University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Deep learning for analysing immune cell interactions

Leonore Papaloizos

April 4, 2020

Abstract

The protective responses of our immune system are initiated by interactions between immune cells. These interactions can be inhibited or enhanced by the application of drugs. This project investigated using deep learning models to analyse microscope images of immune cells to study their interactions under different experimental conditions. Deep learning has scarcely been applied in the field of immunology. We implemented and evaluated a convolutional autoencoder and an autoencoder-based regression model to qualitatively and quantitatively analyse the interaction between T cells and dendritic cells in microscope images. We found that the autoencoder helped speed up the process of data visualisation and that the regression model successfully predicted measures of interaction in unseen images of immune cells. With carefully selected and pre-processed datasets, deep learning can be a useful technique for immunology researchers to analyse immune cell interaction.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Leonore Papaloizos Date: April 4, 2020

Acknowledgements

I would like to thank Dr Carol Webster and Dr Hannah Scales for their continuous guidance and advice throughout this project. Thank you to all the friends and family that supported and encouraged me in the past four years.

Contents

1 Introduction

1.1 Motivation

Your immune system is your protective shield against pathogens. It functions by discriminating between what is part of your self, and what is not, and fighting what is alien to it.

Now picture your immune system as a speed date. Your immune cells go on quick dates with other immune cells, who tell them about their life. Immune cells might be under the influence of certain substances to various degrees. The success of the discussion between two immune cells during their speed date determines how your immune system is going to evolve as a whole. The consumption of substances might help make the situation more positive, or worse. Your immune system might be pleased with the date, and react positively, or an immune cell might get offended by its date, and trigger a negative response.

Indeed, the onset of an immune response in our immune system depends on the interaction strength between different types of immune cells. Certain types of immune cells relay information about their environment to other types of immune cells, which can then trigger an appropriate response depending on what they have learned from the other cell about the environment. The environment might contain substances like alien, dangerous bodies. These interactions between immune cells can be enhanced or inhibited by the application of drugs. Studying the reactions of immune cells under the influence of different types of drugs is key to the development of drugs for diseases such as viral infections, cancer or auto-immune diseases.

1.2 General problem and our idea

One way of studying the interactions of immune cells is through microscope images obtained in artificial settings. We can place immune cells in a dish and study them under a microscope under different experimental conditions, which could involve different types of drugs being injected into the dish. Microscope images of these cells can then be systematically captured.

We are however not directly interested in how these images are captured, but how they can be analysed. Microscope images are often analysed with proprietary software which is costly to maintain and whose inner workings are hard to understand or customise. On the other hand, applying deep learning techniques to biomedical data is becoming increasingly popular and more accessible, while delivering promising results. In the specific case of image analysis of immune cells, we want to explore how deep learning could be used to systematically extract metrics of immune cell interactions from microscope images. Specifically, the aim is to assess whether using deep learning in the field of immunology can provide useful information on interaction levels between immune cells under different experimental conditions.

2 Background

2.1 Immunology concepts

2.1.1 Our immune system

Our immune system consists of organs, cells and groups of cells working in collaboration to defend us from other organisms that could pose a danger to our health. Such outside forces could be harmful viruses, bacteria or parasites for example. The human body is a haven for these to thrive in, to our detriment. Our immune system comes into contact with many materials, which might be harmless (e.g. food, pollen, our own body) or harmful (e.g. a virus, a parasite). These materials are called antigens. Our immune system protects us by attacking these antigens when they are recognised as dangerous. The key in this exchange is for our immune system to recognise which biological entities are ours, and which are alien, potentially dangerous elements. In some cases, the immune system also makes the wrong decision. It can classify a harmless substance as dangerous, causing an auto-immune response such as allergies. It can also fail to respond to a harmful substance, for example in cases of cancer or vaccine failure (British Society for immunology 2020).

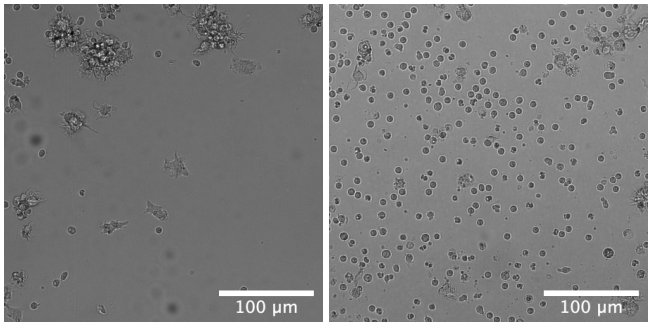


Figure 2.1: *Brightfield microscope images of dendritic cells (left) and T cells (right). Dendritic cells are named after their branching, tree-like structure. T cells are generally round.*

The actors of our immune systems are thus the key defenders of our bodies. The actors we are interested in for the purpose of this research are T lymphocytes – “T cells" – and dendritic cells – DCs (Figures 2.1 and 2.2). Dendritic cells are “sentinels" and initiate our immune system’s responses by sensing and integrating information from their environment and sending it over to T cells. T cells are “master controllers" and trigger the appropriate immune response, if any, from the information they have received, notably from dendritic cells, often in the form of chemical signals or intercellular interactions (Brewster 2015; Roghanian 2020).

Antigens can be fought by antibodies, which are defensive proteins produced by our immune system. More specifically, antibodies are produced by B cells in a process which starts in T cells, and in some cases is activated by T cells seeing antigens on the surface of dendritic cells (Benson et al. 2015). B cells need the support of T cells to make highly effective antibodies. Hence, the

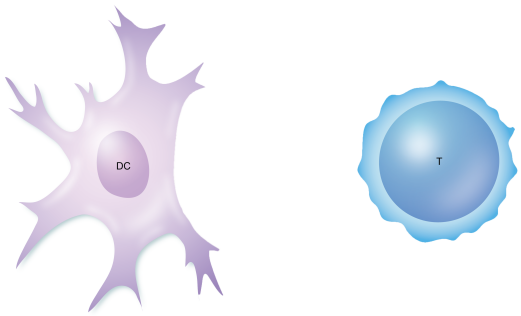


Figure 2.2: *Schematic model of a dendritic cell (left) and a T cell (right), highlighting the tree-like structure of dendritic cells and the roundness of the shape of T cells. Adapted from Cavanagh and Findlay (2020)*

interaction between dendritic cells and T cells is critical in the decision for our immune system to produce agents to defend our body.

The purpose of this dissertation is to evaluate how much interaction is observed between immune cells. There is existing work in the field of immunology looking into the effects of these changes in interaction. Benson et al. (2015) show how the generation of antibodies might be impacted by T cell and dendritic cell interaction. They studied how dendritic cells and T cells interacted in the mouse immune system, both in terms of whether or not interaction was witnessed, and of duration of interaction. This interaction was studied under different conditions, with different drug compounds being used to attempt to drive interaction or to inhibit it. They found that under conditions where compounds were blocking interaction between T cells and DCs, fewer antibodies were generated, meaning that the mice were not defending themselves as much. Hence, the study of the impact of compounds on the interactions between immune cells can tell us how our immune system will then operate.

2.1.2 Implications

Concepts and research described in this section show that changes in interactions between immune cells control the way in which our immune system protects itself. Understanding how interactions react under the control of different drugs can give us targets for new therapies. Hence, analysing the interaction between immune cells under different experimental conditions bears a particular interest in the field of immunology for studying immune responses. We want to analyse this interaction with the help of deep learning techniques.

2.2 Concepts of interest in deep learning

The following sections collate selected research that show how deep learning techniques could be applied in the context of our study.

2.2.1 Convolutional operations for image feature extraction

Convolutional operations in neural networks were first introduced for pattern recognition by Fukushima (1980). They were later popularised by LeCun et al. (1989) as a method for object recognition, once back-propagation was put to use as a learning procedure for networks. LeCun applied his convolutional neural network (CNN) to digit recognition and subsequent classification and explored these networks in multiple papers. Since then, convolutional operations in neural networks have proven successful to extract features from more complex images. The

AlexNet model published by Krizhevsky et al. (2012) popularised the use of CNNs employing the acceleration of GPUs in computer vision. Rawat and Wang (2017) provide a comprehensive review of deep convolutional neural networks applied to the general task of image classification. In a recent medical example, Shen et al. (2019) trained a convolutional neural network structure to detect breast cancer from mammography screenings which showed competitive results compared to commercial systems.

2.2.2 Autoencoders for dimensionality reduction

An autoencoder is a type of neural network trained to map its input to itself via a compressed representation of the input, as shown in Figure 2.3. The compressed representation of the input obtained from the bottleneck layer is a *coded* representation of the input, while the final output of the network is the *decoded* version of the input. Autoencoders are not trained to learn a perfect copy of the input data, but a smaller, compressed copy with features which the neural network learns to be most important to be able to gain an overall understanding of the input. Autoencoders were first introduced in the 1980s (Rumelhart et al. 1986) and are traditionally used for dimensionality reduction and feature extraction (Goodfellow et al. 2016).

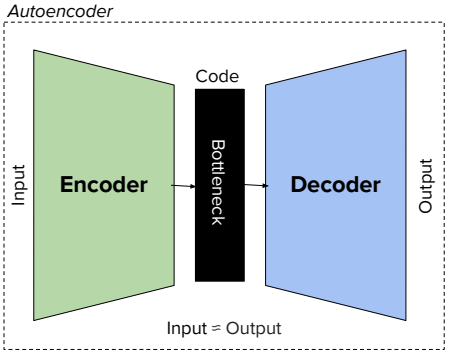


Figure 2.3: *Schematic representation of an autoencoder. An autoencoder consists of two blocks of layers: encoding layers and decoding layers. The bottleneck layer holds a smaller representation of the input created by the encoder layer, from which we try to recreate the input in the decoder layers.*

Zamparo and Zhang (2015) show that autoencoders can be successfully applied for dimensionality reduction in the context of biomedical data. Their autoencoder approach, applied to the unsupervised clustering of cell phenotypes, outperformed other dimensionality reduction techniques such as Principal Component Analysis. However, this approach was not applied to imaging data. Nonetheless, autoencoders have been successfully used for reducing the dimensionality of large imaging data by using convolution operations in their structure (Wang et al. 2016; Saenz et al. 2018).

2.2.3 Deep regression models

Neural networks can be constructed for regression tasks such that the model is trained to map from an input data, e.g. images, to real-values from a continuous range. Lathuilière et al. (2018) provide a review and comparison of common regression network architectures. In a medical example, both Xie et al. (2015) and Xue and Ray (2017) show promising results for using CNNs to extract numerical features from images of cell by using neural networks for the regression task of counting the number of cells in an image.

2.3 Finding structure in high-dimensional data

The data we will be studying consists of images of cells obtained through high content screening (HCS). HCS is a method for capturing images of cells in multi-well plates, using high-resolution microscopy (Buchser et al. 2014). A plate captured with high content screening can yield a large number of images in very high-resolution, typically around 2000 × 2000 pixels. This makes the analysis of the physical characteristics of a cell possible at a granular level. However, this also makes the dataset high-dimensional, which requires the use of visualisation techniques that map high-dimensional data points to a low-dimensional plane if we are looking to gauge the structure of the data. In this section we highlight two commonly used techniques for high-dimensional data visualisation.

t-distributed stochastic neighbor embedding (t-SNE) was developed in 2008 by van der Maaten and Hinton as a technique to map high-dimensional data to two- or three-dimensional space. t-SNE can find structure in high-dimensional data by using the local relationships between data points and optimising results using gradient descent. These local relationships are defined using a Gaussian probability distribution in high dimensional space, and then recreated using the Student t-distribution. Wattenberg et al. (2016) provide a comprehensive guide on understanding the inner workings of t-SNE.

Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) is a dimensionality reduction technique first published in 2018 by McInnes and Healy. It has shown competitive results compared to t-SNE. UMAP works by constructing a high-dimensional weighted graph representation of the data. Each edge between points in the graph is weighted according to how likely the points are to be connected. UMAP transforms the high-dimensional graph representation into a low-dimensional representation that is as similar as possible, optimising results in the same way that t-SNE does. Coenen and Pearce (2019) provide a similar guide as for t-SNE above for grasping the inner workings of UMAP.

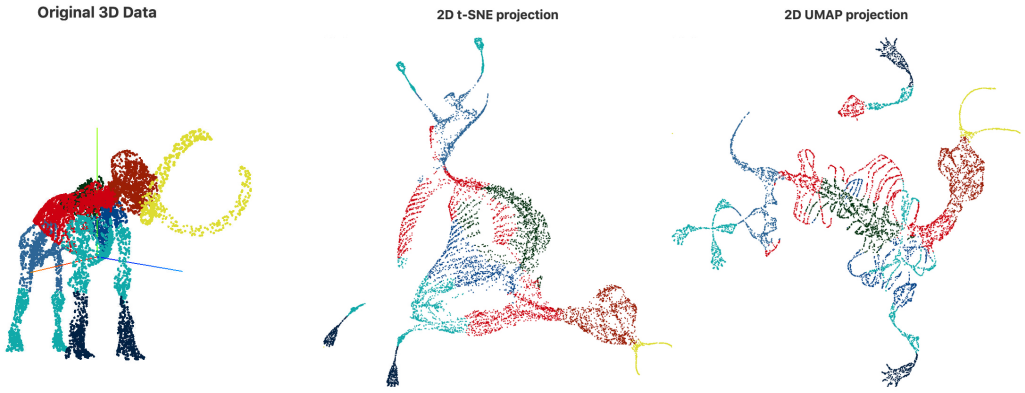


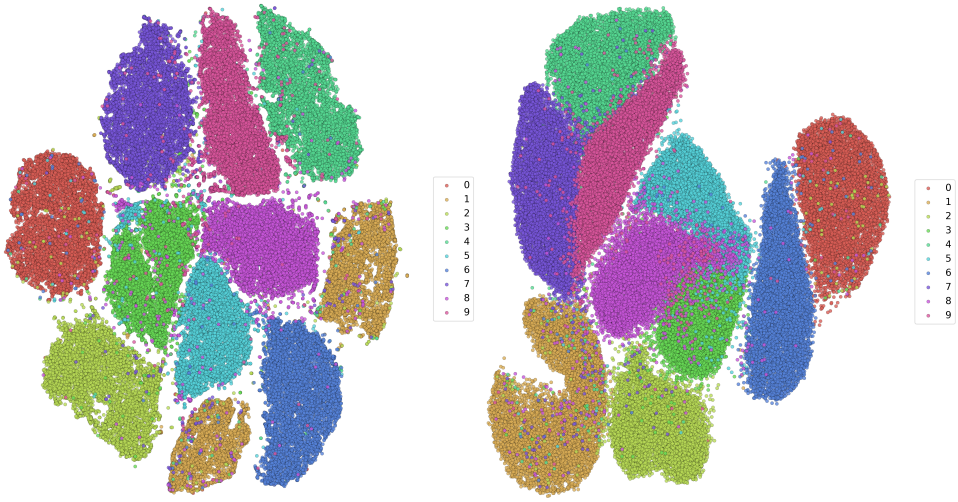
Figure 2.4: *Three-dimensional model of a mammoth (left) projected to a two-dimensional plane with t-SNE (middle) and UMAP (right). t-SNE was applied with* `perplexity` = 2000*. UMAP was applied with* `n_neighbors` = 200 *and* `min_dist` = 0.25*. Source: Coenen and Pearce (2019)*

The main differences between t-SNE and UMAP are of speed and parameters. The original UMAP paper compares UMAP's performance with t-SNE's on the MNIST digit dataset which consists of 70,000 28 × 28 images of digits (0-9). On the researcher's 2017 MacBook Pro with i7 core and 8 GB of RAM, UMAP takes 87 seconds to run, while t-SNE takes 1,450 seconds (McInnes and Healy 2018).

t-SNE's main parameter to be tweaked is 'perplexity', which loosely corresponds to an estimate of the number of neighbours each data point has (Wattenberg et al. 2016). UMAP's main parameters

are number of neighbours and minimum distance. The former corresponds to the number of approximate neighbours a data point has, similar to t-SNE's perplexity. The latter corresponds to the minimum distance between points in low-dimensional space, meaning that it will tell UMAP how tightly to cluster points together, making visualisation more flexible.

The capabilities of t-SNE and UMAP are best illustrated through examples as shown in Figures 2.4 for a 3D to 2D projection and 2.5 for a projection from higher high-dimensional points. It is important to note that in both techniques, distances between points in lower dimensions do not necessarily reflect their distances in higher dimensions (Coenen and Pearce 2019).



(a) *t-SNE projection* ***(b)*** *UMAP projection*

Figure 2.5: *70,000 points of the MNIST digit dataset images projected to a two-dimensional plane with t-SNE (left) and UMAP (right). t-SNE was applied with its default parameters and* `perplexity` = 30*. UMAP was applied with* `n_neighbors` = 50 *and* `min_dist` = 0.65 *to have clusters of similar sizes as t-SNE. Each point corresponds to the image of a handwritten digit. We can see that same-digits are generally clustered together with both techniques.*

In the case of our research, applying t-SNE or UMAP to a dataset of microscope image could allow us to uncover whether or not immune cells behave in recognisable ways under different experimental conditions and whether or not this can be recognised from the structure of the images.

2.4 The place of deep learning in immunology

A quantity of existing research already uses broader machine learning (ML) techniques in the field of immunology. Muh et al. (2009) applied Support Vector Machines (SVMs) to the study of allerginicity. Allergic reactions are triggered when the immune system wrongly assumes a harmless substance to be dangerous, such as dust, and produces antibodies to attack it (Tregoning 2020). The SVMs were used to analyse the DNA sequences of known allergens and known non-allergens. The aim was to try and make accurate predictions on previously unseen sequences and classify them as either allergenic or non-allergenic. The model achieved 95.3% accuracy. In another classification example, David et al. (2010) used a Bayesian classifier and a decision tree to predict the likelihood of degenerative disorders from the sequencing of antibodies and achieved a best accuracy score of 89%.

The above examples represent examples of immunology research carried out using traditional ML

techniques. Emerging deep learning research on cell imaging data now additionally includes cell segmentation (Al-Kofahi et al. 2018). In cancer research, deep neural networks are increasingly being used for feature extraction from images to accurately detect cancer (Litjens et al. 2016; Bychkov et al. 2018). Research specifically using immune cell data is mostly focused on cell counting (Turkki et al. 2016; Aprupe et al. 2019).

2.5 Summary

The research presented and cited here highlights that there is an increasing array of methods available to process high-dimensional, visual data through deep learning and visualisation techniques. This section has shown that immunology researchers have successfully made use of machine learning techniques, and that immunology-related fields such as cancer research have successfully applied deep learning methods in their research to obtain promising results. There is indeed an increasing interest in the applications of deep learning in medical fields, but the use of deep neural networks has not been fully explored in the context of studying immune cells interactions. There seems to be a lack of research into the qualitative and quantitative analysis of immune cell interactions from imaging data through deep learning. However and as shown above, immune cell interactions are of particular interest in drug research as they are key in understanding how an immune system operates.

This dissertation will focus on filling this gap, applying deep learning to extract features from images of immune cells in order to generate qualitative or quantitative data about the interactions between T cells and dendritic cells under different experimental conditions.

3 Materials and Methods

This chapter covers the images that were available for analysis, how they were processed, and which methods were applied to analyse them. Detailed results of the implementation of these methods are then presented and discussed in Section 4.

3.1 Immune cells dataset

3.1.1 Setup

The images that were used for the purpose of this research were provided by the Laboratory of Immune Cell Visualisation and Examination (LIVE) at the Institute of Infection, Immunity & Inflammation at the University of Glasgow. The images are captured from 384-well plates with a commercial INCell Analyzer 2000. As established in Section 2.1.1, the type of immune cells we are studying are T cells and dendritic cells (DCs). Each plate to be imaged in the INCell Analyzer contains a grid of wells as shown in Figure 3.1. Each of those wells is assigned a label and an experimental condition. T cells, dendritic cells, and compounds related to the experimental conditions are injected in the well. In order to be able to distinguish between them, cells are loaded with fluorescent dyes: the T cells are dyed with a green dye (CSFE dye), and the dendritic cells are dyed with a red dye (CMTPX dye). After imaging, we obtain three field-of-view images per well:

- a Brightfield image, which shows both T cells and dendritic cells (Figure 3.2a)
- an image showing only the T cells, which has been captured through the fluorescent green dye (Figure 3.2b)
- an image showing only the dendritic cells, which has been captured through the fluorescent red dye (Figure 3.2c).

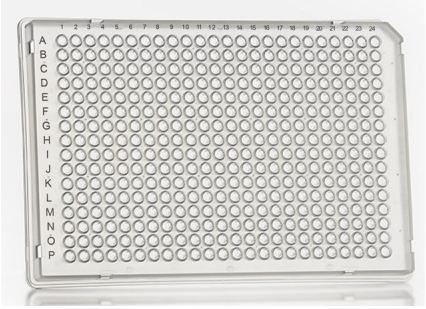
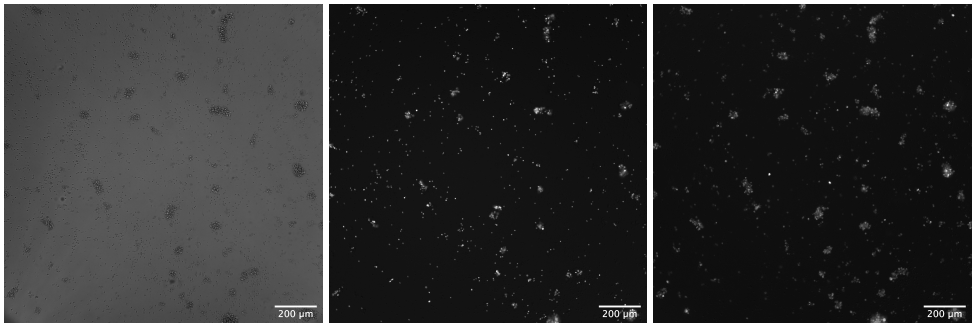


Figure 3.1: *Example of a 384-well plate. Each well is labelled by a letter (vertically) and a number (horizontally). Source: Brooks Life Sciences.*



(a) Brightfield view *(b) Green dye (T cells) view* *(c) Red dye (DCs) view*

Figure 3.2: *Microscope images extracted from the dataset showing each view obtained with the capture system. Brightness has been adjusted for (b), (c) for details to come through.*

3.1.2 Experimental conditions

Approximately 8,000 dendritic cells and 8,000 T cells were cultured in each of the 384 wells. The INCell Analyzer 2000 captured an image of 25% of each well. The dendritic cells were cultured from bone marrow cells obtained from mice. T cells were obtained from mouse lymphoid tissues. About 70 to 90% of mouse T cells expressed a receptor for the Ovalbumin (OVA) peptide antigen, which means they recognised it well.

Each well in a plate was associated with an experimental condition:

- The well are injected with one of 45 drug compounds,
- These compounds could be injected at a bigger or lesser concentration,
- Moreover, one of two types of antigens could be supplemented to further stimulate the cells.

We were interested in using the latter as a label for our images, as it gives us a smaller number of categories to look at. More specifically, there are three categories of stimulation:

- No stimulation
- Stimulation with Ovalbumin (OVA) peptide, which is the antigen that should be recognised by most T cells in the well
- Stimulation with Concanavalin A (ConA), an antigen which cross-links T cells receptors with a molecule on the surface of DCs that would hold the OVA peptide antigen and present it to T cells

3.1.3 Image selection

There was a large amount of images available from different well plates with the different experimental conditions described above. However, each set of images corresponding to a plate represents about 8 GB of data on average. Moving images through disks or cloud filing system thus represented substantial time and was vulnerable to transfer errors. Hence, a limited number of plates were selected for training and evaluation to make sure their consistency could be validated. Chosen plates were selected to best represent the experimental conditions were chosen for study, and were assigned to three datasets.

- **The 'full' dataset**: this dataset contained an equal number of images in the three categories of stimulation: no stimulation, stimulation with OVA, and simulation with ConA. This was to prevent issues of class imbalance when training the model.

- **The simpler 'dual' dataset**, with two categories: this dataset contained an equal number of images in two categories: no stimulation, and stimulation with OVA peptide. This was selected under the hypothesis that if useful results cannot be obtained with a dataset containing images from all categories, a deep learning model might be able to perform better with two categories.
- **The 'DMSO-only' dataset**: Dimethyl sulfoxide (DMSO) is a solvent that helps solubilise the investigated drug compounds, as most compounds are not initially water soluble. When in solution, drug compounds should be more bioavailable and have more of a biological impact. DMSO images in the original dataset only included DMSO and no drugs, and were used as a control. However, images of this dataset should show the most difference between stimulation categories as immune cells have not been impacted by drugs.

These datasets and characteristics are summarised in Table 3.1.

Table 3.1: *Number of images in each category of stimulation for each of the three datasets. Both full and dual datasets should have no issues with category imbalance.*

Dataset	Unstimulated	OVA	ConA
Full	9,800	9,800	9,800
Dual	6,900	6,900	0
DMSO	4,000	2,000	2,000

3.1.4 Pre-processing

The datasets obtained from this setup consisted of 2048×2048 16-bit images in TIFF format, captured by a 12-bit camera. As mentioned above, each "image" consists of a set of three views: the green fluorescent image (T cells), the red fluorescent image (dendritic cells), and the brightfield image. The brightfield images show both cell types and can be used for diagnostic purposes when deciding on pre-processing methods, but were otherwise discarded and not used for further analysis.

Each of the images was about 8 MB in size, representing 4,194,304 pixels. This raised the issue of very high dimensions to handle for a model. Moreover, each plate had 384 wells, which corresponds to a total of about 800 images when counting both the T cells image and the DCs image. This meant that we had quite a limited amount of images to feed into any kind of model.

Furthermore, as shown on Figures 3.2b and 3.2c, the smaller white dots of immune cells could easily be confused for dust on the screen, even to the naked eye. This could be as confusing for a deep learning model trying to learn features from an image as it could be for us. Furthermore, the issue of images being of very high dimensions remained. Indeed, a basic autoencoder with three 2×2 downsampling operations would yield around 500,000 pixel points from a 2048×2048 input image, which is still a very high number for a visualisation technique like t-SNE or UMAP.

Sliding window

To resolve this issue, the first idea was to make the images more manageable by a neural network by cropping a set square subsection of the image of smaller dimensions, for example a square of 250×250 pixels. However, this left the issue of having only limited input to train a neural network. Instead, images were pre-processed by passing a sliding window over the image, creating patches of images per file. This quickly expanded the size of the dataset, making it as large as 58,000 samples in some cases. Smaller images also made more sense to the naked eye, hence we assumed that a trained neural network would perform better on this gridded dataset than on a full-image dataset.

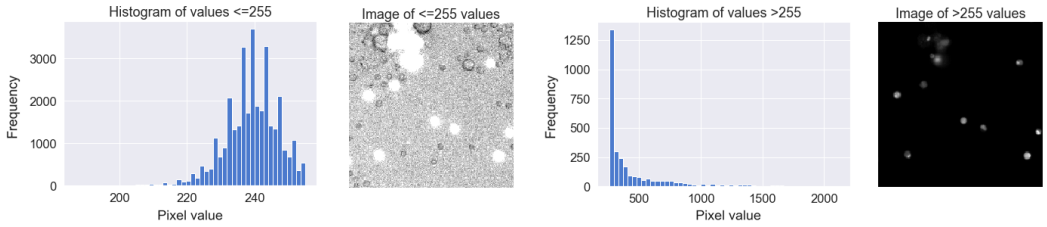
Normalisation

Each sub-image was normalised with min-max normalisation (Equation 3.1) to get a [0, 1] range of pixel values. It is common to normalise imaging data to the floating-point [0, 1] range over keeping an integer representation of an image, as integers are subject to saturation and rounding error. We used min-max normalisation as not all images in our dataset had 0 as their minimum value.

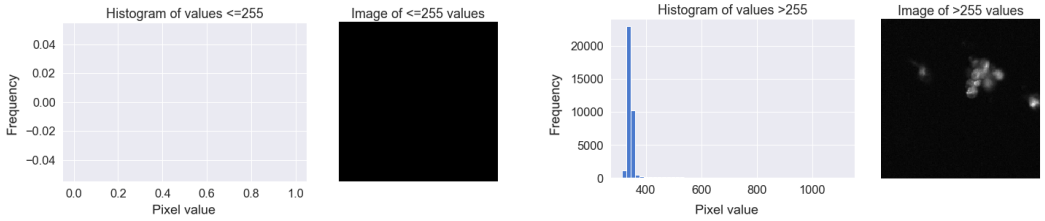
$$minmax(x) = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

Noise and outlier detection

From analysing the images, it was also found that some images contained many pixels in the range of [0, 255] before normalisation. Moreover, such pixels usually seemed to correspond to background noise as shown in Figure 3.3. The fact that these lie in the range of an 8-bit image pixel range could be coincidental, or the result of the way the imaging system picked up non-stained elements in the well. The immunology researcher providing the images confirmed that this noise was present in some images read from specific plates. Wells in early experiments were injected with additional cells as researchers thought this made the cells behave more naturally, however this was later found to have no real impact. These cells were not stained by dyes, but some of their details still came through the imaging system. Although this background was not always visible to the naked eye, we did not want those pixel values to confuse a neural network model. Hence, values below 255 in images were clipped to 255 before min-max normalisation was applied.



(a) Histogram and image analysis for a sub-image with noisy cells in the background

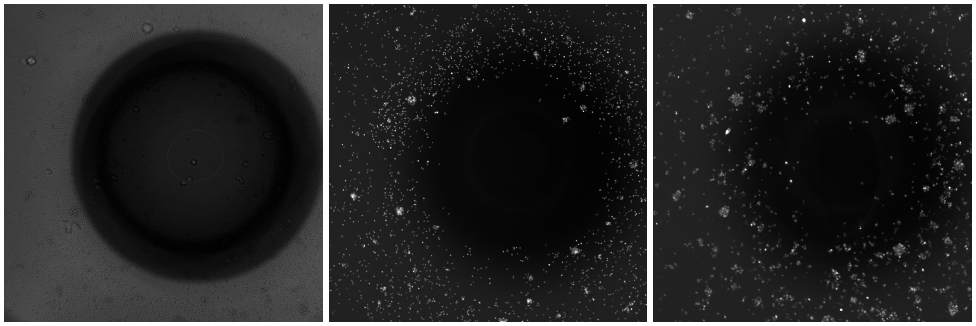


(b) Histogram and image analysis for a sub-image with no noisy cells in the background

Figure 3.3: *Histogram and image analysis for background noise detection*

As described above, the provided images sometimes contained some background noise. Furthermore, some images contained larger amounts of noise coming from defects in the wells, such as water droplets. This is illustrated in Figure 3.4. Their pixel value distribution followed the [0, 255] range as described above, but these values also covered the whole sub-image and no cells of interest were present in those patches. Removing them entirely from the dataset would have made it more difficult to rationalise the analysis of whole images. Indeed, by pre-processing

full images with a sliding window, each full image is represented by a patch of a set number of sub-images. This would have created inconsistencies in the amount of sub-images per image file. Instead of removing them, it was decided to keep them in to evaluate if a neural network could make sense of them as a category. Noisy sub-images were labelled as “Faulty".



(a) *Brightfield view* ***(b)*** *Green dye (T cells) view* ***(c)*** *Red dye (DCs) view*

Figure 3.4: *Different views of an image which contains “Faulty" patches. The fault is likely coming from a drop of water in the well. (b), (c) have been brightness-adjusted for visualisation.*

Labelling

Each plate came with an Excel sheet giving information about plate layout. Each well was assigned a letter and a name, and the Excel sheet contained information about the stimulation, drug compound ID number, and compound concentration used for each well. The filename of each image of immune cells was formatted by plate number, and letter and number associated to the image’s experimental condition. We could link this information to the Excel sheets. However, the sheets were not automatically parsable, because types of drugs or placements in the sheet might vary from one plate to the next, hence labelling had to be hardcoded and handchecked by plate.

3.1.5 Combining images to qualify interaction

To summarise the section above, we made sure that for each well representing an experimental condition, we obtained two images of interest: an image of T cells, and its counterpart image of dendritic cells. While these were obtained separately with the help of fluorescent dyes, they were still captured from the same well in which they were placed together. Hence, for us to gain any understanding of cell interaction from these images, we needed to combine them one way or another.

We decided to combine each black-and-white T cell and DC image in one RGB image. Computationally speaking, an RGB image is represented by a multi-dimensional array of three arrays. Each of these arrays corresponds to a colour channel: red, green, or blue. The dendritic cell images were obtained through fluorescent red dye screening, so the red channel of the image was set to this image. Similarly, the T cell images were obtained through fluorescent green dye screening, so the green channel of the image was set to this image. The blue channel of the RGB image was left blank. The resulting RGB image thus allowed us to visualise T cells in green, DCs in red, with close overlap between those cells in orange hues. Figure 3.5 illustrates a sample of combined sub-images after this operation was completed.

This combination allowed us to visualise areas of proximity in cells as well as areas of overlap. As such, this visualisation provided us a means of qualifying interaction between the different types of immune cells.

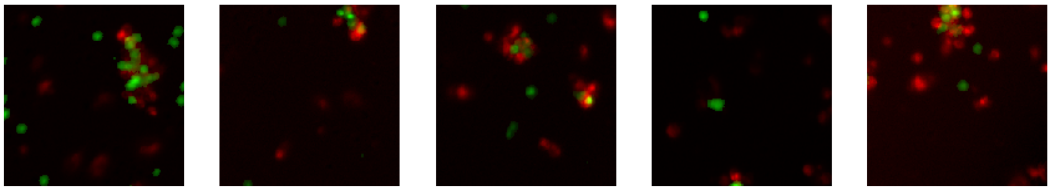


Figure 3.5: *Random sample of five RGB images of immune cells, where the red channel contains the dendritic cells and the green channel contains the T cells. We can observe areas of interaction where there are hues of orange.*

3.2 Image segmentation

Image segmentation refers to the process of separating out the parts of interest of an image into different 'segments' or objects. In our context, applying image segmentation to our dataset had two purposes. The first was to use image segmentation as a method to separate the background from the cell objects for background correction. The second was to use it to obtain binary objects of the cells in order to compute numerical data about cells in the images.

3.2.1 Background correction

Although we have defined a method to perform some background noise reduction in images in Section 3.1.4, this method might not be able to catch all irregularities in the background. As such, we wanted to be cautious and explore some alternatives.

All the images in the original, uncombined dataset were black and white, with the details of interest (i.e. the cells) in bright white spots. However, as discussed in Section 3.1.4, the images could contain noise coming from grey details in the image that the naked eye cannot see immediately. Moreover, our noise removal methods might not be foolproof. There might be some noise remaining that could influence how a model learns. The best way for a model to learn about cells only is to only provide pixel values for cells, with a background sent to 0. Hence, we needed a method that will separate out the cell pixels from the background. We achieved this by obtaining a binary mask of the image, such that the white pixels of the binary image corresponded to the cells, and the dark pixels corresponded to the background. We tested two methods for doing this: k-means and thresholding, which will be described further in Section 4.

Once a binary mask was obtained through these methods, we removed the background of the original image by multiplying it with the mask, a procedure known as *masking*. If the binary mask is satisfactory, the output of this image should only contain the cells, which will have kept their detail, while the background will have been blacked out.

3.2.2 Quantifying interaction

The process used in subsection 3.1.5 describes a way of visually qualifying interaction between immune cells from combined images. However, we are also interested in quantifying interaction and obtaining a numeric value for the interaction between the T cells and the dendritic cells.

We can use the same method as described above to obtain the mask of the cell objects in each image before combination. This pair of masks can be used for further calculations. A common metric for evaluating image segmentation quality is **intersection-over-union (IoU)**, also known as the Jaccard index (Rahman and Wang 2016; Rezatofighi et al. 2019; Beers et al. 2019). This metric is normally used to evaluate how successful the segmentation of an object in an image is by comparing the segmented object to its ground truth value. It is computed from binary

objects as shown in Equation 3.2. The process of calculating overlap in our dataset is illustrated in Figure 3.6.

$$IoU(X, Y) = \frac{X\&Y}{X|Y} \tag{3.2}$$

In this case, we can use the IoU as a metric for area of overlap between two separate cell objects: the T cell objects and the dendritic cell objects obtained from the same sub-image. We can use the concept of overlap to quantify the level of interaction between cells. These overlap numbers will be used as label input to a deep regression model. Each pair of images which will be combined will be associated with an overlap value. The aim will be to evaluate whether we can train a model to predict a numeric value of interaction from an image.

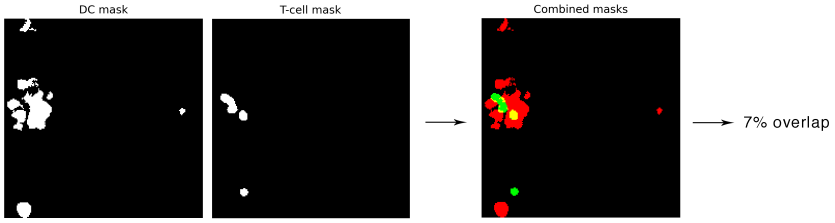


Figure 3.6: *Example calculation of the numerical overlap between two sets of immune cells. The overlap is visually represented in yellow. In this example we obtain a 7% overlap with intersection-over-union of the two masks.*

3.3 Deep learning models

The following two sections highlight what kind of deep learning models we used and their specific purposes of in the context of our research. Their general structure is highlighted in Figure 3.7.

3.3.1 Autoencoder for visualising high dimensional data

As described in Section 2.2, autoencoders are a particular type of neural network built with symmetrical layers around a bottleneck. The aim of an autoencoder is to map an input to itself as close as possible, while reducing its dimensions.

Different types of autoencoders exist. As our input consists of images, we set ourselves to implement a convolutional autoencoder. The hope was that if an image is reduced to a certain number of dimensions through convolutional neural networks, and another convolutional neural network is able to reconstruct the original image very closely just based on that compressed representation, then that smaller code representing an image is a good enough representation of the original input that can be fed into other models or algorithms.

The first aim for our autoencoder was to reduce the dimensions of our image input, as the size of the data points made it slow for data visualisation techniques to process. Furthermore, we hoped that the most important and distinguishing features of an image would come through the neural network layers and be collated in the bottleneck layer of the model.

High dimensionality visualisation techniques such as t-sne and UMAP can help visualise if there is an inherent structure to the data. We wanted to use this as a tool for analysis of immune cells interaction. We decided to use UMAP as our visualisation tool as both techniques yield similar structures as shown in section 2.3, but UMAP has been shown to outperform t-SNE time-wise.

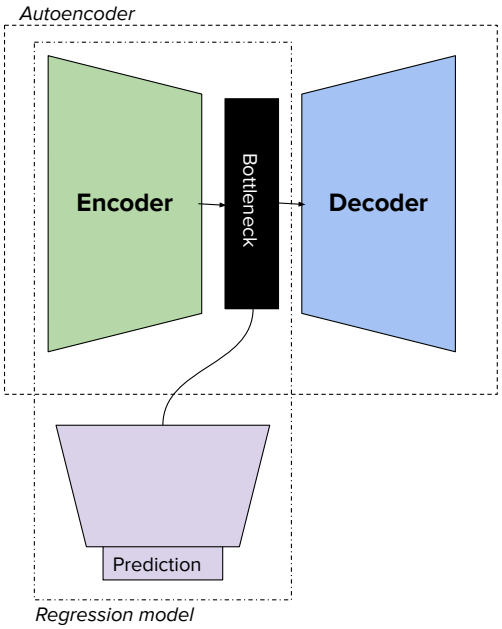


Figure 3.7: *Schematic diagram of the models to be implemented. The encoder layers will transform the input into a smaller coded representation, from which the decoder layers will create a reconstructed version of the image. The encoder layers will then be connected to a regression model, from which we hope to make predictions on how much interaction is perceived in an image. The bottleneck representation will also be fed into the UMAP algorithm.*

This also allowed us to experiment with UMAP's different parameters more easily, and to observe different structures of clusters in the data.

We wanted to be able to qualify interaction only from an image, without our model being supervised with labels about the image's associated experimental condition. This could be evaluated from a UMAP projection and whether or not clusters were formed around the same experimental conditions. If such successful groupings were found, it would allow us to show that different experimental conditions yield structurally similar cell interactions.

3.3.2 Regression model for quantifying interaction in unseen images

We also developed a deep regression model. Our regression task was to predict overlap values on unseen images. We used our autoencoder as a building block for a deep regression model. Again, we hoped that the autoencoder would successfully extract the most important features from the images through convolution operations and this would be a good starting block.

We used the encoder block of the model and extended it in a regression structure with fully connected layers. This regression model took images and associated overlap values as input for training. We could then assess whether or not the model successfully predicted interaction values from an image of T cells and dendritic cells. We would also be able to evaluate whether this was a faster approach than segmenting cells and calculating their overlap. Similar regression models could also be trained with other interaction metrics easily.

4 Implementation

All development of the methods detailed in this section was done using Python.

4.1 System diagram

Figure 4.1 can help the reader gain a better understanding of how each of the materials and methods used as described in Section 3 fit together.

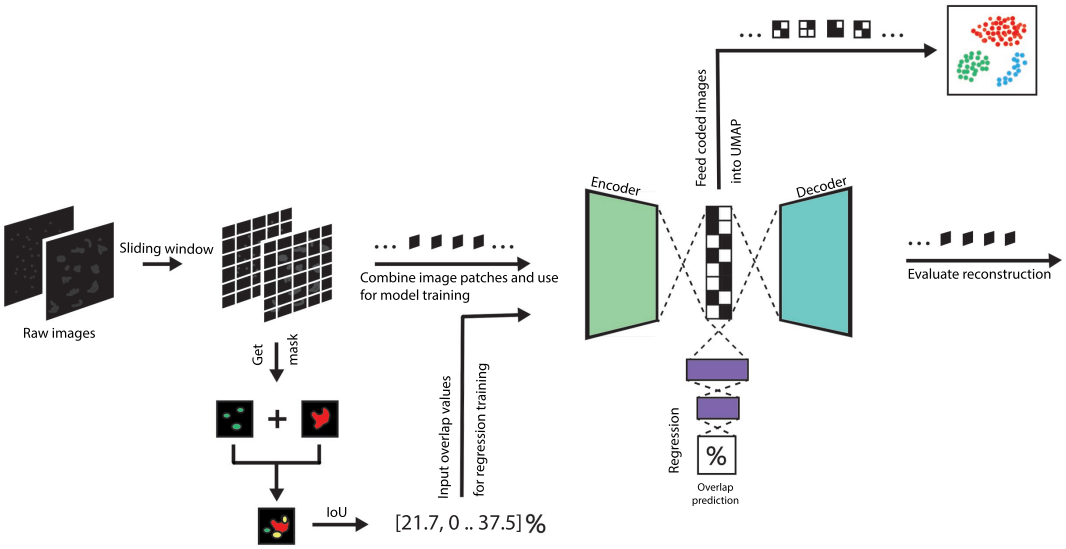


Figure 4.1: *System diagram showing the project workflow and how each image is decomposed and analysed. The images are transformed into smaller patches, from which we can gain overlap labels. The images patches are then combined and used for training. On one branch, these images will be fed into an autoencoder to accomplish dimensionality reduction and evaluate the projection of this smaller data in a two-dimensional plane. On the other branch, the images will be fed in a regression model with the overlap labels in order to train a deep neural network for a regression task.*

4.2 Pre-processing

Pre-processing steps are best explained through the diagram shown in Figure 4.2.

Images were processed with a sliding window of size 192×192 as we wanted a window that was large enough to contain a few cells but also wanted to make it smaller to be able to reduce individual image dimensions further. A sliding window of 192×192 yielded 100 patches as our images were originally of size 2048×2048. This meant the left and right border of the full image were cropped out. We used NumPy's compressed *.npz* file format for writing and reading the

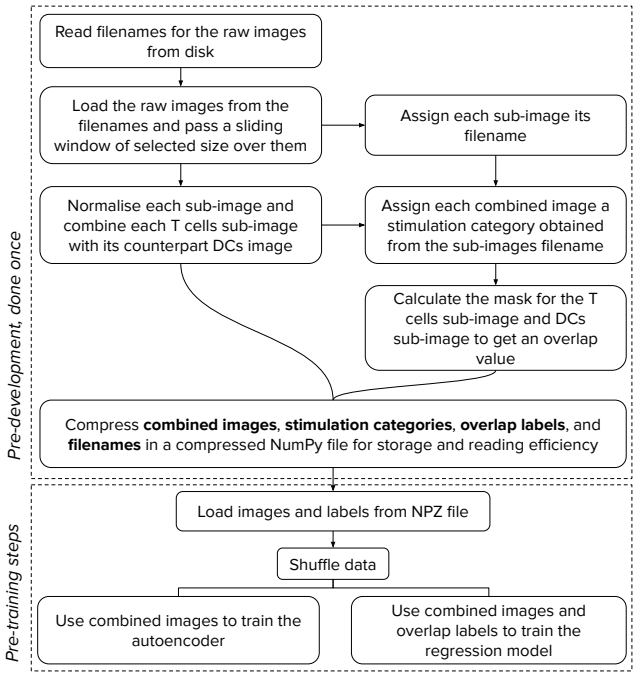


Figure 4.2: *Diagram illustrating the different steps taken for pre-processing our images. We pre-process the full images into sub-images, normalise them and combine them. We compute the overlap labels for each combined image from the mask of the T cells image and the dendritic cells image. We store the combined images and their labels in a compressed .npz file. When needed, the combined images are loaded and used to train the autoencoder model. The regression model is also trained with the overlap labels. All data is shuffled before training to avoid order bias. Stimulation categories are used post-training for evaluation.*

images as they were computationally represented as multi-dimensional arrays of floating point values and it made loading the arrays much faster. It also reduced storage space by a factor of 2.5 on average.

4.3 Image segmentation

As discussed in Section 2.4, performing cell segmentation from greyscale microscope images of cells has been researched and is necessary when an image contains different types of cells which have not been separately labelled in one way or another. In our case, cell segmentation did not need to be applied as the images of T cells and dendritic cells have been captured separately through fluorescent dyes. Instead, we were interested in using segmentation techniques to obtain the mask of each type of immune cell for the purposes of background correction and estimating interaction, as explained in Section 3.2.

As the immune cells in our images were bright blobs on a dark background, we hoped that obtaining masks from each image would be straightforward. This section describes the methods we explored for this task. Both k-means and thresholding methods yielded good results and their specifics are detailed below.

Furthermore, we also explored the use of a deep learning model which was developed with the purpose of segmenting biomedical data: the U-Net model.

4.3.1 *k*-means colour clustering

k-means has been shown to perform well on image segmentation by quantifying the number of colours in an image into k clusters (Ng et al. 2006, for example). Formally, k-means aims to partition data points in an array into k sets such that the variance between points within clusters is minimised. In our case we wanted to use k-means to transform our greyscale images of immune cells into bichrome images that we could use as masks. The following pseudocode details the process of k-means.

Data: I, an array of pixel values making an image.
k, the number of colours to partition the image's colour palette to.
Result: A set of k clusters.
begin
    Initialise k objects picked from I
    while *clusters are still changing* **do**
        Assign each item i in I to the cluster with closest mean value
        Recompute the mean (centroid) of each cluster
    end
end

Algorithm 1: Pseudocode for the k-means algorithm applied to image segmentation.

k-means clustering is conveniently offered by multiple libraries in Python. We looked at both scikit-learn's and OpenCV's k-means. scikit-learn is a general library for machine learning tools, while OpenCV is a more specialised library built for computer vision purposes. Both their k-means functions are straightforward to initialise and use. Their performance was benchmarked in order to select the best one. The table below reports times for k-means with k=2, 10 iterations, and each of the random and optimised methods of initialising centroids.

Table 4.1: *CPU times for OpenCV's and scikit-learn's k-means tool ran on 1,000 samples of* 192×192 *pixels with different methods of initialising centroids. The computation was ran on a 2015 MacBook Pro with 2.7 GHz i5 core and 8 GB memory.*

Initialisation	OpenCV	scikit-learn
Random	18.9s	165s
k-means++	29.3s	139s

As we can see OpenCV outperforms scikit-learn in all cases. OpenCV for Python is a wrapper library around the original OpenCV code built in C++, which gives it a boost in performance. OpenCV's k-means was thus selected. Initially, k-means centers were initialised randomly in development for the performance. However, during validation it was found that this method of initialisation was yielding highly different results for the intersection-over-union metric at every run, causing issues in model predictions. Hence, some speed was traded for consistency and the k-means++ center initialisation method was selected instead.

4.3.2 Thresholding

An alternative to k-means in the case of black-and-white image segmentation is thresholding. We decided to explore this option as it could have performance improvements compared to k-means.

Thresholding refers to the process of converting a greyscale image to a binary image of pixels. Pixels above a set threshold are set to 1, and the pixels below that threshold are set to 0. Thresholding depends on pixel distribution analysis. Usually, thresholding works well for images which have different peaks of pixel values in their distribution. We can then pick the value which seems

to separate out the two peaks as our threshold. However, in the case of our images we had one visible peak of pixel values and could not identify a viable threshold from the histogram. Figure 4.3 illustrates this.

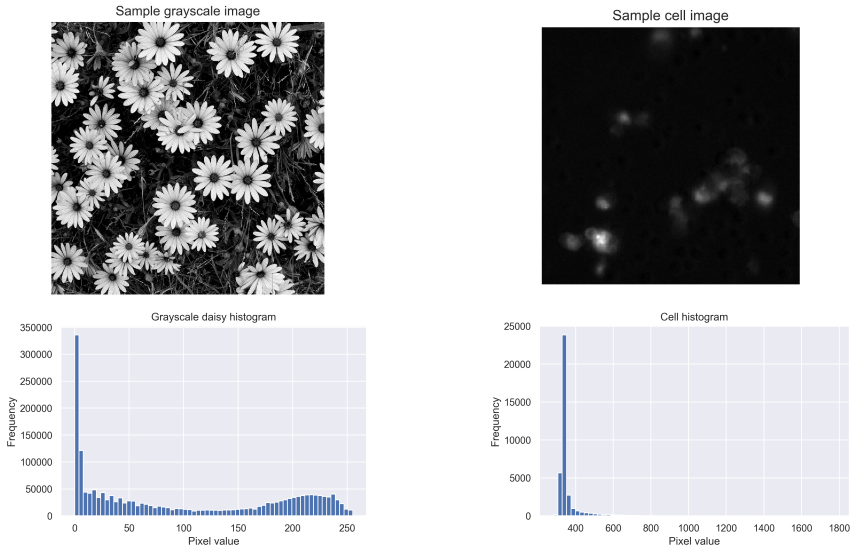


Figure 4.3: *Example greyscale images and their histogram. As we can see the greyscale image of lillies has two peaks of frequency, and we can see there is a separation between the peaks at a pixel value of around 110. The greyscale image of cells has one peak, and there is a smaller frequency of pixel values past that peak, but we cannot identify a specific pixel value as a threshold from the histogram.*

As such, we had to find an alternative for finding a suitable threshold. First, we selected the mean pixel value as the threshold. This yielded acceptable results, however some noisy pixels still came through the mask (see Figure 4.4a). To fix that problem, the threshold value was set as the sum of the mean pixel value and the standard deviation. This decision was based on the hypothesis that the noise level of an image with a flat structure can be estimated from its variation. Results were satisfactory, as shown in Figure 4.4b.

(a) *Threshold: mean pixel value* ***(b)*** *Threshold: mean + standard deviation*

Figure 4.4: *Segmented images according to different threshold values. Using mean pixel value as a threshold still lets in a lot of noise in the mask, while using the standard deviation with the mean yields a much cleaner result.*

Image segmentation through thresholding is also much faster than k-means, with 1,000 images being processed in 1.11 seconds with the use of NumPy arrays. That is a 26x speedup on k-means's OpenCV performance. Nonetheless, masks yielded by k-means are more granular. Figure 4.5 shows the difference between the masks obtained through different methods.

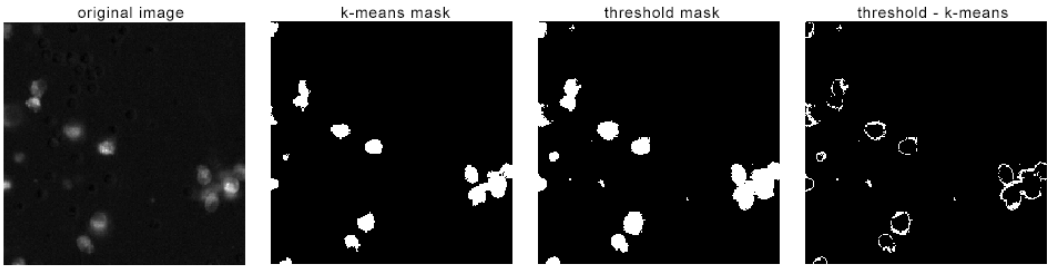


Figure 4.5: *Different versions of the same image showing, from left to right: the original image, the k-means mask of the image, the thresholded mask of the image, and the difference between the k-means and the threshold mask.*

4.3.3 U-Net

The U-Net model is a convolutional neural network developed for the purpose of segmenting biomedical data (Ronneberger et al. 2015). It is particularly useful for labelling biomedical data when images contain multiple types of cells which have not been previously separated through the use of markers such as fluorescent dyes. Moreover, U-Net models make use of data augmentation to cope with small datasets of biomedical data.

In our case, we do not need to segment the T cells from DCs as they have already been separated by the use of dyes. However, we wanted to compare the performance of simpler U-Net models trained to predict the mask of a greyscale image to the more traditional segmentation methods described in the sections above.

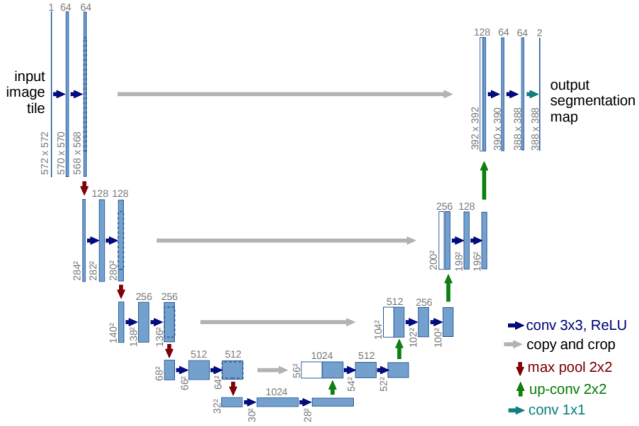


Figure 4.6: *Schematic diagram of the U-Net model. The name of the model comes from its 'U' structure, with an expansive path and a contracting path. Each blue rectangle represents a multi-channel feature map obtained through the network's operations. White rectangles are copied feature maps. The arrows show different operations. Source: Ronneberger et al. (2015)*

We trained a simple U-Net model[1] on a sample of our full dataset. The input images were the **uncombined** images of T cells and DCs, and the input labels were the masks of the input images obtained with OpenCV's k-means. Training was done on 16,000 samples and validation was done on 4,000 samples of uncombined images. Training the model took 24 minutes for 20 epochs on a Google Colaboratory notebook. We then saved the model's weights and timed the generation of masks for 1,000 of the validation samples on the same machine on which we tested

[1] https://www.depends-on-the-definition.com/unet-keras-segmenting-images/

the k-means and thresholding methods. It took 3 minutes and 3 seconds on the machine with a CPU. With Google's GPU backend, this took 1.82 seconds.

To conclude on all these segmentation techniques, thresholding was the fastest, with k-means coming second for CPU machines. We picked k-means with k-means++ initialisation for the granularity and satisfactory performance across all machines.

4.4 Autoencoder and regression models

4.4.1 Experimental setup

As this is a research project based on using deep learning models, a large part of the research involved an iterative process of repetitively tweaking the deep learning models, training them, and evaluating them. Immunology experiments produce a lot of imaging data, and we wanted our models to perform well on unseen data. Moreover, not all of our selected datasets contained instances of all classes. We wanted our trained model to be able to deal with such datasets too. As such, we selected the full dataset containing instances for all classes for training our model (see Table 3.1). Part of it was set out for validation and testing. The other two selected datasets were used purely for testing. Table 4.2 reports the number of samples in each dataset.

Table 4.2: *Train-test-validation splits for selected datasets. Models are only to be trained on the full dataset. The validation set represents 15% of the training set. The dual and DMSO-only dataset images were only used for testing.*

Dataset	Train	Validation	Test
Full	16,490	2,910	10,000
Dual	N/A	N/A	13,900
DMSO	N/A	N/A	8,000

All deep learning development was done using Keras[2] on a Tensorflow backend. Training was carried out on Google Colaboratory notebooks[3] to be able to accelerate computations with Google's GPUs.

We chose the *adam* optimizer for training (Kingma and Ba 2014). While training, we used Keras's callbacks feature to monitor the model's validation loss and change training parameters accordingly. The model would reduce its learning rate by a factor of 0.2 if no improvements in the validation loss were seen over 3 epochs (i.e. the model entered a plateau), and we used early stopping to stop training if no improvements were witnessed in validation loss over 5 epochs. Training was usually ran with a batch size of 64 over 40 epochs.

4.4.2 Convolutional autoencoder

The main model to be developed was a convolutional autoencoder. The autoencoder was built for two purposes: obtaining a smaller representation of each of the images to be fed into high-dimensional visualisation algorithms, and to be the building block for a deep regression model.

An autoencoder follows a symmetrical structure of reduction and expansion operations. The reduction operations represent the encoding part of our model, and the expansion operations represent the decoding part of our model. In order for the autoencoder input to be reduced and then reconstructed to the same number of input dimensions, we need the two sets of operations

[2] https://keras.io
[3] https://colab.research.google.com/

to follow the same pattern and use the same number of parameters. Figure 4.7 illustrates the symmetrical structure of an autoencoder, and shows the core layers of a convolutional autoencoder: convolution layers combined with activation functions, and pooling/upsampling layers.

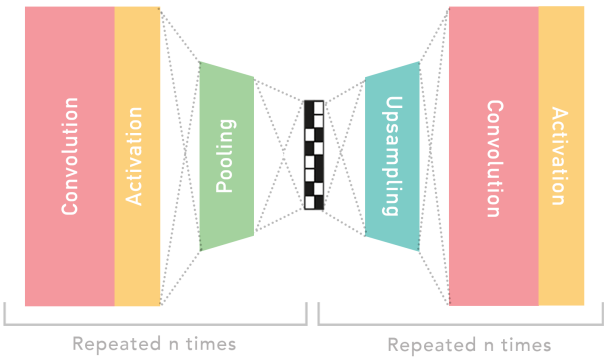


Figure 4.7: *Diagram for a typical autoencoder structure. Operations are repeated the same amount of times in the reductive path and the expansive path for the input to be reconstructed with the same dimensions at the output. Convolution layers are used with an activation function, and followed by a pooling operation in the reductive path, and preceded by an upsampling operation in the expansive path. The bottleneck layer holds a smaller representation of an image.*

Convolution refers to the process of taking the weighted sums of neighbouring values. Convolutional layers can be thought as a window sliding over an image and summing everything within that window. The specific behaviour of a convolution operation is determined by the filter it uses. Each element in the window will be multiplied by the corresponding element of the filter. We can specify both how big the filter should be, as well as how many different filters should be used. If we use 64 filters, we will learn 64 features about the image, but it will also multiply our dimensions by 64. Figure 4.8 illustrates the convolution process. The values of the filters are parameters to be learned by the model.

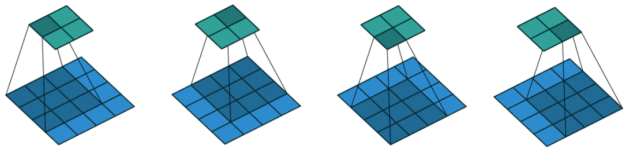
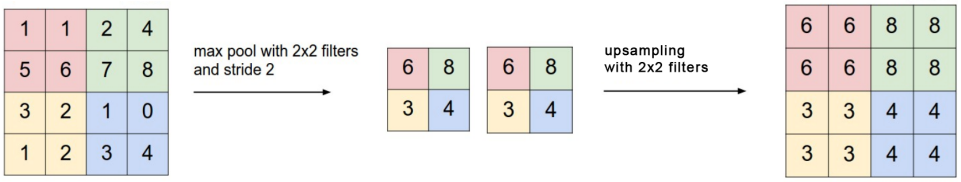


Figure 4.8: *Step-by-step illustration of a simple* 3×3 *convolution operation on a* 4×4 *image, resulting in 4 convolution features. Source: Dumoulin and Visin (2016)*

Activation functions define whether or not each element in the output of the convolutional layer (in this case) should be set to 'on' or 'off'. The purpose of an activation function is to only 'activate' elements of the hidden layers' outputs if it seems to be contributing to the learning of the model. Activation functions transform an input to an output according to their definition so they have to be chosen accordingly.

Pooling layers will downsample an image by passing a window over an image and selecting a set value from that window as the one to keep. There are two main types of pooling operations commonly used: average pooling and maximum pooling. In average pooling, the mean value of a window will be chosen as output. In maximum pooling, the maximum value of a window will be chosen. Their counterpart are *upsampling layers* which will resize an image to the right dimensions for a convolutional operation to be applied to it. An illustration of the how these layers function is provided in Figure 4.9. Using a combination of upsampling and convolutional layers

is an alternative to using deconvolution layers which have the drawback of creating checkerboard artifacts (Odena et al. 2016).



(a) Pooling *(b) Upsampling*

Figure 4.9: *Left: maximum pooling of 2×2 with stride of 2 applied to a 4×4 image. The image is reduced by a factor of 2. Right: upsampling of 2×2 with stride of 2 applied to a 2×2 image. The image is augmented by a factor of 2. Adapted from Karpathy (2020).*

Throughout the development of the autoencoder, the aim was to maximise the reduction of dimensionality while maintaining a satisfactory reconstructed image. A trade-off had to be done as quality decreases with the decrease of dimensionality. Size of the coded representation was mainly impacted by the number of hidden layers and size of the convolution and pooling filters. Quality of the reconstructed image could be improved with larger features being used in the convolutional layers and smaller filters.

Implementation choices

Knowing this, the structure of the autoencoder was tuned by trial and error and literature review of existing neural networks for similar applications.

In order to avoid losing too much detail in the reconstructed image, we only used 3×3 filters in convolution layers and pooling was done by a factor of 2. Moreover, we kept the number of features retrieved by convolution layers relatively high (64 -> 32) so as not to lose too much detail in the images. Our structure also used decreasing filter size instead of increasing filter size. Even though increasing filter size showed slight improvements in reconstruction, they were minimal, and having bigger filters towards the bottleneck layer generated bigger coded dimensions.

We also decided to replace the pooling operation in the last layer before the bottleneck by a strided convolution layer. Springenberg et al. (2014) show that adding striding to convolution layers instead of using pooling operations can achieve the same levels of accuracy. Furthermore, by adding the downsampling operation to the convolution layer, the network will learn how to best perform this operation as convolutions are weighted layers. We did not add strides to all our convolution layers in place of pooling layers as strides do come at the performance cost of having more trainable parameters. Our experiments showed that the colours in the reconstructed image seemed slightly more detailed when using striding over pooling, but this was very minimal and the choice came down to wanting to experiment.

In the hidden layers, the choice of activation function was a variant of the Rectified Linear Unit (ReLU), the Parametric Rectified Linear Unit (PReLU). ReLU simply returns 0 if the weight unit in the output of a layer is less than 0, or the actual weight if it is bigger than 0. It has become popular because it has been shown to outperform the conventional sigmoid function in hidden layers (Nair and Hinton 2010; Glorot et al. 2011), as well as helping with faster training convergence (Krizhevsky et al. 2012). PReLU differs from ReLU by having a small slope for negative values which means it does not map all negative values to 0. This slope is made a training parameter and is determined by the neural network during training. Figure 4.11 illustrates the difference between the two. He et al. (2015) showed that PReLU can improve model fitting for image classification applications. Reconstruction difference using PReLU over ReLU in our

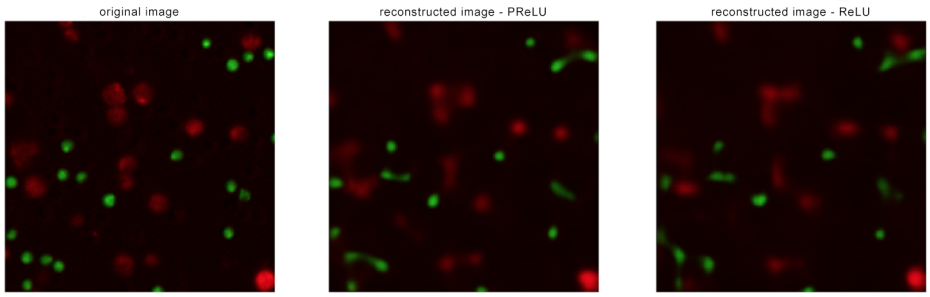


Figure 4.10: *Autoencoder reconstruction of validation images under different activation functions. The original image is on the left, and we can observe the difference between reconstruction with PReLU (middle) and ReLU (right).*

model is shown in Figure 4.10. Our PReLU model also had a slightly lower validation loss of 0.0871 compared to ReLU's 0.0875.

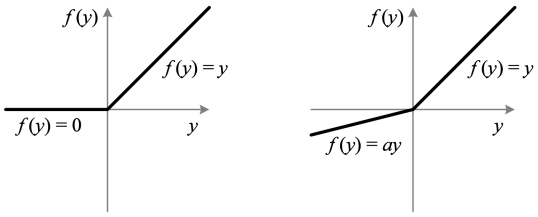


Figure 4.11: *ReLU (left) vs. PReLU (right). The negative part of the PReLU function is variable and a parameter to be learned. Source: He et al. (2015)*

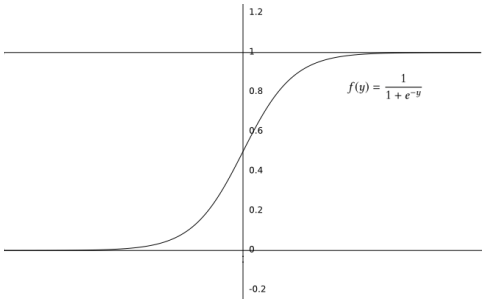


Figure 4.12: *Function graph of the logistic sigmoid function. The logistic sigmoid function is characterised by its 'S' shape. Any input will be bounded in [0, 1] at output.*

The activation function at the output of the network was the logistic sigmoid activation function, as we needed to restrict output to the [0, 1] range of our image pixels. Figure 4.12 illustrates this activation function.

Our final autoencoder structure reduced dimensions by a factor of 2 a total of 5 times, resulting in a tenfold reduction of dimensions on the original 110,592 pixels. The final coded dimensions were of 1,152. Figure 4.13 illustrates how the autoencoder reconstructed an input image depending on number of pooling operations. We traded off some reconstruction quality in order to obtain a smaller coded dimension in the hope that it would help high-dimensional visualisation algorithms perform better, as well as be a better starting block for a regression model.

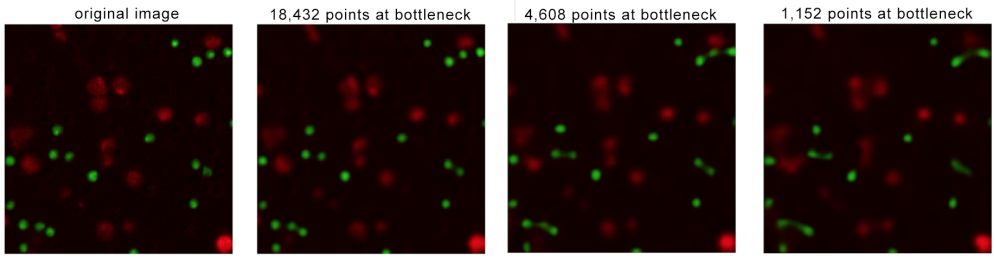


Figure 4.13: *Images reconstructed by the autoencoder according to the size of the coded layer. From left to right: the original image to reconstruct, the image reconstructed from 3 convolution-pooling operations, the image reconstructed from 4 convolution-pooling operations, the image reconstructed from 5 convolution-pooling operations. Quality decreases with the number of pooling operations.*

The autoencoder was trained with binary cross-entropy as a loss function, as we were aiming to minimise the difference between two distributions: the input, and the reconstructed output. The final structure of the autoencoder is shown below in Figure 4.14. Keras code for building the model is available in Listing A.1.

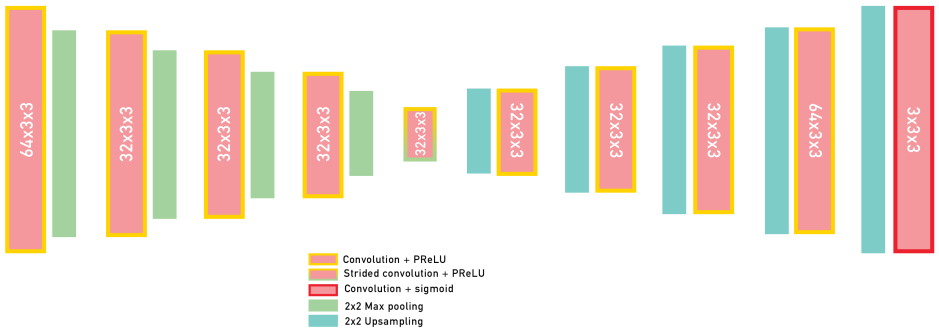


Figure 4.14: *Diagram of the final implemented autoencoder structure. Convolution blocks are represented in pink, attached to activation functions in yellow or red. Downsampling operations are in green and upsampling operations are in blue. The special strided convolution layer uses a green to yellow gradient to represent the downsampling of the striding and the activation function set in the layer.*

4.4.3 Deep regression

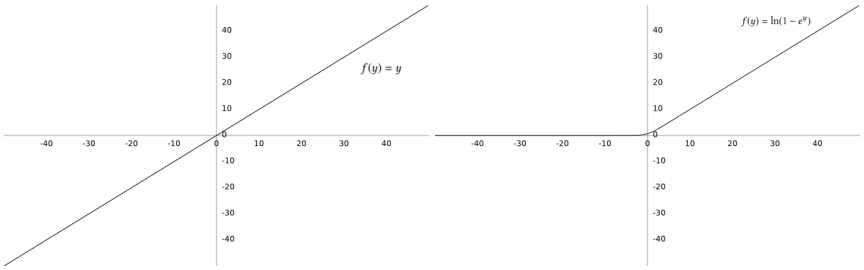
The regression model was built on the encoder part of the autoencoder. We wanted to use the dimensionality reduction capacities of the encoder layers to evaluate whether interaction could be quantified from an image's features.

The structure of the regression model was kept simple. Research reviewing existing general-purpose neural networks such as VGG-16 showed that adequate tuning of these models resulted in a performance close to "state-of-the-art" without having to develop more complicated structures (Lathuilière et al. 2018). We did not reuse such a general purpose model but the convolutional part of the VGG-16 network is similar to the encoder part of our autoencoder, using 5 combinations of convolutions and pooling operations. These operations are followed by a small number of fully-connected layers (Simonyan and Zisserman 2015). Hence, we decided to follow this template and not make our regression structure overly complex.

The encoder model was extracted from the autoencoder with the bottleneck layer flattened. This encoder model was then extended with two fully connected layers separated by a dropout layer.

The fully connected layers were activated by ReLU. Dropout layers are used to randomly turn some units of the output of hidden layers to 0. Dropout was shown to make models more robust and prevent overfitting (Hinton et al. 2012). The final regression predictions were outputted with a fully connected layer of size 1, activated by a linear function.

The choice of the linear function was justified by the following. The input to the regression model was $192 \times 192 \times 3$ images accompanied with labels in the range $[0, 100]$ representing a percentage of overlap between two groups of cells. As such, we want the output activation function to be limited to $[0, 100]$. Both softplus and linear activations were candidates. The linear activation had to be used with a non-negative kernel constraint, as overlap percentage cannot be negative but the linear function normally allows for negative values. The softplus activation function outputs values in the range $[0, +\infty]$. Performance in terms of training loss was similar, however the linear function performed overall better. Furthermore, softplus does not always map an input to itself around 0 (shown in Figure 4.15) and is harder to differentiate compared to the linear function, which would make training slower.



(a) *Linear* ***(b)*** *softplus*

Figure 4.15: *Function graphs of the linear function (left) and the softplus function (right).*

The regression model was trained with mean-squared-error loss, which calculates the difference between the predictions of the model and the truth values of the labels. The final structure of the deep regression model is shown in Figure 4.16. Keras code for building the model is available in Listing A.2.

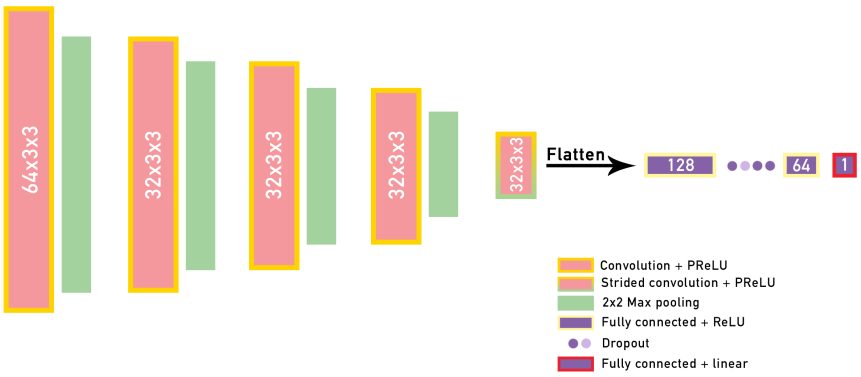


Figure 4.16: *Diagram of the implemented regression structure. Note that the first part of the diagram is the exact same as the contracting path of the autoencoder shown in 4.14, but the output of the strided convolutional layer is flattened and passed onto fully connected layers. Fully connected layers are purple rectangles, and the Dropout layer is shown by purple dots. Activation functions are represented in yellow or red.*

5 Evaluation

5.1 Methodology

This section will evaluate the performance of our autoencoder and regression models on unseen images. As explained in Section 3.1.3, three sets of images were selected for research:

- A **full dataset** containing all categories, with no class imbalance issues
- A **dual dataset**, with no class imbalance issues, containing images from two classes only
- A '**DMSO-only' dataset**, which should show the most distinction between classes, with class imbalance issues. This dataset did not include any experimental conditions using drugs, only stimulation conditions.

Each image in these datasets was categorised by stimulation level: one of *Unstimulated*, *OVA*, or *ConA*. If the image contained image aberrations, it was labelled *Faulty* instead in the case of the autoencoder. In the case of the regression model, the overlap value of *Faulty* images was set to 0. Figure 5.1 illustrates what an image from each category looks like.

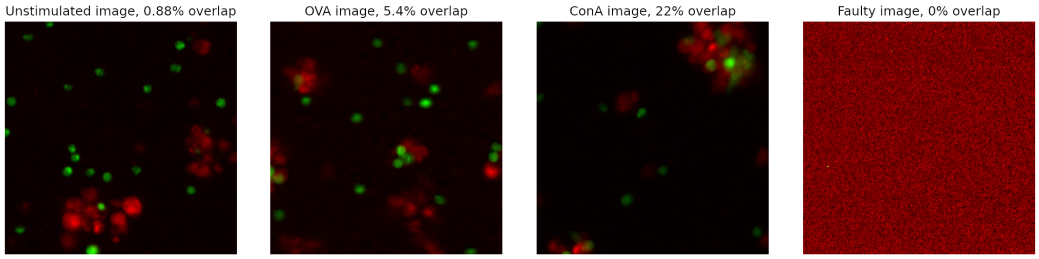


Figure 5.1: *Example validation images taken from each stimulation category. Each image is also labelled with its overlap value. From left to right: Unstimulated,* 0.88% *overlap; OVA,* 5.4%*; ConA,* 22%*; Faulty,* 0%*.*

We had two versions of each dataset: one which had been noise-corrected, and one which had been background-corrected through the help of k-means colour segmentation. This dataset was denominated the 'masked' dataset. We trained separate models on those two datasets to evaluate the impact of the background on the model's learning.

Training and validating was done on the full dataset's 19,000 instances. We had 10,000 instances of this dataset left for testing, and 13,800 instances of the dual dataset, as well as 8,000 instances of the DMSO dataset. All datasets were consistently shuffled before training and testing with the same random state parameter. We made sure there was no overlap between any of the datasets. No image was used for both training and testing.

The following criteria were chosen in our evaluation:

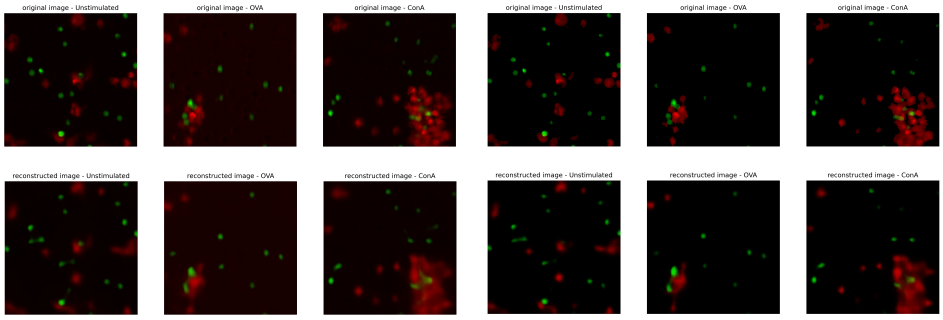
- The success of the autoencoder model will be evaluated on its capacity at image reconstruction, as well as if a UMAP projection of the coded images shows some underlying structure of the data.
- The regression model will be evaluated on how well it predicts overlap values for unseen images of immune cells.

5.2 Autoencoder

5.2.1 How well can we reconstruct an image?

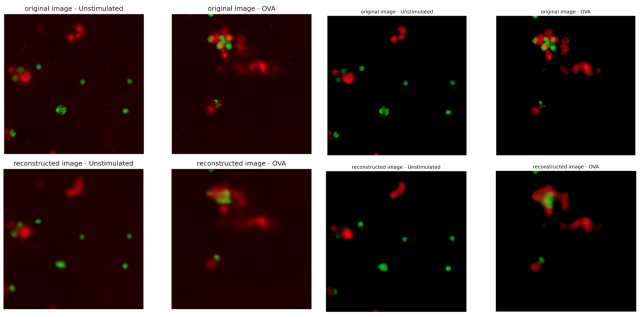
Our autoencoder model is trained on binary cross-entropy loss. The model learns by trying to minimise the difference between two distributions of pixels: the input, and the output. The closer the loss is to 0, the more similar the two distributions are. This means that a score closer to 0 is equivalent to the autoencoder reconstructing an image well. We can use this value as a metric for comparing results between models and datasets. We can also assess the visual quality of an image on reconstruction.

The following images in Figures 5.2, 5.3, 5.4 show how the autoencoder performed on image reconstruction for each of the datasets. The output images were all reconstructed by the model from a 1,152 inner representation of the input image of 110,592 pixels.



(a) *Normal images* ***(b)*** *Masked images*

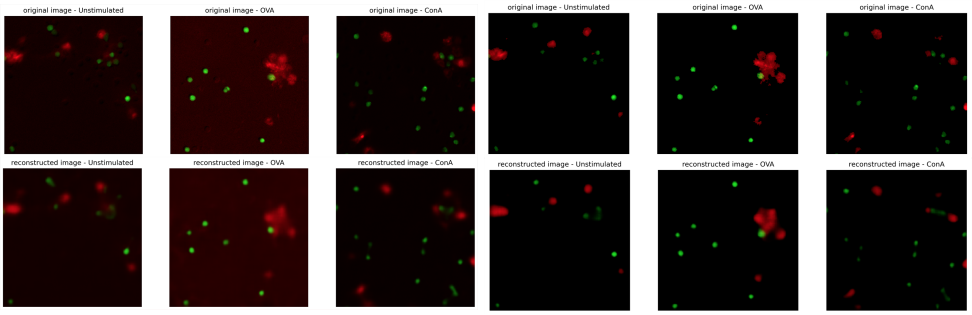
Figure 5.2: *Autoencoder-reconstructed images from the test images of* ***full*** *dataset. For each of the subfigures, categories are Unstimulated, OVA, ConA from left to right. Input images are at the top, and reconstructed images at the bottom.*



(a) *Normal images* ***(b)*** *Masked images*

Figure 5.3: *Autoencoder-reconstructed images from the* ***dual*** *dataset. For each of the subfigures, stimulation categories are Unstimulated (left) and OVA (right). Input images are at the top, and reconstructed images at the bottom.*

The main feature we can note from these images is that visually, the autoencoder reconstructed the images in quite a satisfactory way – we can still see where cells are and where they overlap. We can recognise the images's similarities. We can also identify some flaws. The main drawback of the autoencoder model is the way it merges different cells together, creating 'blobs' rather than more detailed cell objects.



(a) *Normal images* ***(b)*** *Masked images*

Figure 5.4: *Autoencoder-reconstructed images from the* ***DMSO*** *dataset. For each of the subfigures, categories are Unstimulated, OVA, ConA from left to right. Input images are at the top, and predicted images at the bottom.*

On the other hand, we can note that the masked autoencoder trained on the masked images of immune cells achieve a higher level of detail, for all datasets. Particularly, the images reconstructed by the masked model retain the circularity of the cell objects better, even in the unseen datasets. This can be observed in Figures 5.3b and 5.4b where the dendritic cells in red hold their original shape and detail better in comparison to their non-masked counterpart.

Visually, it is not immediately clear how much better the autoencoder performs on the full dataset, part of which it has seen, compared to the unseen dual and DMSO datasets. Indeed, we will not be able to fully compare the reconstruction of these images by eye. Further to visual evaluation of the reconstruction of the immune cells, the two models's prediction losses can provide another perspective on their performance. Table 5.1 reports loss values for each model on each dataset.

Table 5.1: *Binary cross-entropy loss value for the autoencoder models's performances on each of the test datasets. A value closer to 0 is better. We can observe a gain in performance across datasets in the masked model.*

Model	Full	Dual	DMSO
Normal	0.0870	0.1159	0.0936
Masked	0.0156	0.0174	0.0166

From this data we can say that removing noisy backgrounds entirely from the images of immune cells achieved substantial improvements. It also made the autoencoder's performance on unseen datasets more comparable to the full dataset.

5.2.2 Can we find an underlying structure in the images of immune cells?

While we have shown that an autoencoder can complete the task of reducing the dimensionality of an image of immune cells and reconstruct it fully from this reduced image, we want to look at whether or not this would help with a two-dimensional projection of the datasets. The aim for two-dimensional visualisation of the datasets was that it would allow us to uncover clusters of images grouped around the same stimulation conditions.

Each element projected onto a 2D plane is labelled with one of the *Unstimulated*, *OVA*, *ConA*, or *Faulty* categories at the plotting step.

Baseline performance

The expectations were that an autoencoder would allow us to extract the necessary features from an image that would be sufficient for it to be reconstructed, but also for it to be analysed. As such, we wanted to compare the visualisation performance of UMAP on the coded images to a baseline performance on raw images that have not been reduced in dimensionality. This was only ran for the full dataset as the large dimensions meant this computation took a considerable amount of time. On a 2015 MacBook Pro with 2.7 GHz i5 core and 8 GB of RAM, running UMAP on the $10,000 \times 192 \times 192 \times 3$ test instances of the full dataset worth took 1 hour, 5 minutes and 33 seconds. The result visualisation is shown in Figure 5.5.

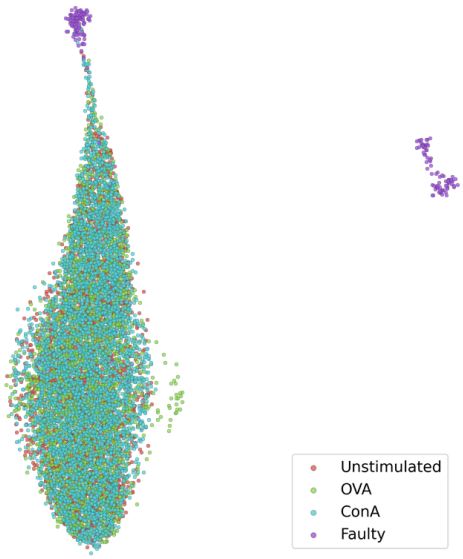


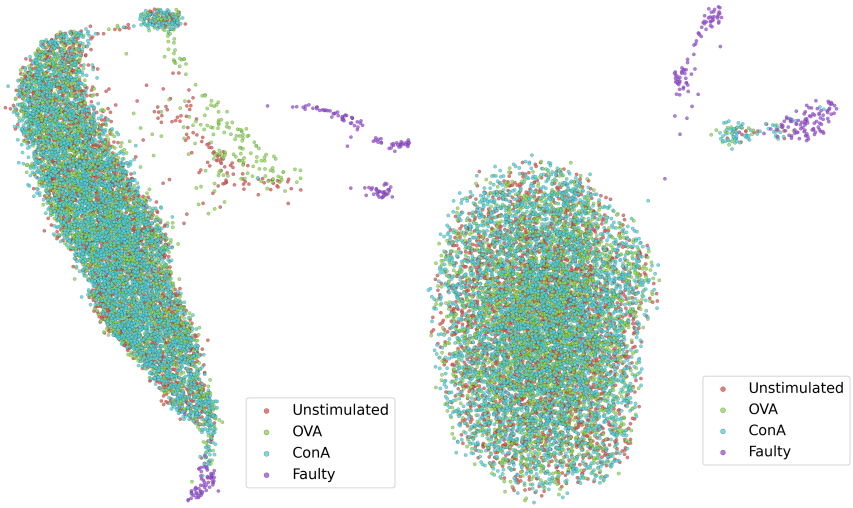
Figure 5.5: *Two-dimensional UMAP projection of the 10,000 testing images in the full dataset. UMAP was ran with base parameters* $n\_neighbors = 15$, $min\_dist = 0.1$. *The legend highlights which colour corresponds to which stimulation category.*

This projection highlights that with the raw $192 \times 193 \times 3$ images of immune cells, only *Faulty* images seem to be recognised and there is no clear distinction between images of other categories. A few OVA/green points are distanced from the main cluster. We will now look at whether reducing the dimensionality of the images with an autoencoder helps make this distinction.

Trained performance

UMAP's speed allowed us to try different parameters for our visualisations to attempt to yield the best projections of the data, as running UMAP only took between 30 seconds and 1 minute and 15 seconds to run on the encoder-processed images. That is an average 75x speedup on the baseline performance.

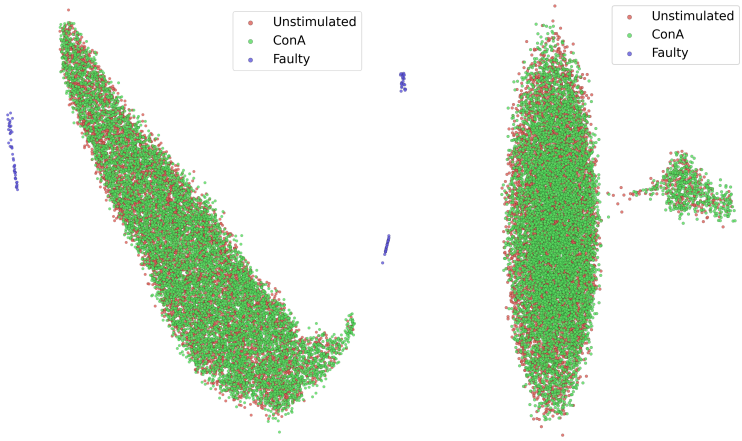
We can see the projection of the **full dataset**'s test images in Figure 5.2. Two smaller clusters have emerged which were not present in the baseline projection. The labelling shows us that the sparser cluster contains two lines of red/Unstimulated and green/ConA points, which could indicate that the autoencoder did help extract the most important features out of the images. However, the larger cluster still bears no distinction between images of different categories. Furthermore, the projection obtained with the masked model did not seem to perform better, with less clusters being uncovered this way.



(a) Normal images *(b) Masked images*

Figure 5.6: *Two-dimensional UMAP projections of the 10,000 test images from the **full** dataset. These projection were obtained with UMAP parameters n_neighbors=30 and min_dist=0.8. The legend highlights which colour corresponds to which stimulation category.*

We hoped for the **dual dataset** to perform better as it only had two stimulation categories to distinguish between. The normal and masked visualisations are shown in Figure 5.7. Although no clear distinction of clusters is shown the normal projection, a small cluster formed in the projection of the masked dataset. However, this cluster contains images of both *Unstimulated* and *OVA* categories.



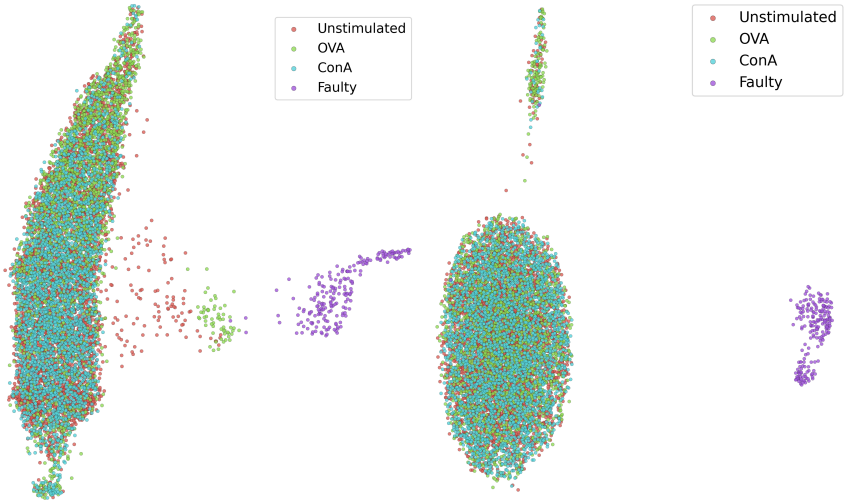
(a) Normal images *(b) Masked images*

Figure 5.7: *Two-dimensional UMAP projection of the 13,800 test images from the **dual** dataset. These projections were obtained with UMAP parameters n_neighbors=30 and min_dist=0.8. The legend highlights which colour corresponds to which stimulation category. The normal images are not clearly distinguished by category, but the masked images yield a separate cluster. None of these clusters bear distinction between images of different stimulation.*

The images from the DMSO images should show the most distinction between each other as the immune cells from the images have only been influenced by the stimulation antigens and

no other compounds, while immune cells from the experiments represented in the other two datasets have also been injected with drug compounds.

The **DMSO** projection is shown in Figure 5.8, from which we can observe some interesting features. Firstly, there is one main cluster, but interestingly the coloured points show a green to blue gradient, which we could attribute to a distinction between the *OVA* and *ConA* categories. Blue/*ConA* points become sparser at the top of the cluster, and the same happens for green/*OVA* points at the bottom of the cluster. This is overlaid on top of images of the *Unstimulated* category, however this might show that some distinction is being made between the images. Furthermore, we can see two smaller collections of red/*Unstimulated* points and green/*OVA* points. The projection of masked images does not show this *OVA*-*ConA* distinction, but does have a separate cluster of *Unstimulated*/*OVA* points.



(a) *Normal images* ***(b)*** *Masked images*

Figure 5.8: *Two-dimensional UMAP projections of the 8,000 test images from the* ***DMSO*** *dataset, which have been noise-corrected (left) and background-corrected (right). UMAP was ran with parameters* `n_neighbors` = 30 *and* `min_dist` = 0.8. *The legend highlights which colour corresponds to which stimulation category. In the normal images, the main cluster of points does not show a clear delimitation between images of the same category, however there does seem to be some distinction as we can see a gradient.*

Every projection highlights that UMAP does not struggle to distinguish *Faulty* images from images containing immune cells. This is useful as it acts as a control that tells us that UMAP is noticing differences in images, and that the autoencoder does not corrupt the images and remove important features. Nonetheless, UMAP did not yield clusters showing any kind of distinction between categories in the cases of the **full** and **dual** datasets. A difference is more noticeable in the **DMSO** dataset. This difference is no longer visible in the masked projection.

5.2.3 Are images within the same cluster structurally the same?

Some projections shown above still showed some clusters and outlier points. We wanted to assess whether the images in these points were indeed structurally similar. In order to explore this, we developed a tool that makes use of matplotlib's animation API[1]. We can hover over the points of the visualisation to display the original image that is represented by one of the projection points. The drawback of the tool is that it struggles over large clusters of points as it requires quite a lot

[1] https://matplotlib.org/api/animation_api.html

of memory to update the coordinates in the graph, resize a $192 \times 192 \times 3$ image, and display it, and is slow in such cases.

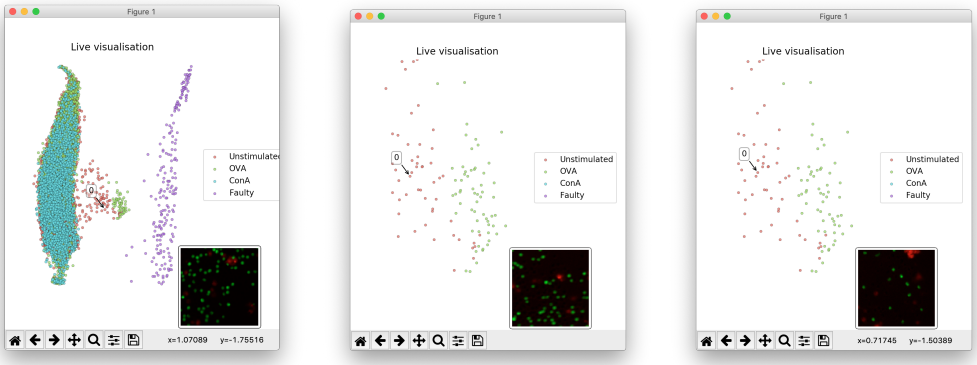


Figure 5.9: *Three images extracted from a live visualisation graph of the UMAP projection shown in Figure 5.8. The graph on the left is unzoomed. The other two graphs have been zoomed in, showing the tool's functionality as well as a closer view of the cluster. We give two example images in the same cluster of similar structure, but one that does not follow the trend.*

Figure 5.9 highlight the tool's functionality. We picked the projection of the DMSO dataset from Figure 5.8 above that showed distinct clusters of images within the same category to evaluate whether or not they were meaningful. The projection in the first image will look slightly different as the plot is of smaller size and the method for drawing the scatter plots had to be adapted for the task. We then zoom over clusters of interest to be able to pick out points more efficiently. This view shows us that we can find two images within the same cluster that look structurally the same, but another image which does not is also in that same structure.

Figure 5.10 shows a different case. We highlight three points in a cluster which are close to each other and show they are similar – they all are a particular hue of green. This highlights that our noise correction was not necessarily successful in all cases, and explains why less clusters are found with the masked images.

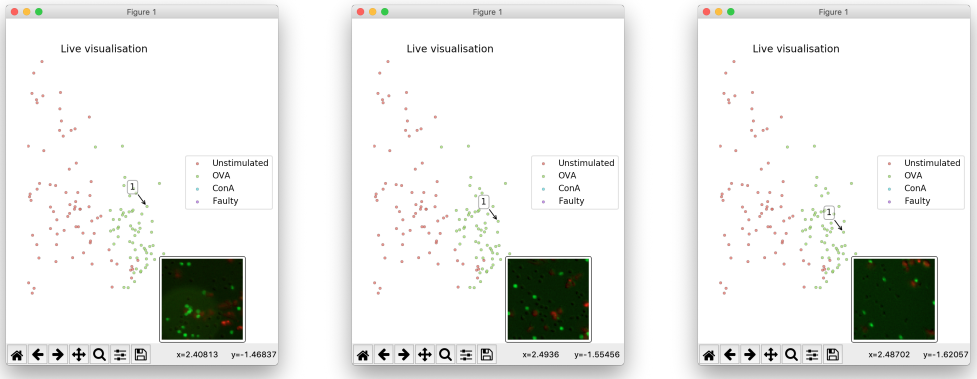
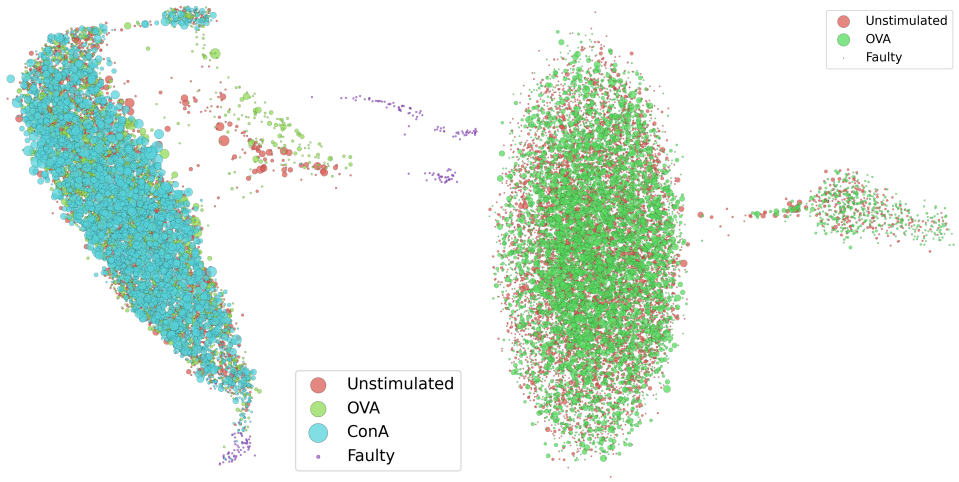


Figure 5.10: *Three images extracted from a live visualisation graph of the UMAP projection shown in Figure 5.8. The graph has been zoomed in to the distinctive green cluster. This highlights three images that are similar – they are all a shade of green.*

5.2.4 Are our visualisations more meaningful with interaction measure meta-data?

In the case of visualisations that did yield some groupings, but contained points of mixed categories, we hypothesised that other meta-data about the images might explain why some clusters were formed. Indeed, maybe the stimulation level of immune cells was not creating enough distinction between images, and they were more influenced by something else. For example, how much the cells overlapped in the image, regardless of their stimulation category.

We plotted the visualisations again, but using the interaction measures collected with the intersection-over-union metric to change the size of the markers. We decided to tweak the visualisations shown in Figure 5.6a and Figure 5.7b as they yielded some separated groupings, but of mixed labels. The results are shown in Figure 5.11 below.



(a) UMAP projection of the full dataset *(b) UMAP projection of the masked dual dataset*

Figure 5.11: *UMAP projections for the full dataset and masked dual dataset, tweaked with the size of overlap in each image as the size of the scatter points. The legend shows the category of each point as well as the average size of the points.*

These visualisations show us that although all images in the separated clusters contain a similar level of interaction, some other images in the main clusters also have that level of interaction. As such, we also cannot say that UMAP is finding structure in the images based on how much overlap (i.e. shades of orange) there is.

5.3 Regression

Our regression task is to predict the percentage of overlap between T-cells and dendritic cells from an image where T-cells are shaded green, and dendritic cells are shaded red.

5.3.1 Metrics

Our regression model is trained on mean squared error (MSE). We decided to evaluate on root-mean-square-error (RMSE) as it is a common metric for evaluating the difference between predictions of a model and actual truth values: RMSE has the advantage of being in the same unit as the dependent variable. In our case the unit is the percentage of overlap between the cells objects obtained with intersection-over-union. RMSE is defined as the square root of

the quadratic mean of the difference between our predicted values and their truth values. The equation is as follows:

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2} \tag{5.1}$$

where n is the number of samples in the dataset, y is the true value of a sample, and $\hat{y}$ is the predicted value of a sample.

We also include the (unbiased) standard deviation (SD) of our predicted results to express their variability. The formula for SD is as follows:

$$\sigma = \sqrt{\frac{1}{n-1}\Sigma_{i=1}^{n}(\hat{y}_i - \mu)^2} \tag{5.2}$$

where n is the number of predicted samples, $\hat{y}_i$ is the predicted value of a sample and μ is the mean of all predicted values.

5.3.2 Can we quantify interaction from an image of immune cells?

We evaluate the predictions made by the normal and masked regression models by looking at the predictions of each dataset in each stimulation category, however the model was not fed any stimulation category labels during training. Nonetheless, we might see more overlap across different stimulation categories if they generate more interaction between immune cells, so it is important to also look at errors in prediction within those ranges of values.

Full dataset

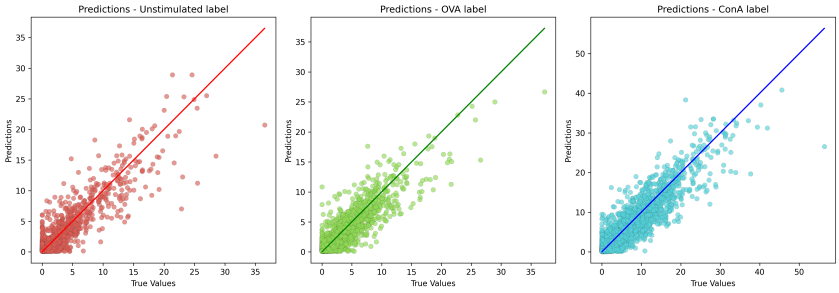
Our regression model reports an overall score of 1.838 ± 4.583 on the test instances of the full dataset. The masked regression model trained on the images which have a black background reports a score of 1.161 ± 4.732, which shows an improvement in RMSE. Variance in the true values is of 4.853 overall, meaning that we can see a very similar amount of variance in the predicted values. Table 5.2 reports the full scores on all stimulation categories.

Table 5.2: *Average RMSE and SD scores (in %) for the regression model's predictions on the* ***full*** *dataset. The closer the value is to 0, the closer the prediction is. We can see that the difference remains quite low across all datasets, with ConA having the highest RMSE and SD.*

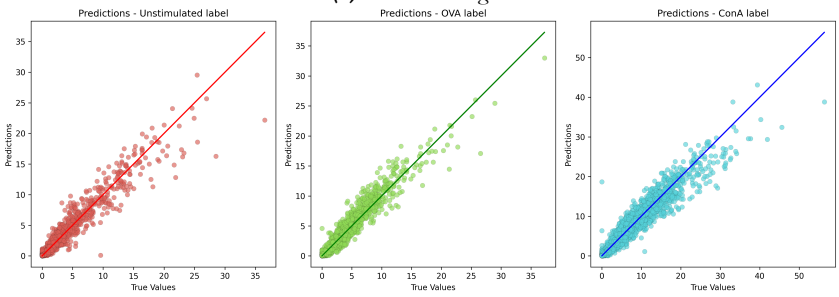
Images	Unstimulated	OVA	ConA	All
Normal	1.344 ± 3.102	1.387 ± 2.963	2.502 ± 6.004	1.838 ± 4.583
Masked	0.832 ± 3.174	0.766 ± 3.355	1.642 ± 6.067	1.161 ± 4.732

We can see that the *ConA* category has a higher RMSE score, while *Unstimulated* and *OVA* scores are similar. Given the large standard deviations on each of the scores, the differences between them for each different stimulation category cannot be considered statistically significant. The high standard deviation for images with *ConA* stimulation could be partly explained with the range of values for that category.

Figure 5.12 is a scatter plot showing the line of best fit for our predictions. In the ideal scenario, where the regression model always makes the right prediction, each point should be placed on the diagonal line. We can see that that in the *Unstimulated* category, points struggle to follow this line. There is a large number of errors when the value to predict is closer to 0. This is also the case for the *OVA* and *ConA* categories, although less so. Furthermore, the plot shows that images in the *ConA* category have a higher percentage of overlap in general. This could explain why the



(a) Normal images



(b) Masked images

Figure 5.12: *Scatter plot comparing true values (x-axis) to predicted values (y-axis) for the* ***full*** *dataset. The line of best fit is the straight continuous line running diagonally true the plot, which is the line we want our predictions on. The predictions are plotted for, from left to right: Unstimulated (red), OVA (green), and ConA (blue). Each sub-plot is not on the same scale as each category does not have the same range of overlap values, but we wanted to show more granularity for each sub-plot.*

ConA category yields a higher RMSE and SD score. In fact, if more points in the *Unstimulated* and *OVA* categories have an overlap value of 0, then it would explain why they have a smaller RMSE and SD score.

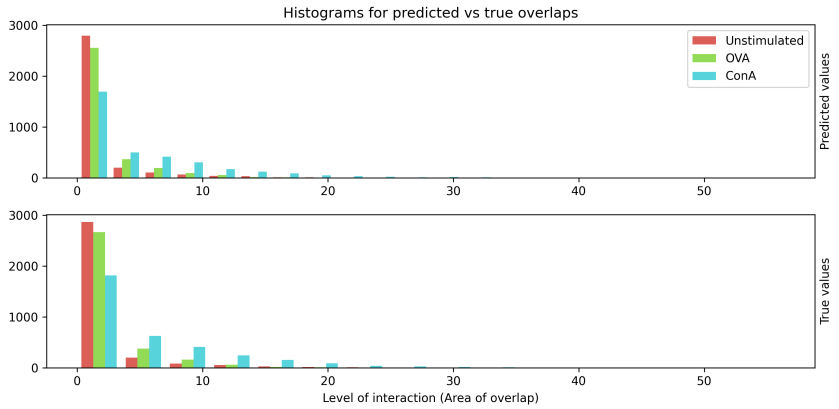


Figure 5.13: *Histogram showing the distribution of overlap values for each stimulation category. The true values are at the bottom, and the predicted values for the normal dataset are at the top. Most images have an overlap value of around 0, and images in the ConA category show the biggest overlaps.*

To illustrate this, we show the distribution of the true and predicted values of overlap between immune cells in the images in a histogram in Figure 5.13. The true distribution of overlap values

highlights that there are a lot more of overlap values distributed around 0 for the *Unstimulated* and *OVA* categories.

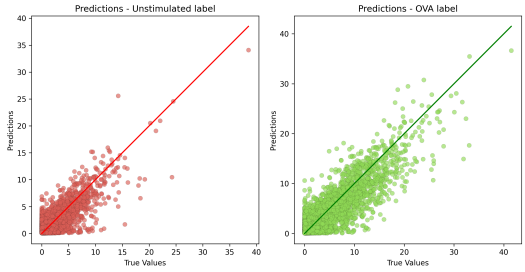
Nonetheless, the RMSE scores for our masked regression model are much more promising. This is also reflected in the scatter plots. Figure 5.12b shows that in comparison to the normal model, predicted values for the masked model follow the line of best fit much closer.

Dual dataset

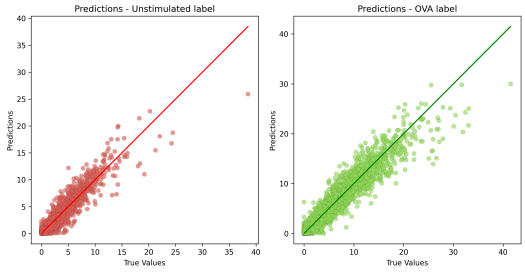
Regression performance on the unseen dual dataset was similar as to above. Overall, it achieved lower scores of RMSE with general scores of 1.677 ± 3.183 for the normal model and 0.932 ± 3.456 for the masked model. The SD of the true values is of 3.626 overall, hence we can say the predictions represent the variability of the true values. The RMSE scores need to be put into contrast with the fact that in the full dataset, the *ConA* category had higher RMSE scores due to having a distribution with higher overlap values. Table 5.3 reports the full RMSE and SD scores. Compared to Table 5.2 above, the regression model achieved a similar performance on the unseen dataset for images in the *Unstimulated* category.

Table 5.3: *Average RMSE and SD scores (in %) for the regression model's predictions on the* ***dual*** *dataset. The closer the value is to 0, the closer the prediction is. There is an improvement in RMSE score in the masked model, and it accounts for more variance in the data.*

Images	Unstimulated	OVA	All
Normal	1.233 ± 1.981	2.026 ± 3.898	1.677 ± 3.183
Masked	0.673 ± 2.265	1.133 ± 4.310	0.932 ± 3.456



(a) *Normal images*



(b) *Masked images*

Figure 5.14: *Scatter plot comparing true values (x-axis) to predicted values (y-axis) for the* ***dual*** *dataset. The line of best fit is the straight continuous line running diagonally true the plot, which is the line we want our predictions on. The normal regression models performs rather poorly around overlap values of 0, in both categories.*

Figure 5.14 shows how the predictions are positioned around the line of best fit for this dataset. We can see that predictions are poor around 0: the model predicts values too high for true values

of 0, and predicts values too close to 0 for values much higher. For example, a true value of close to 15% overlap was predicted as close to 0% overlap as shown in the *Unstimulated* graph. Predictions also become poorer towards higher values, for both normal and masked images.

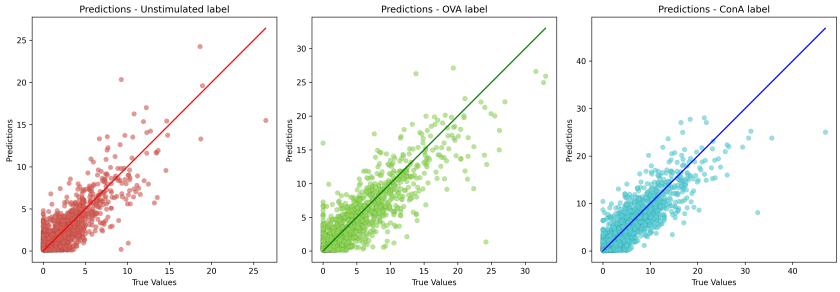
DMSO dataset

For the regression task, we are not comparing images of different categories, but trying to predict an interaction value from an image. As such, we were hoping that the **DMSO** dataset would have similar performance as the **full** dataset, as class imbalance issues should be less significant.

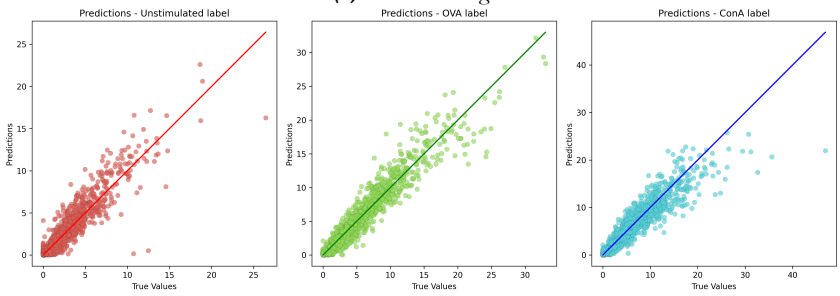
The regression model achieved an overall score of 1.837 ± 3.816 for the normal model, and 1.161 ± 4.732 for the masked model. SD of the true values is of 4.405 overall. Again, we see an improvement in scores in the masked model. Full scores are reported in Table 5.4. As expected, performance is similar across all categories compared to the full dataset, with the *OVA* category showing the most difference.

Table 5.4: *Average RMSE and SD scores (in %) for the regression model's predictions on the* ***DMSO*** *dataset. The closer the value is to 0, the closer the prediction is. We again observe an improvement in the RMSE score in the masked model, and it accounts for variance in the data closer to the true SD value of 4.405.*

Images	Unstimulated	OVA	ConA	All
Normal	1.013 ± 1.945	2.202 ± 4.308	2.567 ± 4.720	1.837 ± 3.816
Masked	0.700 ± 2.147	1.199 ± 4.809	1.773 ± 4.665	1.179 ± 4.011



(a) *Normal images*



(b) *Masked images*

Figure 5.15: *Scatter plot comparing true values (x-axis) to predicted values (y-axis) for the* ***DMSO*** *dataset. The line of best fit is the straight continuous line running diagonally true the plot, which is the line we want our predictions on. For both the normal and masked models, there seems to be some large errors around higher values of overlap. Each sub-plot is not on the same scale as each category does not have the same range of overlap values, but we wanted to show more granularity for each sub-plot.*

The scatter plot showing the line of best fit is available in Figure 5.15. Similarly to the dual

dataset, we can see that there is a higher variation around predictions of higher values in both types of images. This might signify that our regression model is struggling with higher values of overlap.

5.4 Discussion

In this section we have shown the results of the development of our autoencoder and regression models. These results have shown that convolutional autoencoders can reconstruct images of immune cells well with a highest binary cross-entropy loss of 0.1159 and a lowest score of 0.0156. The autoencoder also sped up the process of high-dimensional data projection from an hour to minutes, which is particularly significant for the field of immunology where imaging data is predominantly high dimensional. Moreover, it was an efficient building block to extract features from images to be passed onto fully connected layers for a regression task. Indeed, our deep regression model has shown good results with the highest average RMSE score being 1.838 (%). Particularly, we have noticed that the correct pre-processing of images, particularly the removal of noisy backgrounds, greatly helps a model's learning.

The results also have implications for the analysis of interaction between immune cells. From what we observed here, we can say that deep learning methods are suited for the task of processing images of immune cells. We were able to train a consistent regression model for the task of predicting overlap value from an image of immune cells, where we assumed more overlap meant more interaction between immune cells. This method can be faster on GPUs for obtaining metrics from images rather than segmenting images and calculating overlaps directly.

We were not able to uncover an underlying structure to the data with the help of the autoencoder. However, there could be multiple reasons for this as our evaluation had limitations. The difference between DMSO-only results and the results on the other datasets tells us that the experimental conditions behind each image did have an impact on their structure. This is significant as it highlights that microscope images will have to be carefully selected and processed in order to yield results suited for analysis. Datasets should be built around precise experimental conditions to best exploit the capabilities of deep learning models. For example, a dataset could be built from images of immune cells taken from the same stimulation category, but labelled by drug compound used. It is also worth noting that the results here were obtained with image patches of size 192×192. This gives quite a local view of the cells, and a bigger window size could give a more global view and as such more information about how different cells behave together. Therefore the size of the sub-images should also be taken in consideration, as it might yield different results.

6 Conclusion

The purpose of our research was to apply deep learning methods to microscope images of immune cells and evaluate whether a deep learning approach could be successfully applied to the analysis of interaction between different types of immune cells.

To assess this, we implemented a convolutional autoencoder from which we implemented a regression model. We wanted to use the autoencoder's power at dimensionality reduction to visualise the structure of the imaging data, as well as use it to build a powerful regression model capable of predicting a value of interaction from an image of immune cells. The specific questions we were looking to answer were whether or not there was an underlying structure to the images of immune cells under different experimental conditions, and whether or not we could quantify interaction from an image of immune cells.

Our evaluation of these two models showed that there is potential in using deep learning methods for imaging data of immune cells, following in the footpath of similar research done in cancer research. While we could not successfully find an inherent structure to the data using a UMAP projection, this could have been influenced by the choice of datasets and size for creating patches from the raw images. Moreover, we showed that a regression model could predict a value of overlap from an image with a best RMSE score of 1.161 ± 4.732 on a background-corrected dataset containing images from all our categories.

With appropriate pre-processing and further research, deep learning techniques could be a new approach to the analysis of interaction between immune cells, allowing researchers to analyse their datasets further.

6.1 Future work

There is a number of different routes that could still be explored. Firstly, we only explored one size of sub-images in our pre-processing. A bigger size of sliding window could include more details of the images and allow for a better global overview of the impact of experimental conditions on immune cells. As such, bigger sub-images might reveal a structure that an algorithm such as UMAP could analyse fast with the help of an autoencoder.

Furthermore, we only evaluated one metric for our regression task, which was the percentage of overlap represented by the intersection-over-union metric. This represents some limitations in simplicity of analysis. Indeed, a large clustering of T cells around a dendritic cell *without overlap* could also signify a level of interaction, without overlap being observed. Moreover, we are using a two-dimensional view of the cells. Two different types of cells overlapping might not mean that they are interacting. A T cell could simply be sitting on top of a dendritic cell, without communication happening between the two.

Finally, we have briefly touched upon the U-Net model. Our dataset gave us access to pre-segmented pictures of immune cells. However, we could use this wealth of pre-segmented data to train a U-Net model to segment T cells and dendritic cells object in greyscale images. Alternatively, such a model could be used to extract features from segmented immune cells, such as size of cells and granularity.

A Appendices

A.1 Autoencoder model initialisation

A.2 Regression model initialisation

```
def make_regression(encoder):
    """
    Initialise a regression model for training using
    a previously created encoder model
    """

    model = Sequential()
    model.add(encoder)
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.15))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='linear',
        kernel_constraint=constraints.NonNeg()))

    model.compile(loss='mean_squared_error', optimizer='adam')

    return model
```

Listing A.2: *Keras code for initialising the regression model developed here. It is constructed from an encoder model which has been previously initialised. The encoder is then extended with fully connected (Dense) layers of decreasing sizes, separated by a Dropout layer for robustness. The final layer is activated by a linear function constrained not to be negative, and outputs a numeric value.*

```
def make_autoencoder():
    """
    Initialise autoencoder model for training and return reference to both decoder
        and encoder parts of the model.
    """

    # image shape is defined in the configuration
    input_img = Input(shape=(imw, imh, c))

    # layers for reduction of image
    x = Conv2D(64, (3, 3), padding='same')(input_img)
    x = PReLU()(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = PReLU()(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = PReLU()(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = PReLU()(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), padding='same', strides=2)(x)
    x = PReLU()(x)

    # bottleneck layer
    encoded = Flatten()(x)

    # layers for expansion of image
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = PReLU()(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = PReLU()(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), padding='same')(x)
    x = PReLU()(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (3, 3), padding="same")(x)
    x = PReLU()(x)
    x = UpSampling2D((2, 2))(x)
    decoded = Conv2D(c, (3, 3), activation='sigmoid', padding='same')(x)

    decoder = Model(input_img, decoded)
    encoder = Model(input_img, encoded)
    # the encoder will be trained through the decoder so it does not need to be
        compiled
    decoder.compile(optimizer='adam', loss='binary_crossentropy')

    return decoder, encoder
```

Listing A.1: *Keras code for initialising the autoencoder model developed here. It contains 5 downsampling and upsampling operations. It returns both its encoder and decoder parts. The decoder is used to evaluate the performance of the autoencoder at image reconstruction. The encoder is used to encode images to project them onto a two-dimensional plane using t-SNE and UMAP, and is also the building block for our regression model. Started from a tutorial by Chollet (2016), and was expanded through experiments and research.*

6 Bibliography

Y. Al-Kofahi, A. Zaltsman, R. Graves, W. Marshall, and M. Rusu. A deep learning-based algorithm for 2-D cell segmentation in microscopy images. *BMC Bioinformatics*, 19(1), Oct. 2018. doi: 10.1186/s12859-018-2375-z.

L. Aprupe, G. Litjens, T. J. Brinker, J. van der Laak, and N. Grabe. Robust and accurate quantification of biomarkers of immune cells in lung cancer micro-environment using deep convolutional neural networks. *PeerJ*, 7, Apr. 2019. doi: 10.7717/peerj.6335.

F. v. Beers, A. Lindström, E. Okafor, and M. A. Wiering. Deep Neural Networks with Intersection over Union Loss for Binary Image Segmentation. In *ICPRAM*, 2019. doi: 10.5220/0007347504380445.

R. A. Benson, M. K. MacLeod, B. G. Hale, A. Patakas, P. Garside, and J. M. Brewer. Antigen presentation kinetics control T cell/dendritic cell interactions and follicular helper T cell generation in vivo, Aug. 2015.

J. Brewster. The immune system: looking for love in all the right places. `https://www.youtube.com/watch?v=hRvyCYyab68`, 2015. Retrieved on 2020-03-17.

British Society for immunology. What is immunology?, 2020. URL `https://www.immunology.org/public-information/what-is-immunology`. Retrieved on 2020-03-31.

W. Buchser, M. Collins, T. Garyantes, R. Guha, S. Haney, V. Lemmon, Z. Li, and O. J. Trask. *Assay Development Guidelines for Image-Based High Content Screening, High Content Analysis and High Content Imaging*. Eli Lilly & Company and the National Center for Advancing Translational Sciences, Sept. 2014.

D. Bychkov, N. Linder, R. Turkki, S. Nordling, P. E. Kovanen, C. Verrill, M. Walliander, M. Lundin, C. Haglund, and J. Lundin. Deep learning based tissue analysis predicts outcome in colorectal cancer. *Scientific Reports*, 8, Feb. 2018. doi: 10.1038/s41598-018-21758-3.

M. Cavanagh and E. G. Findlay. T-cell activation, 2020. URL `https://www.immunology.org/public-information/bitesized-immunology/systems-and-processes/t-cell-activation`. Retrieved on 2020-03-31.

F. Chollet. Building Autoencoders in Keras. https://blog.keras.io/building-autoencoders-in-keras.html, May 2016. Retrieved on 2020-03-30.

Coenen and Pearce. Understanding UMAP. `https://pair-code.github.io/understanding-umap/`, 2019. Retrieved on 2020-03-28.

M. P. C. David, G. P. Concepcion, and E. A. Padlan. Using simple artificial intelligence methods for predicting amyloidogenesis in antibodies. *BMC bioinformatics*, 11, Feb. 2010. doi: 10.1186/1471-2105-11-79.

V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, March 2016.

K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr. 1980. doi: 10.1007/BF00344251.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323. PMLR, 11–13 Apr 2011.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, chapter 14. MIT Press, 2016. `http://www.deeplearningbook.org`.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1026–1034. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.123.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. July 2012.

A. Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/convolutional-networks, 2020. Retrieved on 2020-03-28.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL `http://arxiv.org/abs/1412.6980`. Retrieved on 2020-03-30.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud. A comprehensive analysis of deep regression. *CoRR*, abs/1803.08450, 2018.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4): 541–551, 1989.

G. Litjens, C. I. Sánchez, N. Timofeeva, M. Hermsen, I. Nagtegaal, I. Kovacs, C. Hulsbergen van de Kaa, P. Bult, B. van Ginneken, and J. van der Laak. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific Reports*, 6, May 2016. doi: 10.1038/srep26286.

L. McInnes and J. Healy. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, Feb. 2018.

H. C. Muh, J. C. Tong, and M. T. Tammi. AllerHunter: a SVM-pairwise system for assessment of allergenicity and allergic cross-reactivity in proteins. *PloS One*, 4(6), June 2009. doi: 10.1371/journal.pone.0005861.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814. Omnipress, 2010.

H. P. Ng, S. H. Ong, K. W. C. Foong, P. S. Goh, and W. L. Nowinski. Medical image segmentation using k-means clustering and improved watershed algorithm. In *2006 IEEE Southwest Symposium on Image Analysis and Interpretation*, pages 61–65, 2006.

A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

M. A. Rahman and Y. Wang. Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, F. Porikli, S. Skaff, A. Entezari, J. Min, D. Iwai, A. Sadagic, C. Scheidegger, and T. Isenberg, editors, *Advances in Visual Computing*, Lecture Notes in Computer Science, pages 234–244. Springer International Publishing, 2016. doi: 10.1007/978-3-319-50835-1_22.

W. Rawat and Z. Wang. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, 29(9):2352–2449, June 2017. doi: 10.1162/neco_a_00990. Publisher: MIT Press.

H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese. Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666. IEEE, June 2019. doi: 10.1109/CVPR.2019.00075.

A. Roghanian. Dendritic Cells. `https://www.immunology.org/public-information/bitesized-immunology/cells/dendritic-cells`, 2020. Retrieved on 2020-03-17.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, 2015.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

J. A. Saenz, N. Lubbers, and N. M. Urban. Dimensionality-reduction of climate data using deep autoencoders, 2018.

L. Shen, L. Margolies, J. Rothstein, E. Fluder, R. McBride, and W. Sieh. Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, 9:1–12, 08 2019. doi: 10.1038/s41598-019-48995-4.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint*, 2014.

J. Tregoning. Allergy, 2020. URL `https://www.immunology.org/public-information/bitesized-immunology/immune-dysfunction/allergy`. Retrieved on 2020-04-03.

R. Turkki, N. Linder, P. E. Kovanen, T. Pellinen, and J. Lundin. Antibody-supervised deep learning for quantification of tumor-infiltrating immune cells in hematoxylin and eosin stained breast cancer samples. *Journal of Pathology Informatics*, 7, Sept. 2016. doi: 10.4103/2153-3539.189703.

L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomput.*, 184(C):232–242, Apr. 2016. doi: 10.1016/j.neucom.2015.08.104.

M. Wattenberg, F. Viégas, and I. Johnson. How to use t-sne effectively. *Distill*, 2016. doi: 10.23915/distill.00002. URL `http://distill.pub/2016/misread-tsne`.

W. Xie, J. A. Noble, and A. Zisserman. Microscopy cell counting with fully convolutional regression networks. In *MICCAI 1st Workshop on Deep Learning in Medical Image Analysis*, 2015.

Y. Xue and N. Ray. Cell detection with deep convolutional neural network and compressed sensing. *CoRR*, abs/1708.03307, 2017.

L. Zamparo and Z. Zhang. Deep autoencoders for dimensionality reduction of high-content screening data. *CoRR*, 2015.