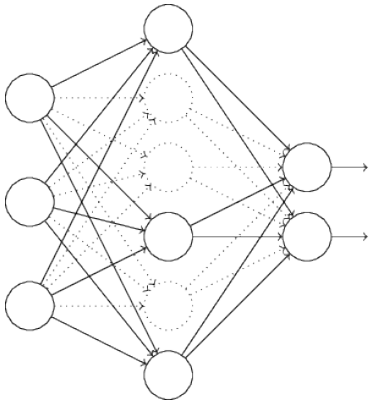

Figure 2.5: Dropout layer $p = 0.5$

Lambda

Maps an expression (function) as a layer object. Used to map the multivariate normal distribution expression results as the encoder of the output layer.

If an operation over data is not defined in predefined layers, it can be done in a lambda layer, where an operation over the elements of the vector

can be performed. Figure 2.6 illustrates this point, where the output of a flatten layer is branched into two layers (`mu` and `log_var`) whose elements are the parameters of the lambda function. In this case, the output of the lambda layer is a sampled point $\vec{z}$ from the latent space distribution defined by `mu` and `log_var`.

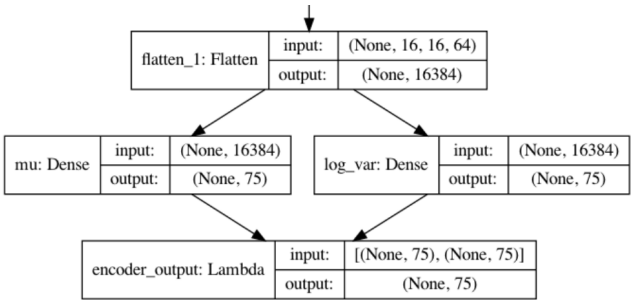

Figure 2.6: Lambda layer

Chapter 3

Materials and methods

Training a Variational Autoencoder based on the premises described in 1.1 and 1.2 requires a series of steps to successfully to **segment** the images involved to distinguish the cells involved. After these steps, a **post-processing** has to be done to check the size of the images and to **upload** this images to a cloud storage. This process is done only one time, making the images available for the models.

The images in the storage will be used in each **model training**, a process that will be controlled through the creation of a **cloud environment** and the execution of the programs that define model training.

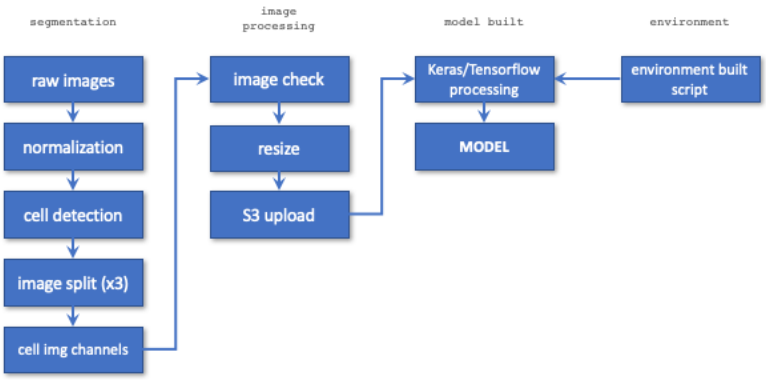

Figure 3.1: Model built: from raw images, a preprocess stage cell image are identified. A second stage checks the integrity of the images in some aspects like image size consistency for each channel and resizes each file to a common width and height. Environment built script creates the Cloud environment and automates the model built process.

After a description of the set of images, this chapter details the method-

ology used to:

1. Segment the images of the **human MCF7 cells - compound-profiling experiment** (Accession number **BBBC021**) to obtain a set of individual cells, choosing the ones that belong to the **DMSO** control group compound, since the focus of this work is to model the variability of a type of cells, not the variability among compounds.

2. Check integrity, enhance images to avoid low signals in training (dim images) and justify how the dimension of the images has been chose.

3. Describe how the cloud environment is performed, and the components that are used in model training.

4. Describe the architecture of the implemented VAE, justifying design decisions.

5. Describe the required steps to train a model and which parameters are involved.

6. Describe how the results have been analyzed and the metrics calculated for each model.

7. Depict which factors affects the budget for a model training

3.1 Image set

The image set used is the BBBC021v1 from [8], and it is available from the Broad Bioimage Benmarch Collection and described in [11]:

> *Phenotypic profiling attempts to summarize multiparametric, feature-based analysis of cellular phenotypes of each sample so that similarities between profiles reflect similarities between samples. Profiling is well established for biological readouts such as transcript expression and proteomics. Image-based profiling, however, is still an emerging technology.*
>
> *This image set provides a basis for testing image-based profiling methods wrt. to their ability to predict the mechanisms of action of a compendium of drugs. The image set was collected using a typical set of morphological labels and uses a physiologically relevant p53-wildtype breast-cancer model system (MCF-7) and a mechanistically distinct set of targeted and cancer-relevant cytotoxic compounds that induces a broad range of gross and subtle phenotypes.*

There are 39,600 image files (13,200 fields of view imaged in three channels) in TIFF format. Broad Institute provide the images in 55 ZIP archives, one for each microtiter plate. The archives are 750 MB each. The set also contains different files as **metadata**, with the information associated to each file like paths, filenames, plates and wells and the applied **compound and its concentration**. Additionally, the file `BBBC021_v1_compound.csv` gives the structures (in SMILES format) of most of the compounds.

Other files associated with the test also associate compounds and concentrations with mechanisms of action (`BBBC021_v1_moa.csv`). A **CellProfiler** pipeline to analyze the images is also provided.

Each image in the set is composed by **DAPI**, **TUBULIN** and **ACTIN** channels, and is also associated with the compound and the concentration of these compound. DAPI is a fluorescent marker that easily binds to AT sequences in DNA so is a good marker to stain the **nucleus**, while TUBULIN and ACTIN channels are related to the **cytoskeleton**. The image subset employed in this work is the subset of the BBBC021v1 which matches with the **DMSO** group compound, which corresponds to the control group; DMSO is a common solvent used for preparing stock solutions in fluorescence.

As it has been pointed out, the whole set has 13200 composed images that are 13200 files for each channel (one file for each channel) totalling 39600 files, while the DMSO subset has 1320 composed images that are 1320 images for each channel, totalling 3960 files.

3.2 Segmentation

This section describes the steps required to obtain individual cell images from the subset image.

3.2.1 CellProfiler. Pipeline and adaption.

CellProfiler [6] is an open-source software for quantitative analysis of biological images. The use of CellProfiler is implemented through pipelines, that are a sequential set of image analysis modules.

The segmentation process has also been based on a public pipeline [12] which has been adapted for the project objectives pruning subtasks related to the measurements like counting the number of cells or measuring nuclei radii that do not apply to the project goals. However, although not necessary for the outcomes, the normalization process was a prerequisite to pipeline, so it was also executed.

In the next section, the pipeline executed to segment the cells is described.

3.2.2 Regularization and Segmentation

Regularization

Because the analysis pipeline was developed to quantify some cellular parameters and those measurements require an image regularization 3.2 since illumination pipeline outcomes are used in the analysis, normalization must be executed before the main one.

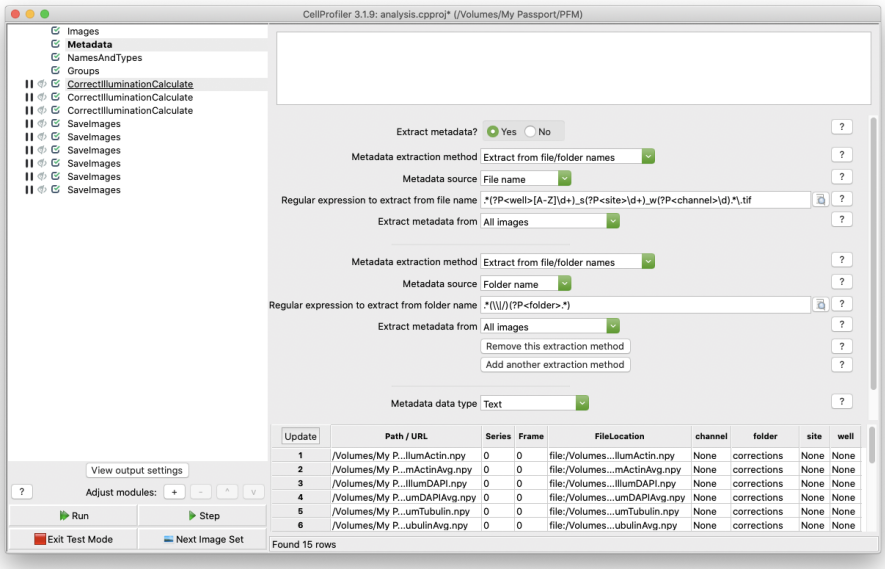

Figure 3.2: Regularization pipeline

After the **image definition** (drag and drop the files/folders from the filesystem), the pipeline defines the **metadata** of the associated files and **distinguishes** image channels. A **join** of the channels for the same image is performed before the **illumination correction**. The process ends with the **saving** of the images.

Segmentation

Pipeline 3.3 has in common with its prerequisite that first steps are basically the same: **image definition** and **metadata** extraction. After that, the correction of the channels is performed with the outcomes of the prior pipeline and corrected images are saved. A process of identification of the elements in the images in `IdentifyPrimaryObjects` is performed in DAPI (Blue), Actin (Green) and Tubulin (Red) channels. Finally the image is **cropped** and **saved** in one `.tiff` file for each channel.

As was mentioned, the original pipeline had more steps to perform calculations.

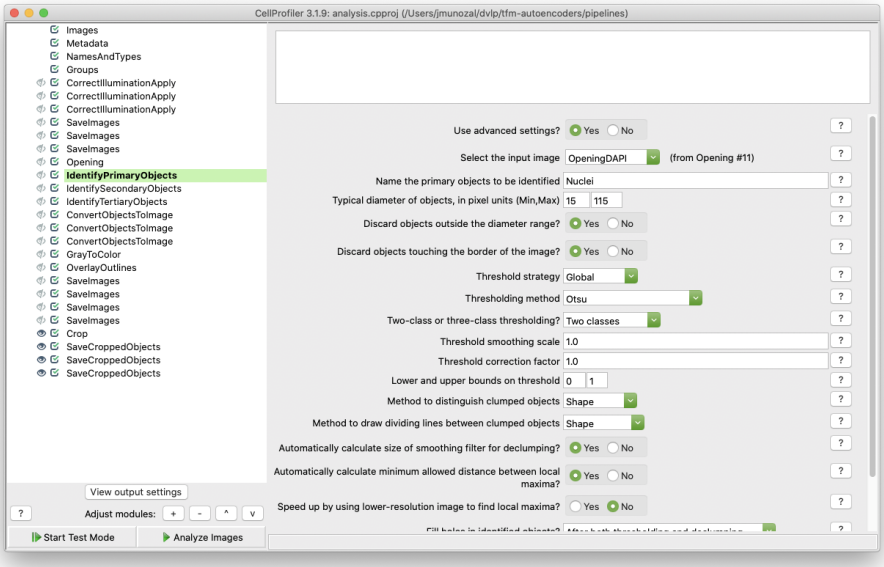

Figure 3.3: Analysis pipeline

3.3 Image processing

Image size

After cell segmentation, a maximum image cell size has to be choose. Considering performance restrictions, the image should not bee too large, without compromising information quantity and quality. The size `128x128` (see Figure 3.4) grabs most of the information of the image set.

Processing

After the images have been segmented, a process[1] is executed prior to the use of the images in the implementations of the VAE. Firstly, an image **resize** is done in each channel to normalize images to `128x128` size. After that, because **Keras** warns the user about the use of `.tiff` images, a `.png` file is created **combining** all channels. Finally, due images frequently are too dim, an **enhancement** is done through brightness correction.

```
# image syntesis (from .tiff to .png)
def create_png_images(folder, dest_folder):
    for file in glob.glob(''.join([folder, os.sep, "*.tiff"])):
        png_filename = os.path.basename(os.path.splitext(
```

[1] some functionality like enhancement or resizing could have been performed using **Keras/PIL** libraries

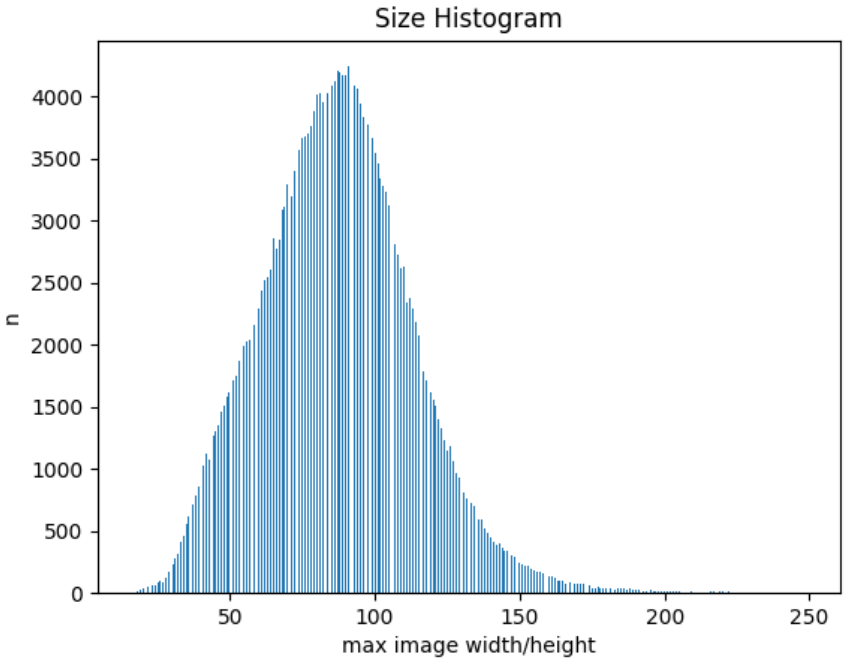

Figure 3.4: Image size histogram

```
 \\ file)[0] + '.png')
im = Image.open(file)
im.save(''.join([dest_folder, os.sep, png_filename]))
```

Once the images have been totally processed, they are uploaded to an S3 bucket.

3.4 Cloud environment

3.4.1 Introduction

Deep Learning training usually requires a significant amount of computational power, and although some studies[13] have challenged it, seems clear that one of the most optimal configuration for the task is associated to leverage **GPU**[2] performance due its ability to perform multiple parallel calculations in gradient descent algorithm. Thus, although possible, train deep learning models is a task that underperforms in a **CPU** architecture, which is the common one in general purpose computers.

In general, high-performance computers are not within reach on most occasions. At this point, the lack of computational resources is a significant

[2]Graphic Process Unit

challenge to train a model; because of that, a pay-per-use solution at the less possible cost is a significant improvement due to budgetary restrictions.

Cloud Computing offers a resource framework to deal with this kind of limitations. Besides, the possibility of environment description using tools like **Terraform** allows automating the creation and the destruction of network components, computing resources and persistence resources in a simple manner.

3.4.2 Components and environment

Components

Cloud components can be divided into three types:

- **Network components** allow the communication among cloud components and between those cloud components and the *local environment*, that is the environment that launches the formation scripts and captures the result of the model training to analyze. The main cloud network component is the **VPC**, that is the virtual network where to launch computing resources. The **Internet Gateway** is the component that connects the VPC with the Internet. Other components are the **Route Tables** to manage packet transmission and **Security Groups**, that are virtual firewalls.

- **Computing resources** are the core components of the system, where models are trained. The use of a standard template (Amazon Machine Image, AMI) pre-installed with Deep Learning tools (Tensorflow, Keras, Jupyter Notebook among others) allows a straightforward use of the environment, without the necessity to set up other software packages. The use of ML instance types, that combine CPU, memory, storage, and networking capacity in a certain way to optimize training works is also a key point in model training. The configurations used are:

Instance	**GPUs**	**vCPU**	**Mem (GiB)**	**Mem (GiB)**	**GPU P2P**	**Storage (GB)**	**Dedicated EBS Bandwidth**	**Networking Performance**
p3.2xlarge	1	8	61	16	-	EBS-Only	1.5 Gbps	Up to 10 Gigabit
p3.8xlarge	4	32	244	64	NVLink	EBS-Only	7 Gbps	10 Gigabit

- **Storage components** are the persistent components involved in the training. The S3 object storage is an optimal infrastructure not only to store images between trainings but also to store training results in `.h5` format.

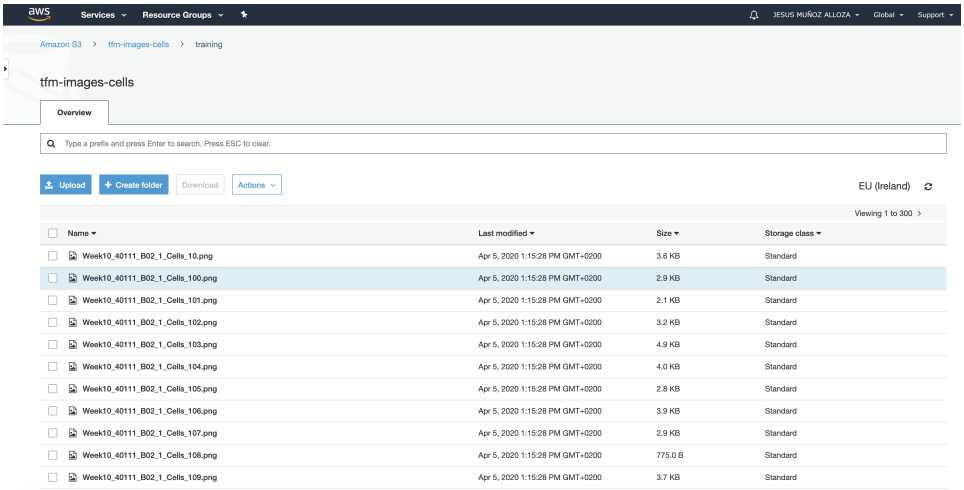

Figure 3.5: Images in a S3 bucket

Another resource that has been used to perform model training an EBS (Elastic Block Storage) unit is a virtual block disk that is attached to the computing resource and stores the images from the S3 resource (a. k. a. bucket). That is implemented in this fashion to avoid costly web service EC2 to S3 requests to get the images and because Python libraries (Tensorflow, Keras) natively implement location on disk.

Environment operation

The environment is built through a **terraform**[3] script, that processes the declarative description of components described in 3.4.2 and performs its creation in the cloud environment. The first step is the creation of the networking components; after that, the creation of the computational node and the attachment of the EBS disk allow the local system to access the **virtual machine (VM)** through a previously created public-private keypair via ssh.

The copy of the images to the VM is manually executed. It is performed through a utility that launches multiple threads and allows to concurrently access the S3 bucket, thus reducing the time to download the images. Once images are in the VM, files from the GIT repository containing the developed code is pulled from the repository[4]. Jupyter Notebook (JN) can be executed now in the root of this repository, and the notebook can be accessed from the local environment through a browser through the URL that AWS public DNS and JN token provides. The service is executed by default in `8888` port, so the access is done through the URL:

[3]https://www.terraform.io/
[4]https://github.com/jmunozal/tfm-autoencoders.git

`https://[awsnds]:8888?token=[token]`

Once the notebook `cell_analysis.ipynb` is parametrized, the execution can start, and when the process finishes, the results are manually grabbed to the S3 bucket. Once the results have been recovered, the process ends manually destroying the whole environment via Terraform.

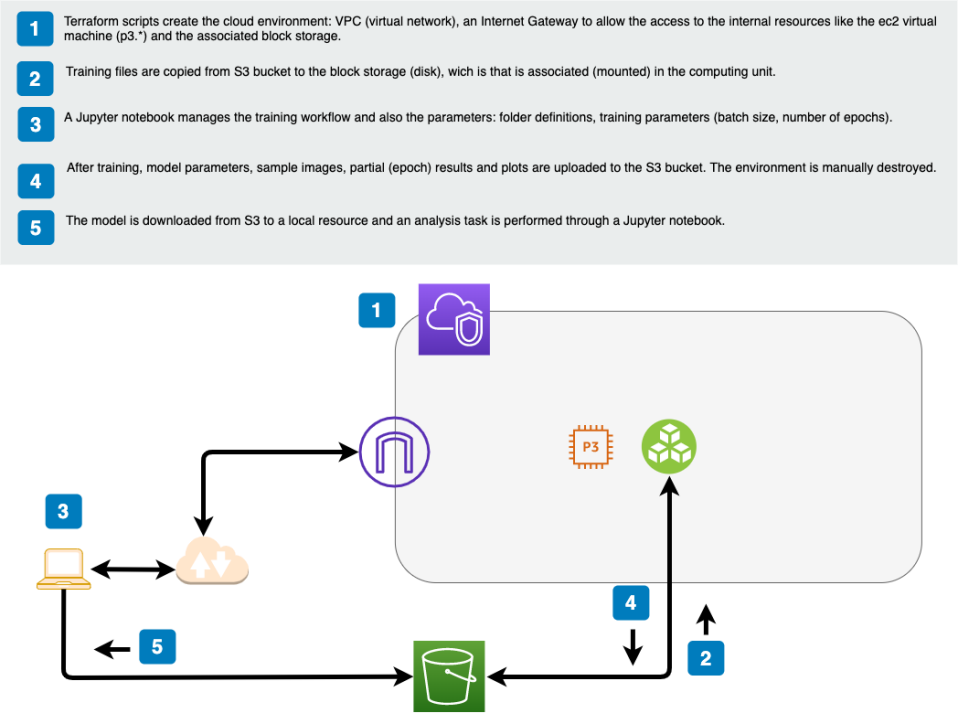

Figure 3.6: The environment generated by scripting the infrastructure. Options like virtual machine type, region and maximum spend amount are fully parametrized.

Detailed procedure is developed in Appendix B.

3.5 Model design and training

3.5.1 Model design

The Variational Autoencoder developed to model images is based in Deep Convolutional Neural Networks both in the encoder and in the decoder and in training mode the use of batch normalization and dropout can be activated or deactivated:

```
conv_t_layer_1b = Conv2DTranspose(
    filters=32
    , kernel_size=3
    , strides=2
```

```
        , padding='same'
        , name='decoder_conv_t_1b'
    )
    x = conv_t_layer_1b(x)
    if self.use_batch_norm:
        x = BatchNormalization()(x)
    x = LeakyReLU()(x)
    if self.use_dropout:
        x = Dropout(rate=0.25)(x)
```

After normalization/convolution layers, an activation layer is implemented based on the leaky version of the Rectified Linear Unit.

Model architecture

Puting all together the components: convolutional layer, ReLu layer, dropout and normalization a model has been designed and implemented. The model fora latent space of dimension 75 is ploted in figures 3.7 and 3.8.

3.5.2 Training steps

The source code is widely based on the example code of [2]. **Jupyter Notebook** `cell_train.ipynb` is implemented to execute the sequence of training steps. Kernel **Python 3.6** with **Tensorflow conda_tensorflow2_p36** is the kernel used to execute both **training** and **analysis** notebooks.

1. **Imports of the implemented VariationalAutoencoder class and the other necessary resources** like Keras ImageDataGenerator, numpy, glob and the os and time packages from the standard library.

2. **Definition of the variables for execution**, including `cloud_training`, that distinguishes the environment between cloud (real training) or local (development purposes). Depending on this variable, batch sizing, the number of epochs and folders are established. Image dimension is also set in this step.

```
DATETIME = time.strftime("%Y%m%d-%H%M%S")

LEARNING_RATE = 0.0005
R_LOSS_FACTOR = 1000

cloud_training = False

if cloud_training :
```

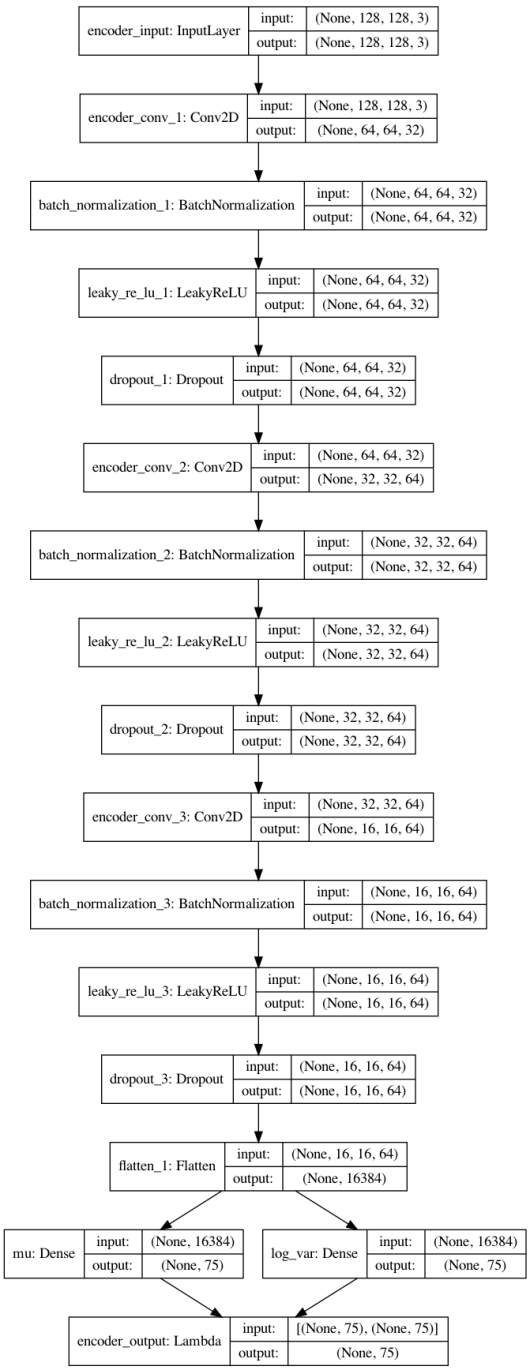

Figure 3.7: Encoder architecture for a LS dim 75

```
BATCH_SIZE = 35
EPOCHS = 100
DATA_FOLDER = '/data/train'
```

```
    RUN_FOLDER = '/data/run/'
else:
    BATCH_SIZE = 5
    EPOCHS = 1
    DATA_FOLDER = '/Volumes/My Passport/PFM/fastcheck'
    RUN_FOLDER =  '/Volumes/My Passport/PFM/run/'

run_id = '0001'
data_name = 'cells'
RUN_FOLDER += '_'.join([run_id, data_name, DATETIME])

INPUT_DIM = (128,128,3)
```

3. The **training setup** step includes the Variational Autoencoder instance set up and compilation using the defined variables in the latest step. Variational Autoencoder is parametrized with execution folders and some attributes like the **latent space** dimension and others to set the use of **dropout** and **batch_normalization**.

```
t = VariationalAutoencoder(image_folder=DATA_FOLDER, \\
       run_folder=RUN_FOLDER, train_mode=True, \\
       use_dropout=False, z_dim=400)

t.compile(learning_rate=LEARNING_RATE, r_loss_factor=R_LOSS_FACTOR)
```

4. Train execution using epochs:

```
t.train_with_generator(data_flow = data_flow, epochs = EPOCHS, \\
       steps_per_epoch = NUM_IMAGES / BATCH_SIZE, \\
       run_folder = RUN_FOLDER, print_every_n_batches = 10)
```

   The result of each partial result is printed on screen until the process ends:

```
Epoch 1/50
7166/7165 [==============================] - 302s 42ms/step - loss: 103.6501 - vae_r_loss: 84.7759 - vae_kl_loss: 18.8821

Epoch 00001: saving model to /data/run/0001_cells_20200513-131813/weights/weights-001-103.66.h5

Epoch 00001: saving model to /data/run/0001_cells_20200513-131813/weights/weights.h5

(...)

Epoch 00049: saving model to /data/run/0001_cells_20200513-131813/weights/weights.h5
Epoch 50/50
7166/7165 [==============================] - 296s 41ms/step - loss: 26.7648 - vae_r_loss: 19.8169 - vae_kl_loss: 6.9476

Epoch 00050: saving model to /data/run/0001_cells_20200513-131813/weights/weights-050-26.76.h5

Epoch 00050: saving model to /data/run/0001_cells_20200513-131813/weights/weights.h5
```

3.6 Model analysis

3.6.1 Trained models

Four models were trained with different LS dimension. The architecture of **models A, B and C** is the same (see Figures 3.7 and 3.8) excluding this LS Dim parameter. Model **model 3** has a slightly different output layer, but results were worse than the other design because although the model captured position, size and orientation, output pixels were **saturated**.

Trained Models				
model id	LS Dim	r_loss_factor	epochs	training time
model 3	200	10K	200	22 h
model A	75	50K	20	6 h
model B	150	10K	50	6 h
model C	300	10K	50	6 h

3.6.2 Metrics

In order to compare the results of the quality of reconstruction among the models, a metric has been established considering that all models have the same set of images to work with:

$$S = 1 - \frac{1}{c\ h\ w\ N} \sum_{\forall\ image} \sum_{\forall\ pixel} |i_1 - i_2|$$

Where c is the color deepness (255 in our case), h, w the height and witdth of the images and N the number of images; i are the intensities of each pixel (1:255), that is because we divide by the color deepness. This similarity metric S can be used for a collection of images of the same height and width and its range is between 0 (no similarity) and 1 (all images are equal).

Regarding the images collection, c is 256, h and w are 128 in both cases and the number of images N is **250.783**.

3.7 Budget

Based on the type of Amazon Machine Image (AMI) and the amount of time that the environment is up and running, an estimation can be done. In all cases the *price/hour* is the same and the AWS region is `eu-west-1` (Ireland). Machines are **spotted** so the price is substantially lower than on demand machines:

Model	AMI	Epochs	LS dim	Time	Price/hour	Total price
model 3	p3.2xlarge	200	200	22 h	1.80 €	39.60 €
model A	p3.2xlarge	20	75	6 h	1.80 €	10.80 €
model B	p3.2xlarge	20	150	6 h	1.80 €	10.80 €
model C	p3.2xlarge	20	300	6 h	1.80 €	10.80 €

Tariffs for other regions are different, and the price for persistence storage is about 7 €/month for both the models and the images.

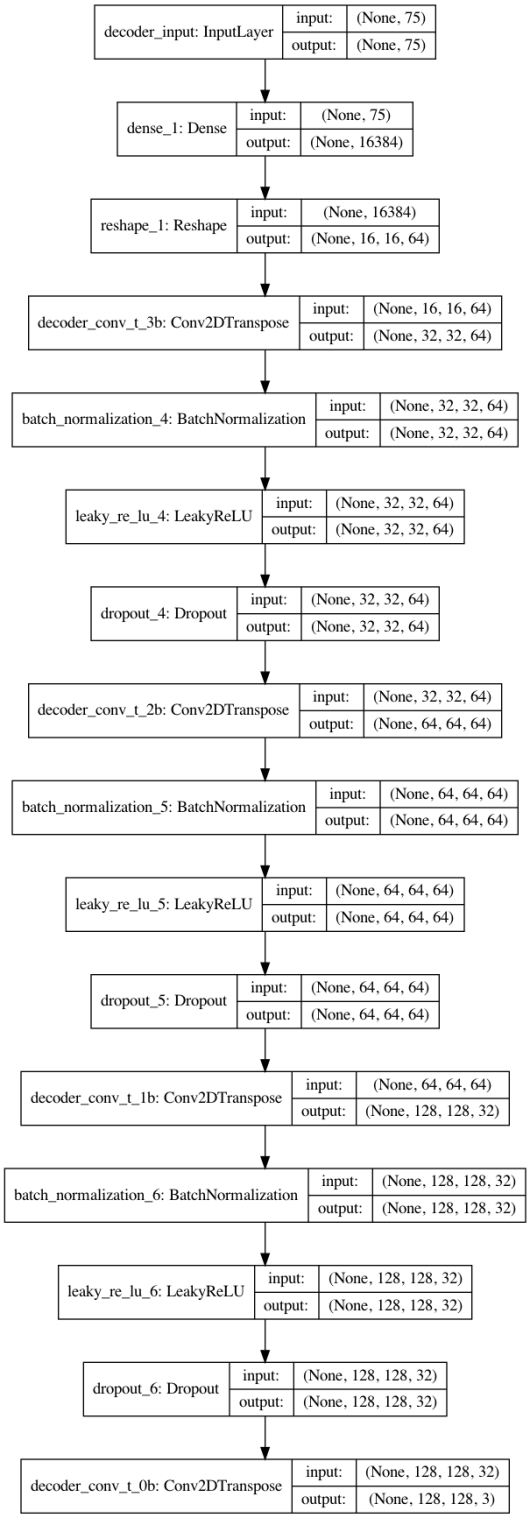

Figure 3.8: Decoder architecture for a LS dim 75

Chapter 4

Results

4.1 Segmentation

Although resource intensive, the process can be performed in a standard computer, firstly generating the illumination adjusts with `illum.cppie` and afterwards proceeding to the segmentation with `analysis.cpppie` pipeline. An example of the segmentation for one file is show in figure 4.1.

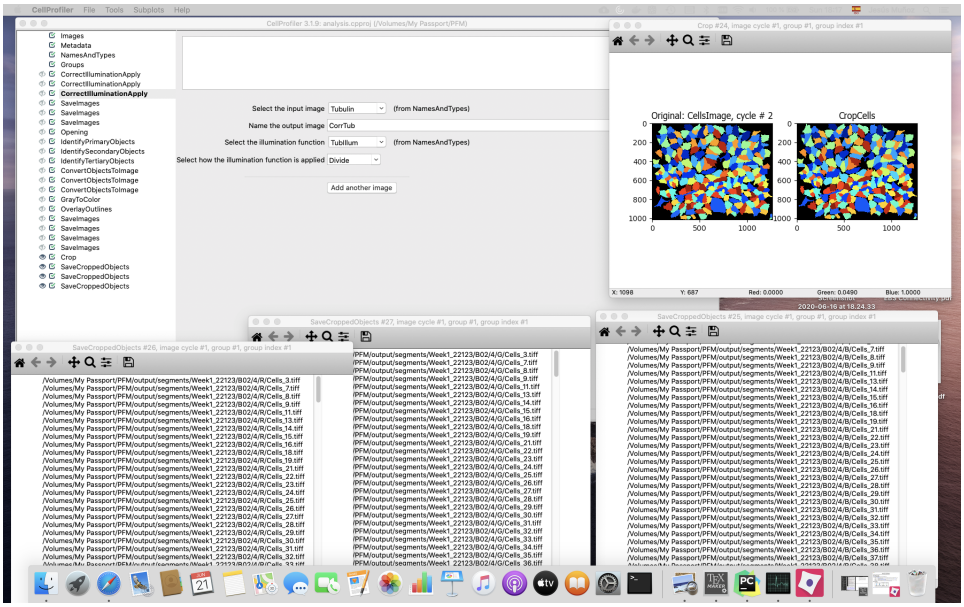

Figure 4.1: Segmentation process for an image. The output are the multiple cropped files saved (R/G/B) for each cell (numbered suffix). Division in cropped cells can be observed in the upper right side.

4.2 Training process

Loss function stabilizes quickly, and around epoch 10 the value is not substantially reduced; 4.2 is the plot for loss function for **model B**.

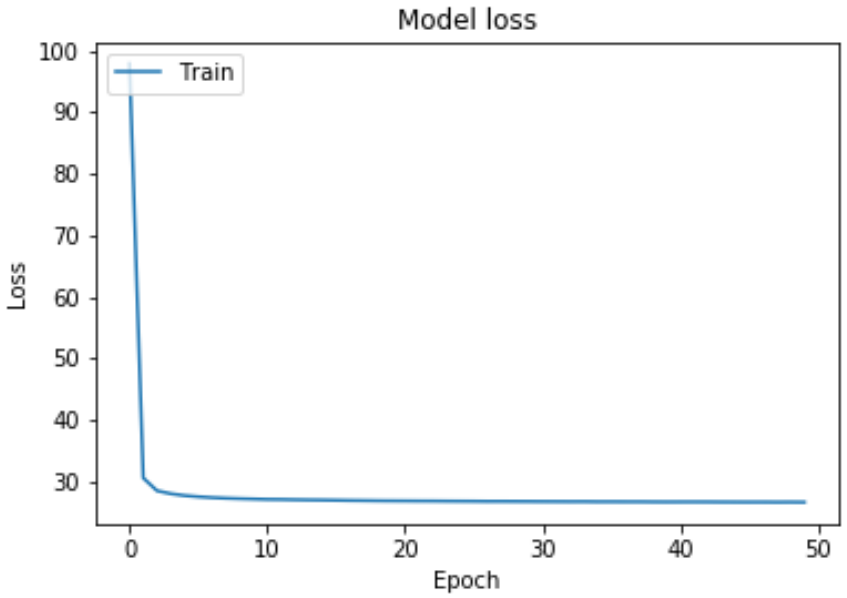

Figure 4.2: Loss plot for the model **model B**

Models B and C had to be mandatory trained using **dropout** layers; otherwise the loss function diverges (NaN values). Dropout layer ignores neurons in the training phase not considering them in forward or backward pass.

Latent space distribution that is detailed in Figure 4.3 for the 20 first components of a 5000 sample size shows that some distributions are **significantly different** from the standard normal distribution. The poor adjust of some components points out that the relation between both terms of the loss function should have been different, reducing the reconstruction loss factor to get a more optimal latent space distribution.

4.2.1 Metrics

Metrics for the models reveal that the change in the network design has improved metric's value, and even the model with lowest latent space improves the result of the old architecture. The results also indicate that the increase of the latent space (between models B and C) to a wider one does not imply a better performance.

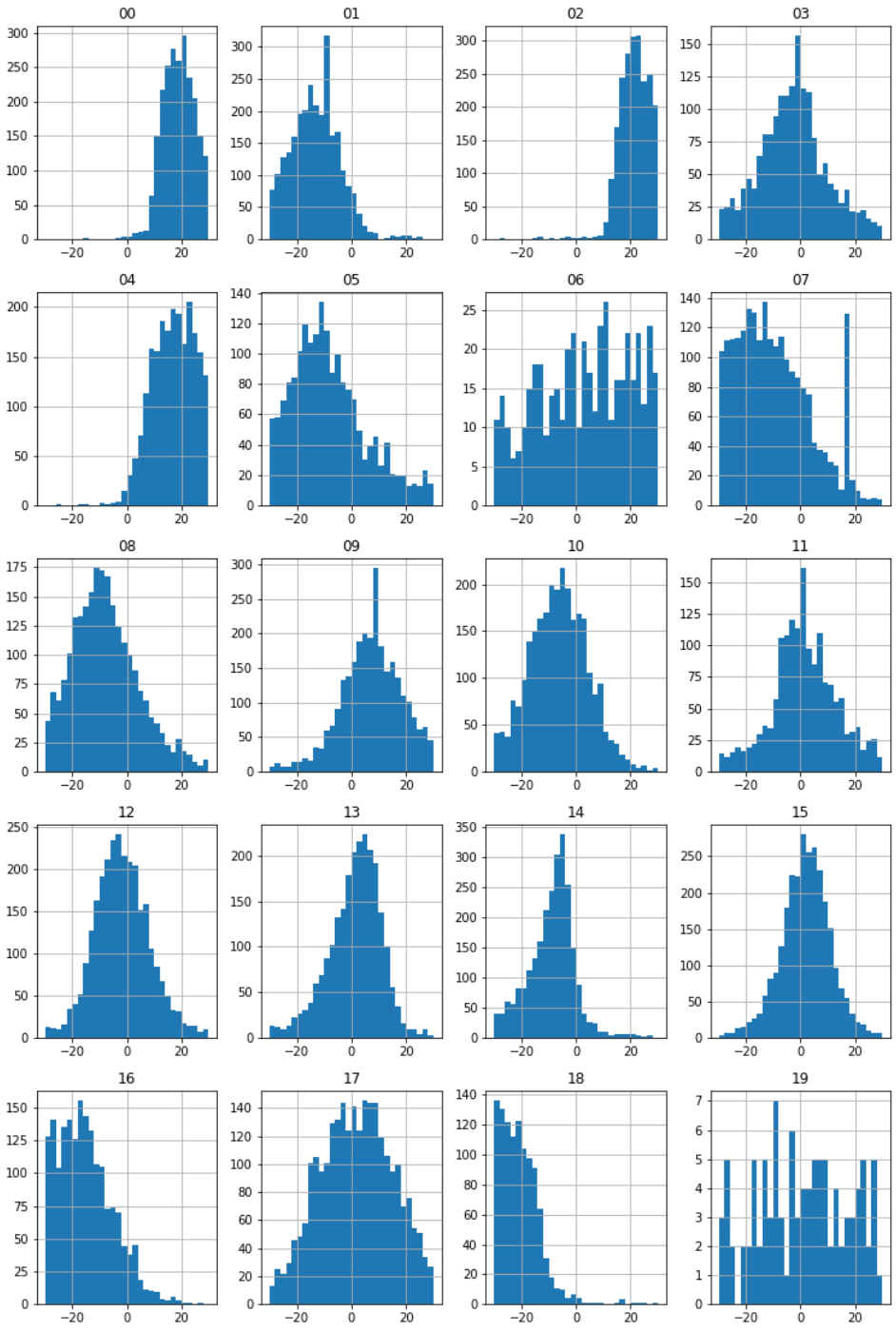

Figure 4.3: Latent space distribution for **model B**

Model	Metric	latent space size
model 3 (PAC 2)	0.82813	200
model A	0.85946	75
model B	0.91598	150
model C	0.90817	300

A possible cause is that because models B and C can **reconstruct more images**, their metrics value is better than model A, although model A produces more detailed images. Is reasonable to infer that because the high variability of the images not only in size but also about predominant channels and orientations, the model does not capture all variables and is not able to reconstruct more configurations than models B and C, reducing this way its metrics value.

4.3 Reconstructing images from the image set

Although some images are not properly reconstructed, in general the quality is fair not only under an abstract point of view, but also under a more objective angle as section 4.2.1 shows.

1. *in general*, models capture position and orientation of the cell
2. models distinguish among color layers, specially **model A** (see Figure 4.4). A possible justification of this will be developed supported by [14] in the final work.
3. better performance of **model A** regarding details contrasts in respect to the ability of this model to reconstruct some images (see Figure 4.5)
4. some images are not properly reconstructed in any model (see Figure 4.6)

The problem of **meaningless reconstruction points** in figures 4.5 and 4.6 points out that **the decoder can not rebuild the image through the encoder result**: because the encoder has returned a *coordinate* in the latent space that is out the modeled space, the system has *overfitted* the point, behaving like a *classic* autoencoder. An improvement in the training process to adjust the latent space is the first reasonable step to fix this point as it was pointed in 4.2. Another possible cause is that the image is an *outlier* of the training subset, so the training set should be widened.

4.4 Image generation

Twenty points have been sampled from a normal distribution with z=75 dimensions to generate new cells, whose results are the images in Figure 4.7. Although images suffer from a common VAE blurriness, these are a good approximation.

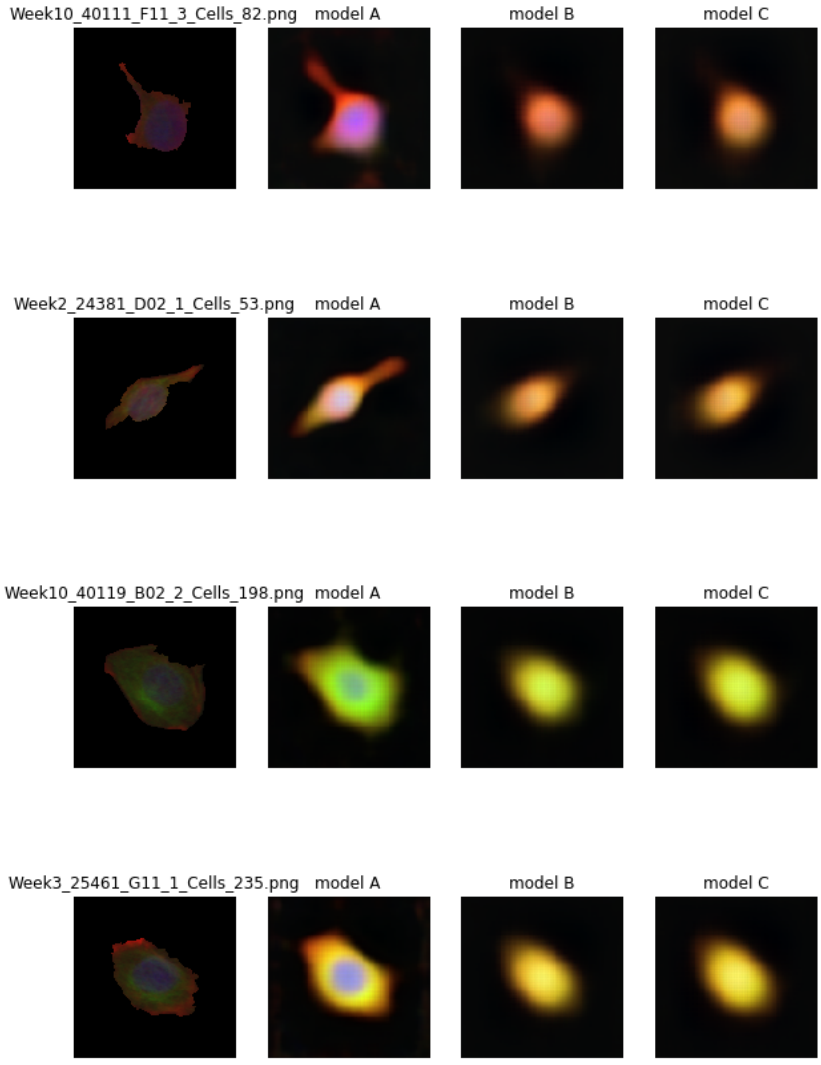

Figure 4.4: Although its latent space has less components, **model A** captures more detail than **models B** and **C**

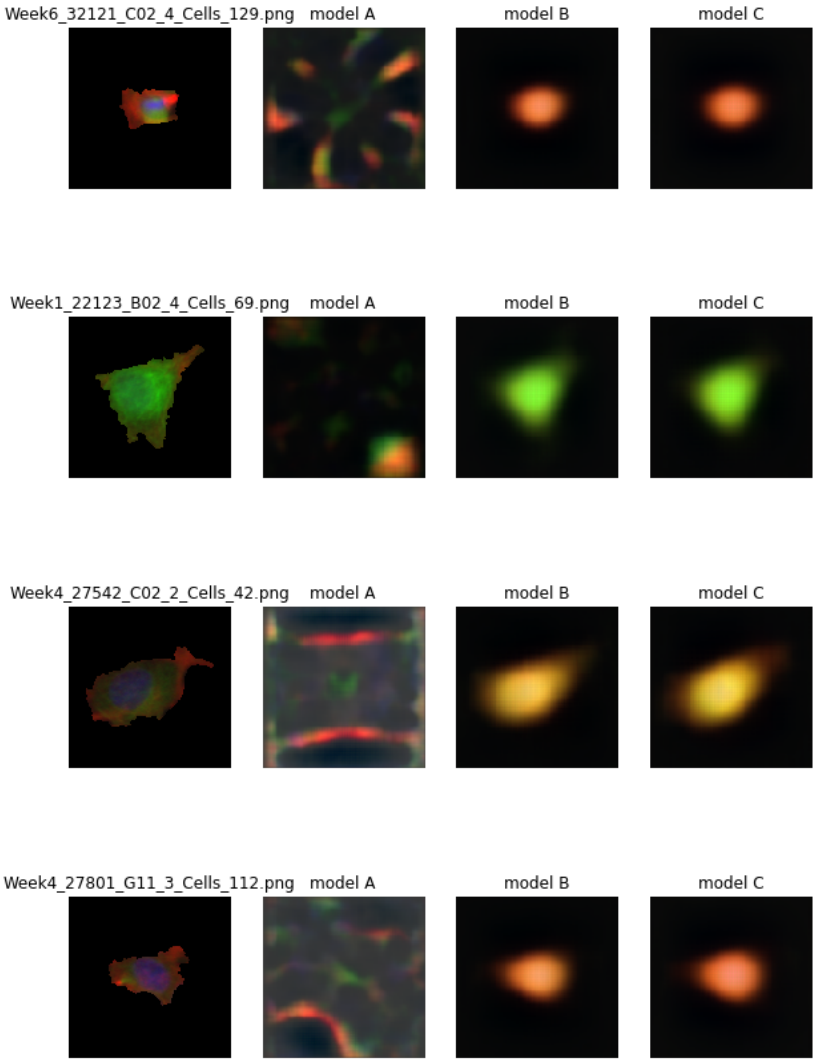

Figure 4.5: Models with wider latent spaces are able to reconstruct a long number of images

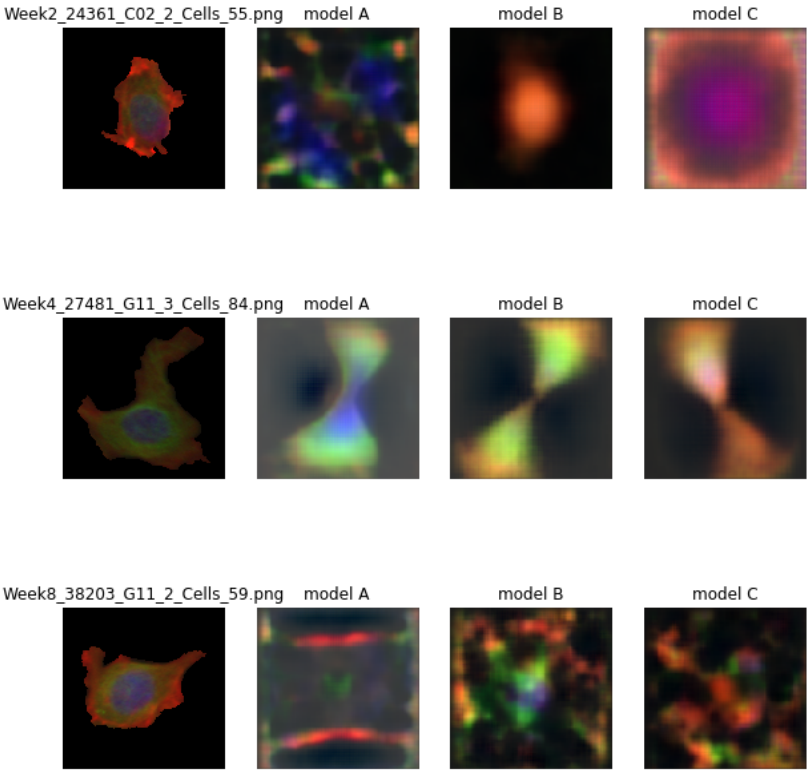

Figure 4.6: Some images are not properly reconstructed

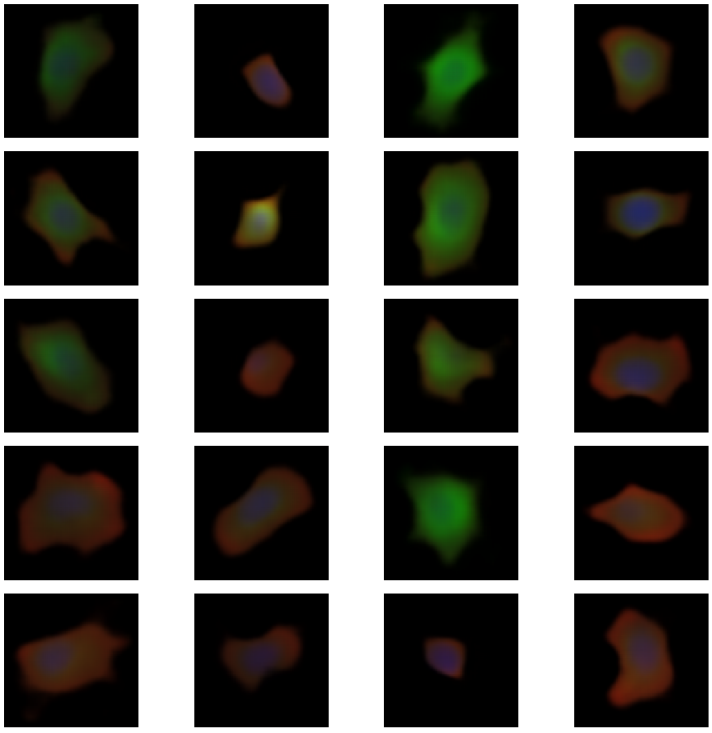

Figure 4.7: Cell images generated with a variational autoencoder of latent space dimension 75

Chapter 5

Conclusions

Outcomes

The implementation of two variational autoencoder architectures (model 3 and model A/B/C) that differs in its decoder design indicates that the performance of the VAE is affected by minor changes like the addition or removal of an activation layer in the output layer.

The analysis of **models A/B/C** that produce a better outcome that differ only in its latent space dimension suggests that **widen this does not produce a better result** at least in the sense of abstract human perception.

Model A is the most **disentangled** [14], dimensionally efficient model (the idea under disentangling is that each dimension in LV encodes an underlying independent factor of variation) that produces, although noisy, **the more performant image reconstructions**.

The improved metrics and primary cell reconstructions of B and C models (see Figure 4.5) points out that a *possible* cause is that widen latent variable spaces allows the VAE to model essential treats like size or orientation in a straight way than model A.

One hypothesis to interpret **collapsed reconstructions** is that the number of examples in the training set could be too low to express the extreme variability of the images, or the failed cases do not fit the latent space distribution. However, because some components of the latent space **does not fit very well the standard normal distribution**, the most likely cause is that images are badly encoded in some of the cases and the decoder is unable to rebuild the image.

Technology

The use of the public **cloud infrastructure** and its pay-per-use price modelling added to the run of pre-built environments that eases the use of pre-installed frameworks and tools makes assumable to tackle a project of this

size. The use of a tool like **Terraform** adds an automation layer that avoids delays between computing components creation and model training.

Further steps

Regarding **outcomes**, future steps would be to check the validity of the hypothesis regarding the failed reconstructions, mainly rebalancing the loss function *reconstruction* and *regularization* terms. With this information, the study of the labelled sets (drugs and concentrations) can be tackled. The **technology** aspect can be improved dealing with the automation of the whole process, from creation and training to upload to the model's repository and the removal of the created environment.

Bibliography

[1] Yanqing Yang, Kangfeng Zheng, Chunhua Wu, and Yixian Yang. Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network. *Sensors*, 19(11):2528, 2019.

[2] D. Foster. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play.* O'Reilly Media, 2019.

[3] embarc machine learning inference library documentation. `https://embarc.org/embarc_mli/doc/build/html/index.html`.

[4] Juan C Caicedo, Shantanu Singh, and Anne E Carpenter. Applications in image-based profiling of perturbations. *Current opinion in biotechnology*, 39:134–142, 2016.

[5] Maxime W. Lafarge, Juan C. Caicedo, Anne E. Carpenter, Josien P.W. Pluim, Shantanu Singh, and Mitko Veta. Capturing single-cell phenotypic variation via unsupervised representation learning. In M. Jorge Cardoso, Aasa Feragen, Ben Glocker, Ender Konukoglu, Ipek Oguz, Gozde Unal, and Tom Vercauteren, editors, *Proceedings of The 2nd International Conference on Medical Imaging with Deep Learning*, volume 102 of *Proceedings of Machine Learning Research*, pages 315–325, London, United Kingdom, 08–10 Jul 2019. PMLR.

[6] Anne E. Carpenter, Thouis R. Jones, Michael R. Lamprecht, Colin Clarke, In Han Kang, Ola Friman, David A. Guertin, Joo Han Chang, Robert A. Lindquist, Jason Moffat, Polina Golland, and David M. Sabatini. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biology*, 7(10):R100, Oct 2006.

[7] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019.

[8] Peter D Caie, Rebecca E Walls, Alexandra Ingleston-Orme, Sandeep Daya, Tom Houslay, Rob Eagle, Mark E Roberts, and Neil O Carragher. High-content phenotypic profiling of drug response signatures across

distinct cancer cells. *Molecular cancer therapeutics*, 9(6):1913–1926, 2010.

[9] O. Du rr and Beate Sick. Single-cell phenotype classification using deep convolutional neural networks. *Journal of Biomolecular Screening*, 21, 02 2016.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Broad Institute. Human mcf7 cells – compound-profiling experiment. `https://data.broadinstitute.org/bbbc/BBBC021/`.

[12] Kyle W. Karhohs. Pipelines for analyzing bbbc021 data set. `https://forum.image.sc/t/pipelines-for-analyzing-bbbc021-data-set/13967`.

[13] Y. Ma, F. Rusu, and M. Torres. Stochastic gradient descent on modern hardware: Multi-core cpu or gpu? synchronous or asynchronous? In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1063–1072, May 2019.

[14] Cody Marie Wild. What a disentangled net we weave: Representation learning in vaes (pt. 1). `https://towardsdatascience.com/what-a-disentangled-net-we-weave-representation-learning-in\discretionary{-}{}{}vaes-pt-1-9e5dbc205bd1`.

[15] Baptiste Rocca Joseph Rocca. Understanding variational autoencoders (vaes). `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, USA, 2002.

[18] Carl Doersch. Tutorial on variational autoencoders, 2016.

[19] Juan C Caicedo, Sam Cooper, Florian Heigwer, Scott Warchal, Peng Qiu, Csaba Molnar, Aliaksei S Vasilevich, Joseph D Barry, Harmanjit Singh Bansal, Oren Kraus, et al. Data-analysis strategies for image-based cell profiling. *Nature methods*, 14(9):849, 2017.

[20] Louis Tiao. Implementing variational autoencoders in keras: Beyond the quickstart tutorial. `http://louistiao.me/posts/implementing-variational-autoencoders-in-keras-\beyond-the-quickstart-tutorial`.

[21] Wikipedia: convolution. https://en.wikipedia.org/wiki/Convolution.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

Glossary

autoencoder a type of artificial neural network that consists of a codification phase to reduce dimensionality and a decode phase to regenerate the input. 14, 15

convolution mathematical operation on two functions (f and g) that produces a third function expressing how the shape of one is modified by the other. 16

convolutional neural network a subset of deep learning and neural networks most commonly used to analyze visual imagery. 16

discriminative modeling a class of supervised machine learning used for classification or regression. 14

internal covariate shift the change in the distribution of network activations due to the change in network parameters during training. 16

kernel matrix used to extract features from an input image. 16

latent space a representation of compressed data. 15, 29, 31, 36, 38, 43

loss function a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some cost associated with the event. 15, 16, 36, 44

variational autoencoder an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process [15]. 7, 15, 16, 43

Acronyms

AMI Amazon Machine Image. 32

AWS Amazon Web Services. 10, 27, 32, 57

CNN convolutional neural network. 16

DM discriminative modelling. 14

GM generative modelling. 14

KL Kullback–Leibler. 16

L2 least square errors. 15

LeakyReLU leaky rectifier linear unit. 17

LS latent space. 14, 15, 31, 32

PCA principal component analysis. 14

VAE variational autoencoder. 9, 10, 11, 12, 13, 21, 24, 38, 43

Appendices

Appendix A

Source code

Source code is available at `https://github.com/jmunozal/tfm-autoencoders`. The repository is divided into three sections: **pipelines**, which contains CellProfiler pipelines, **terraform** that includes the scripts to create the training environment and **vae-images** that includes the Python/Jupyter files to manage images (images folder), the VAE (model) and utilities (utils).

Appendix B

Model training in cloud environments

B.1 Prerequisites

A connection to the Internet, the image set in a S3 bucket (in this case, the bucket is **tfm-images-cells** and it has a subfolder training with all the image set) and the tools **terraform** and **aws cli** installed and configured are the prerequisites to execute the training.

⚠ This appendix does not address the configuration of the **AWS CLI** tool regarding **security** (tokens, authentication), neither the use of **Terraform** tool.

B.2 Preparing the environment for training process

Download GIT repository and change folder

Once we choose a folder, we download the repository from
`https://github.com/jmunozal/tfm-autoencoders.git`
and after the tfm-autoencoders folder can be checked:

```
[jmunozal@MacBook-Pro temp]$ git clone https://github.com/jmunozal/tfm-autoencoders.git
Cloning into 'tfm-autoencoders'...
remote: Enumerating objects: 268, done.
remote: Counting objects: 100% (268/268), done.
remote: Compressing objects: 100% (181/181), done.
remote: Total 268 (delta 136), reused 212 (delta 81), pack-reused 0
Receiving objects: 100% (268/268), 143.42 KiB | 848.00 KiB/s, done.
Resolving deltas: 100% (136/136), done.

[jmunozal@MacBook-Pro temp]$ cd tfm-autoencoders/
[jmunozal@MacBook-Pro tfm-autoencoders]$ ls -ltra
total 8
drwxr-xr-x   4 jmunozal  staff  128 Jun 16 17:34 ..
-rw-r--r--   1 jmunozal  staff   97 Jun 16 17:34 .gitignore
drwxr-xr-x   7 jmunozal  staff  224 Jun 16 17:34 pipelines
drwxr-xr-x   9 jmunozal  staff  288 Jun 16 17:34 terraform
drwxr-xr-x   7 jmunozal  staff  224 Jun 16 17:34 .
drwxr-xr-x  12 jmunozal  staff  384 Jun 16 17:34 vae-images
```

```
drwxr-xr-x  12 jmunozal  staff  384 Jun 16 17:34 .git
[jmunozal@MacBook-Pro tfm-autoencoders]$
```

The repository is divided into three sections: **pipelines**, which contains CellProfiler pipelines, **terraform** that includes the scripts to create the training environment and **vae-images** that includes the Python/Jupyter files to manage images (images folder), the VAE (model) and utilities (utils).

Generating the environment with terraform

The environment has to be generated in **terraform** folder using `init` command:

```
[jmunozal@MacBook-Pro terraform]$ terraform init
Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "aws" (2.66.0)...

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
[jmunozal@MacBook-Pro terraform]$
```

Terraform folder includes:

```
[jmunozal@MacBook-Pro terraform]$ ll
total 56
-rw-r--r--  1 jmunozal  staff   400 Jun 16 17:34 download_data.sh
-rw-r--r--  1 jmunozal  staff   791 Jun 16 17:34 ec2.tf
-rw-r--r--  1 jmunozal  staff   569 Jun 16 17:34 networks.tf
-rw-r--r--  1 jmunozal  staff  1568 Jun 16 17:34 s3.tf
-rw-r--r--  1 jmunozal  staff  1550 Jun 16 17:34 security.tf
-rw-r--r--  1 jmunozal  staff   120 Jun 16 17:34 terraform.tfvars
-rw-r--r--  1 jmunozal  staff   375 Jun 16 17:34 variables.tf
```

The files are:

- **ec2.tf:** defines the computational node and the block storage (disk) associated
- **networks.tf:** virtual network, internet gateway and routing tables
- **s3.tf:** security policies for the already created buckets
- **security.tf:** network security. Open ports 8888 for Jupyter and 22 for ssh access.
- **variables.tf/terraform.tfvars:** setup parameters

`terraform.tfvars` file contains the configuration for the files.

```
[jmunozal@MacBook-Pro terraform]$ cat terraform.tfvars
aws_region="eu-west-1"
ami_id="ami-078d068af898f9114"
spot_price=2.00
instance_type="p3.2xlarge"
device_name="/dev/sdb"
[jmunozal@MacBook-Pro terraform]$
```

Also notice that ec2.tf has a reference to a key that has to be generated using IAM to access the server using SSH[1].

If the environment is properly configured, the execution of `terraform apply` will render the cloud objects to be created:

```
[jmunozal@MacBook-Pro terraform]$ terraform apply
data.aws_s3_bucket.a: Refreshing state...
data.aws_s3_bucket.b: Refreshing state...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_default_route_table.r
      id:                                           <computed>
      default_route_table_id:                       "${aws_vpc.main.default_route_table_id}"
      owner_id:                                     <computed>
      route.#:                                      "1"
      route.~966145399.cidr_block:                  "0.0.0.0/0"
      route.~966145399.egress_only_gateway_id:      ""
      route.~966145399.gateway_id:                  "${aws_internet_gateway.gw.id}"
      route.~966145399.instance_id:                 ""
      route.~966145399.ipv6_cidr_block:             ""
      route.~966145399.nat_gateway_id:              ""
      route.~966145399.network_interface_id:        ""
      route.~966145399.transit_gateway_id:          ""
      route.~966145399.vpc_peering_connection_id:   ""
      tags.%:                                       "1"
      tags.Name:                                    "main"
      vpc_id:                                       <computed>

  + aws_iam_instance_profile.modeller_profile
      id:                                           <computed>
      arn:                                          <computed>
      create_date:                                  <computed>
      name:                                         "modeller_profile"
      path:                                         "/"
      role:                                         "s3_reader_role"
      roles.#:                                      <computed>
      unique_id:                                    <computed>

  + aws_iam_policy.policy
      id:                                           <computed>


(...)

  + aws_vpc.main
      id:                                           <computed>
      arn:                                          <computed>
      assign_generated_ipv6_cidr_block:             "false"
      cidr_block:                                   "10.0.0.0/16"
      default_network_acl_id:                       <computed>
      default_route_table_id:                       <computed>
      default_security_group_id:                    <computed>
      dhcp_options_id:                              <computed>
      enable_classiclink:                           <computed>
      enable_classiclink_dns_support:               <computed>
      enable_dns_hostnames:                         "true"
      enable_dns_support:                           "true"
      instance_tenancy:                             "default"
      ipv6_association_id:                          <computed>
      ipv6_cidr_block:                              <computed>
      main_route_table_id:                          <computed>
      owner_id:                                     <computed>


Plan: 12 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

[1] `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html`. Section Locate the private key/Creating a Key Pair Using Amazon EC2

Once we answer 'yes' the process starts and if it is finished properly:

```
  wait_for_fulfillment:                             "" => "true"
aws_default_route_table.r: Creation complete after 1s (ID: rtb-029e3fac77fcf1b67)
aws_s3_bucket_policy.b: Creation complete after 9s (ID: tfm-images-cells)
aws_s3_bucket_policy.a: Still creating... (10s elapsed)
aws_spot_instance_request.dl_worker: Still creating... (10s elapsed)
aws_s3_bucket_policy.a: Creation complete after 17s (ID: segments-dmso-resized)
aws_spot_instance_request.dl_worker: Creation complete after 18s (ID: sir-5q1r5wsn)

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.
[jmunozal@MacBook-Pro terraform]$
```

Now we can check in Figure B.2 in the console that the resources are created:

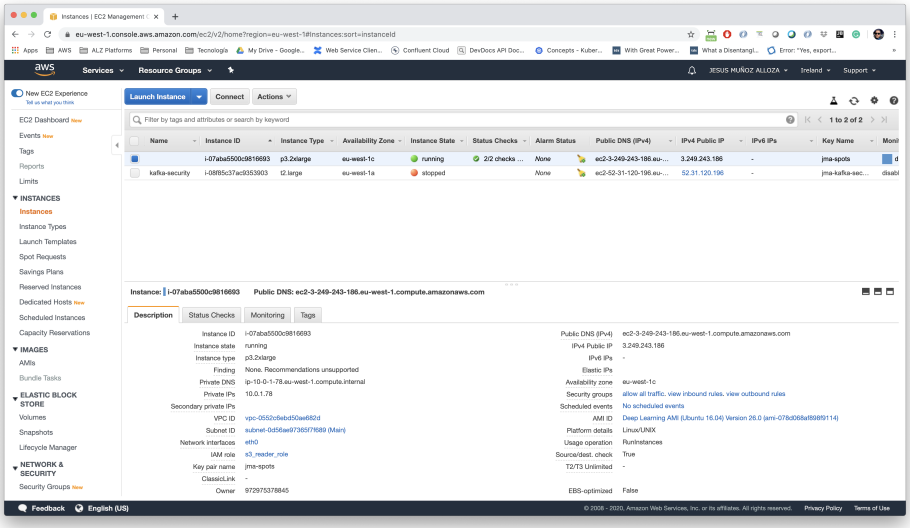

Figure B.1: Virtual machine created after terraform success execution

Access to the VM

Using the private key of our key pair and the DNS created in the last step, we can access to the VM:

```
ssh -i /Users/jmunozal/keys/jma-spots.pem ubuntu@
  \\ ec2-3-249-243-186.eu-west-1.compute.amazonaws.com
```

Downloading the images

Using the s3copier tool that has been downloaded in the creation process, images can be now downloaded:

```
ubuntu@ip-10-0-1-78:~$ export AWS_REGION=eu-west-1 && /tmp/s3copier
  \\ -bucket=tfm-images-cells -baseDir=/data/train -concurrency 30
2020/06/16 16:35:18 Copied 715 files from s3 in 1.000132424s (2.2380 MiB/s)
2020/06/16 16:35:19 Copied 1636 files from s3 in 2.000136126s (2.7734 MiB/s)
(...)
```

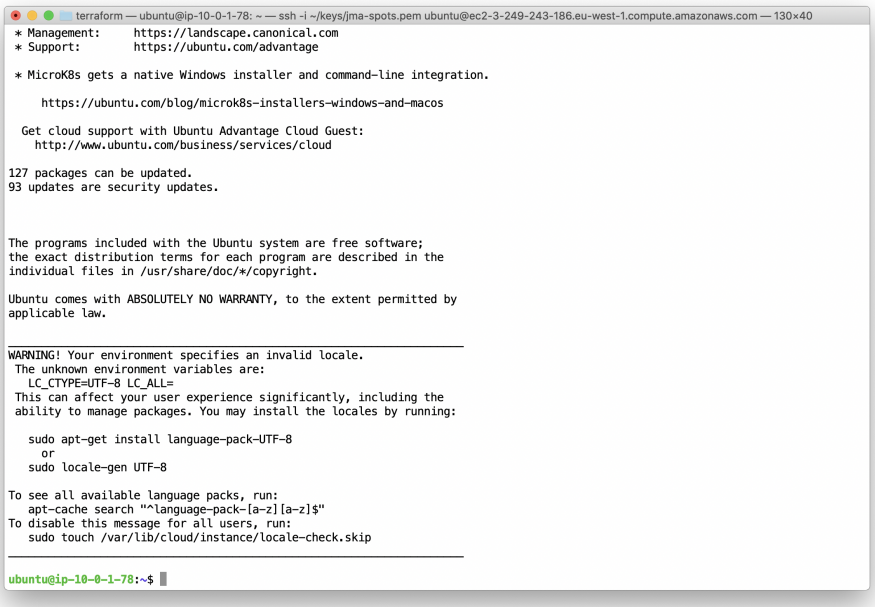

Figure B.2: Access to a remote machine using ssh

```
2020/06/16 16:40:09 Copied 247812 files from s3 in 4m52.000147235s (3.1549 MiB/s)
2020/06/16 16:40:10 Copied 248670 files from s3 in 4m53.000180143s (3.1391 MiB/s)
2020/06/16 16:40:11 Copied 249538 files from s3 in 4m54.000163289s (3.1845 MiB/s)
2020/06/16 16:40:12 Copied 250411 files from s3 in 4m55.000148383s (3.2482 MiB/s)
2020/06/16 16:40:12 Total number of files: 250782, Total time taken: 4m55.405649675s,
  \\Transfer rate 2.8696 MiB/s
ubuntu@ip-10-0-1-78:~$
```

Downloading the git repo and launching Jupyter

Now we can get (again) the GIT repo in, for instance, the `home` folder and now we access to the vae-images folder:

```
ubuntu@ip-10-0-1-78:~$ git clone https://github.com/jmunozal/tfm-autoencoders.git
Cloning into 'tfm-autoencoders'...
remote: Enumerating objects: 268, done.
remote: Counting objects: 100% (268/268), done.
remote: Compressing objects: 100% (181/181), done.
remote: Total 268 (delta 136), reused 212 (delta 81), pack-reused 0
Receiving objects: 100% (268/268), 143.42 KiB | 0 bytes/s, done.
Resolving deltas: 100% (136/136), done.
Checking connectivity... done.
ubuntu@ip-10-0-1-78:~$ cd tfm-autoencoders/
ubuntu@ip-10-0-1-78:~/tfm-autoencoders$ cd vae-images/
ubuntu@ip-10-0-1-78:~/tfm-autoencoders/vae-images$
```

And launch **Jupyter notebook** (notice the `--ip` parameter):

```
ubuntu@ip-10-0-1-78:~/tfm-autoencoders/vae-images$ jupyter notebook --ip 0.0.0.0
[I 16:41:06.923 NotebookApp] Using EnvironmentKernelSpecManager...
[I 16:41:06.924 NotebookApp] Started periodic updates of the kernel list (every 3 minutes).
[I 16:41:06.931 NotebookApp] Writing notebook server cookie secret to /run/user/1000/jupyter/notebook_cookie_secret
[I 16:41:11.943 NotebookApp] Loading IPython parallel extension
[I 16:41:12.377 NotebookApp] JupyterLab beta preview extension loaded from /home/ubuntu/anaconda3/lib/python3.6/site-packages/jupyterlab
[I 16:41:12.377 NotebookApp] JupyterLab application directory is /home/ubuntu/anaconda3/share/jupyter/lab
```

```
[I 16:41:26.679 NotebookApp] [nb_conda] enabled
[I 16:41:26.683 NotebookApp] Serving notebooks from local directory: /home/ubuntu/tfm-autoencoders/vae-images
[I 16:41:26.683 NotebookApp] 0 active kernels
[I 16:41:26.683 NotebookApp] The Jupyter Notebook is running at:
[I 16:41:26.683 NotebookApp] http://ip-10-0-1-78:8888/?token=a8d5b0d5ca8c100eb870eff37edc4eb9c0baa51d583cf982
[I 16:41:26.683 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 16:41:26.683 NotebookApp] No web browser found: could not locate runnable browser.
[C 16:41:26.683 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://ip-10-0-1-78:8888/?token=a8d5b0d5ca8c100eb870eff37edc4eb9c0baa51d583cf982
        \\ &token=a8d5b0d5ca8c100eb870eff37edc4eb9c0baa51d583cf982
[I 16:41:26.685 NotebookApp] 302 GET / (209.17.97.18) 0.66ms
[I 16:41:26.686 NotebookApp] 302 GET / (209.17.97.18) 0.32ms
[I 16:41:26.686 NotebookApp] Starting initial scan of virtual environments...
```

B.3 Training

Training Jupyter notebook

Using the generated token and the AWS ec2 DNS we can get from the console the access url to Jupyter NB can be constructed:

http://ec2-3-249-243-186.eu-west-1.compute.amazonaws.com:8888
?token=a8d5b0d5ca8c100eb870eff37edc4eb9c0baa51d583cf982

The image should be similar to Image B.3.

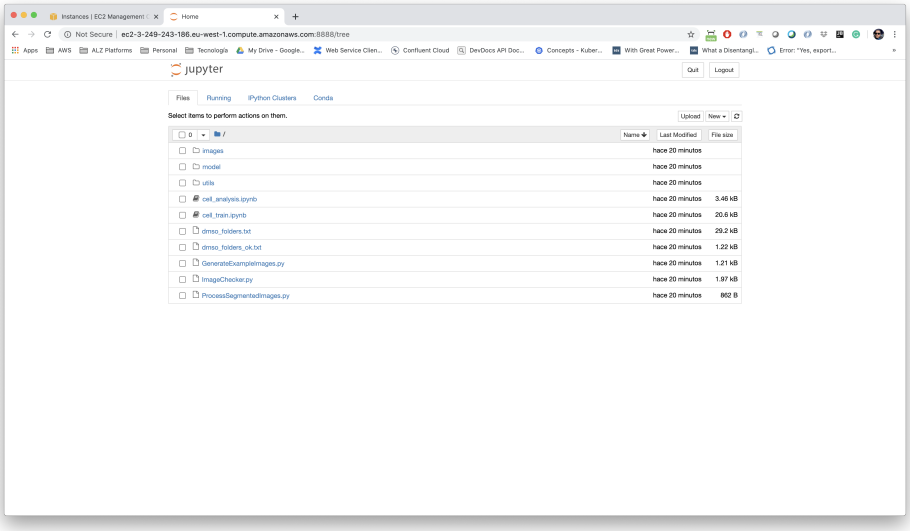

Figure B.3: Notebook home page

After selecting the kernel (conda_tensorflow2_p36) the training process can start. Image B.4 is the trainer. The **cloud_training** parameter has to be set to **True** and the folder `/run` has to be created manually in the file system. The execution can be performed step by step to show the results.

Train section runs the final steps and the process starts (see B.5).

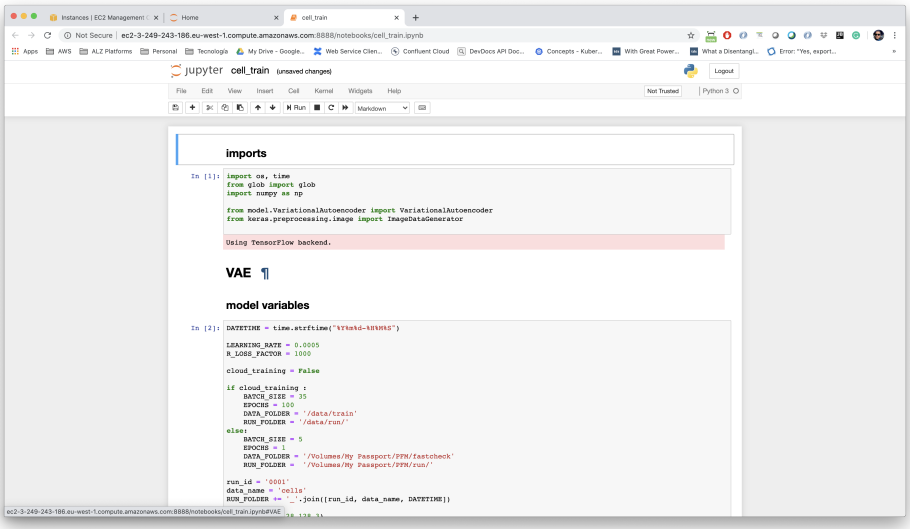

Figure B.4: Training notebook

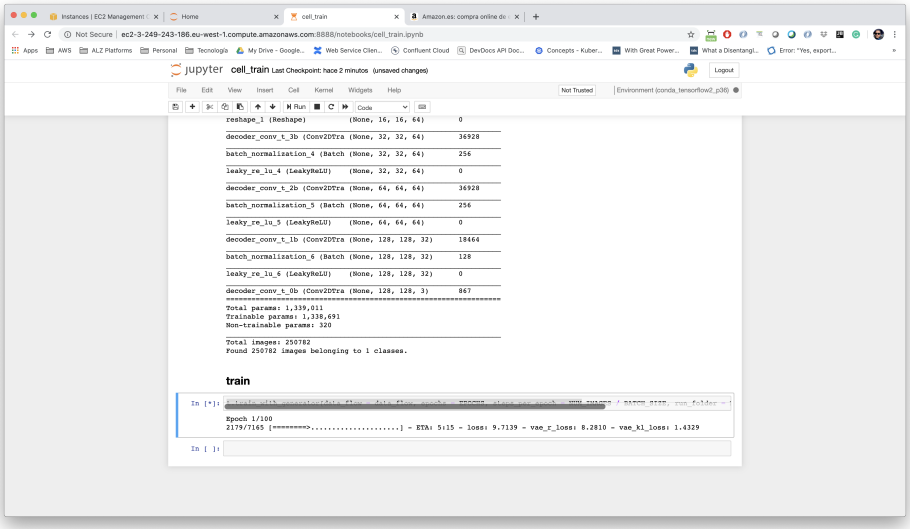

Figure B.5: Training process started

End of the process

When the process ends, a folder with the model has appeared in the /data/run folder. This data can be uploaded to a bucket (in this case segments-dmso-resized).

```
ubuntu@ip-10-0-1-78:/data/run$ aws s3 cp . s3://segments-dmso-resized/models --recursive
upload: 0001_cells_20200616-170048/images/Week1_22141_E11_3_Cells_199.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/Week1_
upload: 0001_cells_20200616-170048/images/Week5_29341_C02_2_Cells_161.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/Week5_
upload: 0001_cells_20200616-170048/images/Week5_28921_G11_2_Cells_121_001_0.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/
```

```
upload: 0001_cells_20200616-170048/params.pkl to s3://segments-dmso-resized/models/0001_cells_20200616-170048/params.pkl
upload: 0001_cells_20200616-170048/images/Week5_28921_G11_2_Cells_121.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/Week5_
upload: 0001_cells_20200616-170048/images/Week5_29341_C02_2_Cells_161_001_0.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/
upload: 0001_cells_20200616-170048/images/Week2_24121_D02_4_Cells_8_001_0.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/We
upload: 0001_cells_20200616-170048/images/Week7_34641_C02_4_Cells_253.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/Week7_
upload: 0001_cells_20200616-170048/images/Week7_34641_C02_4_Cells_253_001_0.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/
upload: 0001_cells_20200616-170048/weights/weights-001-5.95.h5 to s3://segments-dmso-resized/models/0001_cells_20200616-170048/weights/weights-001-5.95
upload: 0001_cells_20200616-170048/weights/weights.h5 to s3://segments-dmso-resized/models/0001_cells_20200616-170048/weights/weights.h5
upload: 0001_cells_20200616-170048/images/Week2_24121_D02_4_Cells_8.png to s3://segments-dmso-resized/models/0001_cells_20200616-170048/images/Week2_24
upload: 0001_cells_20200616-170048/images/Week1_22141_E11_3_Cells_1

(...)
```

Removing the environment

After the model has been uploaded, the environment can now be removed using terraform (local environment):

```
[jmunozal@MacBook-Pro terraform]$ terraform destroy
aws_iam_policy.policy: Refreshing state... (ID: arn:aws:iam::972975378845:policy/s3fullaccess)
aws_vpc.main: Refreshing state... (ID: vpc-0552c6ebd50ae682d)
aws_iam_role.s3_reader_role: Refreshing state... (ID: s3_reader_role)
data.aws_s3_bucket.a: Refreshing state...
data.aws_s3_bucket.b: Refreshing state...
aws_iam_instance_profile.modeller_profile: Refreshing state... (ID: modeller_profile)
aws_s3_bucket_policy.b: Refreshing state... (ID: tfm-images-cells)
aws_s3_bucket_policy.a: Refreshing state... (ID: segments-dmso-resized)
aws_iam_role_policy_attachment.test-attach: Refreshing state... (ID: s3_reader_role-20200616161158060500000001)
aws_security_group.allow_ssh: Refreshing state... (ID: sg-04050ba9890c2c990)
aws_subnet.main: Refreshing state... (ID: subnet-0d56ae97365f7f689)
aws_internet_gateway.gw: Refreshing state... (ID: igw-055305215203aaee0)
aws_default_route_table.r: Refreshing state... (ID: rtb-029e3fac77fcf1b67)
aws_spot_instance_request.dl_worker: Refreshing state... (ID: sir-5q1r5wsn)

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - aws_default_route_table.r

  - aws_iam_instance_profile.modeller_profile

  - aws_iam_policy.policy

  - aws_iam_role.s3_reader_role

  - aws_iam_role_policy_attachment.test-attach

  - aws_internet_gateway.gw

  - aws_s3_bucket_policy.a

  - aws_s3_bucket_policy.b

  - aws_security_group.allow_ssh

  - aws_spot_instance_request.dl_worker

  - aws_subnet.main

  - aws_vpc.main


Plan: 0 to add, 0 to change, 12 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value:
```

The system is totally destroyed after answering 'yes' to the tool.

B.4 Analysis

Download data and running Jupyter

The model is now available to be analyzed in a local environment. Firstly, it has to be downladed from the model's bucket:

```
aws s3 cp s3://segments-dmso-resized/models/0001_cells_20200616-170048/
  ./0001_cells_20200616-170048 --recursive
```

Accessing analysis

The file **cell_analsys.pynb** is the Jupyter NB that performs the analysis, and can be accessed from the page B.3. The execution step by step is analogous to the training, but performed in a local environment.

Now, we have to run Jupyter in local mode (launched from our **tfm-autoencoders/vae-images**). Values of local folders must be parametrized for our own values:

```
# os links
MODEL_FOLDER_A = os.environ.get('HOME') + '/modelA/weights'
MODEL_FOLDER_B = os.environ.get('HOME') + '/modelB/weights'
MODEL_FOLDER_C = os.environ.get('HOME') + '/modelC/weights'

FILE_MODEL = 'weights.h5'
DATA_FOLDER = '/Volumes/My Passport/PFM/output/training_png'
RUN_FOLDER =  '/Volumes/My Passport/PFM/run/'
```

Current implementation reconstruct random images from the folders and generates also new images.