UNIVERSITÀ DELLA CALABRIA

**UNIVERSITA' DELLA CALABRIA**

Dipartimento di Matematica e Informatica

**Dottorato di Ricerca in**

Matematica e Informatica

**CICLO**

**XXXIII**

# Arsenic Ore Mixture Froth Image Generation with Neural Networks and a Language for Declarative Data Validation

**Settore Scientifico Disciplinare** INF/01 INFORMATICA

**Coordinatore:** Ch.mo Prof. Gianluigi Greco

Firma Gianluigi Greco

**Supervisori:** Prof. Mario Alviano

Firma

Prof. Carmine Dodaro

Firma

**Dottorando:** Dott. Arnel D. Zamayla

Firma

# Acknowledgements

I am truly grateful and honored to have been supported by the following people or institutions. In one way or another they have been my rock:

- To my thesis adviser Prof. Mario Alviano for his amazing support, encouragement and guidance. Without him I would have succumb already into discouragement.
- To my loving family and friends who helped me channel my frustrations into meaningful conversations.
- To the Mindanao State University - Iligan Institute of Technology for the funding.
- To the Department of Science and Technology - Engineering Research for Technology Faculty development Program, my biggest gratitude for the scholarship which enabled me to study and conduct research abroad.
- To Curtin University and Prof. Chris Aldrich for the guidance and support that was provided by them during my stay in Australia.
- To UNICAL and particularly DEMACS for providing an avenue for me to conduct my research.

# Abstract

Computer vision systems that measure froth flow velocities and stability designed for flotation froth image analysis are well established in industry, as they are used to control material recovery. However flotation systems that has limited data has not been explored in the same fashion bearing the fact that big data tools like deep convolutional neural networks require huge amounts of data. This lead to the motivation of the research reported in the first part of this thesis, which is to generate synthetic images from limited data in order to create a froth image dataset. The image synthesis is possible through the use of generative adversarial network. The performance of human experts in this domain in identifying the original and synthesized froth images were then compared with the performance of the models. The models exhibited better accuracy levels by average on the tests that were performed. The trained classifier was also compared with some of the established neural network models in the literature like the AlexNet, VGG16 ang ResNet34. Transfer learning was used as a method for this purpose. It also showed that these pretrained networks that are readily available have better accuracy by average comapared to trained experts.

The second part of this thesis reports on a language designed for data validation in the context of knowledge representation and reasoning. Specifically, the target language is Answer Set Programming (ASP), a logic-based programming language widely adopted for combinatorial search and optimization, which however lacks constructs for data validation. The language presented in this thesis fulfills this gap by introducing specific constructs for common validation criteria, and also supports the integration of consolidated validation libraries written in Python. Moreover, the language is designed so to inject data validation in ordinary ASP programs, so to promote fail-fast techniques at coding time without imposing any lag on the deployed system if data are pretended to be valid.

# Dedication

*Dedicated to our loving Creator , our Lord and Saviour Jesus Christ.*

# Contents

# List of Figures

# List of Tables

# Part I

# Arsenic Ore Mixture Froth Image Synthesis and Classification Using Deep Convolutional Generative Adversarial Network

# Chapter 1

# Introduction

The advent of the steam engine in the 18th century led to the first industrial revolution. Since then, it spurred a highly urbanized society that is geared towards automation through the use of computer technology to what is now the Fourth Industrial Revolution or Industry 4.0. Computers indeed have forever changed the landscape in the banking, medical, military, communications, engineering and business processes among the many.

According to the author of the Industry 4.0 and the chairman of the World Economic Forum professor Klaus Schwab in a 2016 article, Schwab wrote that "Unprecedented and simultaneous advances in Artificial Intelligence (AI), robotics, the internet of things, autonomous vehicles, 3D printing, nanotechnology, biotechnology, materials science, energy storage, quantum computing and others are redefining industries, blurring traditional boundaries, and creating new opportunities. We have dubbed this the Fourth Industrial Revolution, and it is fundamentally changing the way we live, work and relate to one another."

Indeed, the promise of computers involved among the industries mentioned is potent. Particularly the field of Mining and Mineral Processing automation has highly benefited from the advancement of Machine Learning in recent years [77, 114]. Because new computational technologies are making computers smarter, they can process vast amounts of data faster than ever before. Unlike the traditional method, deep learning boasts its ability to make meaning from data by discovering features and exploiting patterns that captures highly complex behavior. Sensor based systems and production lines relying on computer vision has paved the way for automation to propel, as huge amounts of data is an essential requirement to any deep learning project.

While deep learning algorithms performs ideally well with huge amounts of data it can also become its drawback. However, with the existence of transfer learning techniques this problem can be partly alleviated. In their study [46, 20], pretrained networks performed exceptionally well on data sets from new domains. In another study [44], they claimed

that large amounts of data may not be required to be optimized. Convolutional Neural Networks or CNN's have significantly pushed the boundaries about image recognition in a wide array of technical applications such as remote sensing, traffic and congestion, cancer diagnosis, and even face recognition but less has been explored in the geosciences particularly in the subset of mining and mineral processing.

Convolutional neural networks (CNNs), recurrent neural networks (RNNs) and deep belief networks (DBNs) are used most commonly in the resource industries. CNNs are primarily used in the computer-vision related tasks, such as image classification, object detection, semantic segmentation and instance segmentation. RNNs, including long short term memory networks (LSTMa) are sequence modelling.

Smart devices, appliances and gadgets are quite the popular ubiquitous in this digital age. However, it is not quite obvious that the materials which they are made comes from underneath the earth. The minerals hide inside deposits in the form of rocks which needs to be liberated through a process. This process typically comprises drilling, blasting, hauling, processing and transport of the liberated material of interest or metals. This research will focus on the "processing phase", particularly on froth flotation. Basically froth flotation is a method that is widely used to separate mineral and gangue species in aerated pulp. This liberation process produces froth which is captured by cameras on the production line so that laboratory technicians could analyze them using digital tools available.

The liberated materials retrieved from this process is then aggregated until it is transformed into wafer silicon, chips and other semiconductor materials like transistors, capacitors and resistors which comprise your smartphones, tablets, TV's and gadgets.

**Statement of the Problem**

Froth image datasets are not readily available for researchers in the field of Mineral Extraction. To get a single image from the mineral mining process, one has to go through drilling from above the ground, blasting the earth in regions of interest, hauling and chemical processing. To add more, mineral extraction requires skilled laboratory technicians in the field. These series of processes is laborious,time-consuming, costly, requires heavy equipment and trained manual labor to complete. Even if collection of froth image data could be readily accomplished on industrial plants, labelling of the data required for training may be more challenging [44]. To add more, even if datasets are available for a specific mixture these are protected by nondisclosure clause specific to every mining company that are difficult to navigate by the researchers in the field. This legal curb has restrained industry-academe collaboration and as a result hampered the establishment of a centralized and publicly available froth image dataset. To no avail of the froth images,this led to the motivation of this research to generate synthetic froth images from limited data. To be more specific the researchers sought to answer these questions.

1. Is it possible to generate synthetic arsenic ore mixture froth images from limited

training data?

2. From the eye of a trained expert, will the generated images be correct enough to be considered a representative of a certain class? and will an expert be able to identify which ones are original and which ones are fake?

3. Taking input from the generated synthetic froth image dataset, is it possible to create a classifier with an acceptable accuracy score?

4. How will a trained expert perform when compared to a machine in terms of accuracy in identifying the correct label of a froth image?

**Objectives of the Study**

In order to answer the questions mentioned above the researchers established the following objectives.

1. Design and implementation of a synthetic Image generator for arsenic ore froth images using Generative Adversarial Networks.

2. Design and implementation of an arsenic ore mixture froth image classifier using deep convolutional neural network.

3. Measure the trained expert accuracy score in identifying the original and fake images.

4. Measure the accuracy score of the designed arsenic ore mixture froth image classifier.

5. Measure trained expert accuracy score in image classification.

6. Compare the accuracy scores of the machine versus the trained expert.

**Significance of the Study**

As mentioned in the previous subsection, the highly manual process of mineral extraction is laborious, time-consuming and costly. Not to mention the involvement of trained experts in the process. To generate a single image is quite an involved process resource wise. As an alterative, generating fake images that are substantially correct and are distinctly identifiable into the correct labels we can save time, money and energy as a result. Moreover, we can then address the huge bump of having no training data to build deep learning model tools. Although froth image synthesis is still at its infancy in the body of knowledge in mineral extraction, the dataset that will be generated from this research can be utilized by other researchers for future work. As of this writing there is no research in the literature in the field of mineral extraction particularly in froth images that addresses the problem of

having no training data and the problem of creating an accurate classifier from a generated dataset.

**Scope and Limitation**

This research is limited only to arsenic ore mixture froth images that have been categorized into four classes. The Deep Neural network models developed is not designed and has not been tested to work with other froth mixtures of different chemistry. Also, the forth images examined in this research was produced using flotation method and likewise the deep neural network models developed is not designed and has not been tested to work with other methods. Although it was emphasized earlier that image generation can be used as an alternative for the lack of training data it does not mean this simulation completely replaces the mineral extraction process. Hence, it should be seen as a viable alternative to avoid the resource issues mentioned above.

**Organization**

The remainder of this thesis is structured as follows. Firstly some introductory concepts is described in Chapter 1. In chapter 2, we will review some related work about this research beginning with mineral processing and then move on to image synthesis and finally some work regarding classifiers and established neural network models is reviewed. Chapter 3 will follow to explore the methodology that was used which is then strengthened by the results and discussion on chapter 4. Chapter 5 is where the conclusions are discussed and it is followed by the bibliography.

## 1.1 Convolutional Neural Network

Images are invariant to several transformations such as scaling, translation and rotation. These challenges has plagued computer vision for years until a revolutionary way of designing Neural Network Model has been discovered. First introduced by LeCun et. al in his 1980 paper [79] , Convolutional Neural Networks (CNN's) belong to a class of deep, feed-forward artificial neural networks that have been applied to image analysis in many different disciplines.

The architecture of CNN as depicted in figure 1.1 is akin to the connectivity pattern of biological neurons and was inspired by the organization of the visual cortex. The architecture includes several building blocks, such as convolution layers, pooling layers and fully connected layers. In figure 1.1 the image of the car from the left will pass through each of these layers. Each layer extracts a specific feature from each level until the correct class of the image is achieved in the fully connected layer furthest to the right. Each layer

produces a feature map and down-samples the image at a minimum for faster convergence.

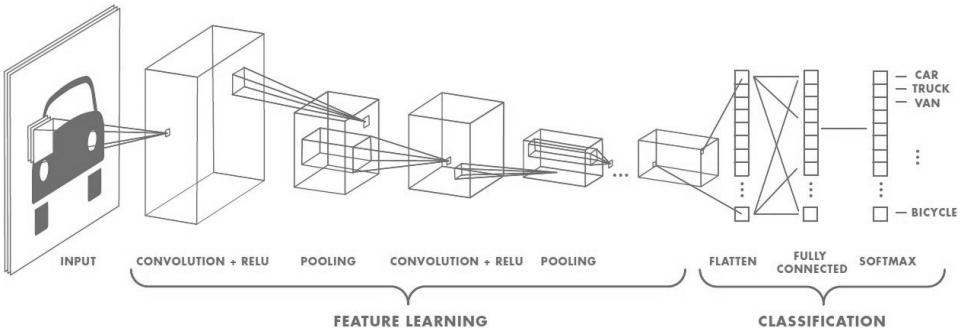


Figure 1.1: Typical Architecture of a Convolutional Neural Network.
(Courtesy of towardatascience.com, accessed 26 January,
2021,https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.)

The convolutional layer requires a few components, which are the vectorized input data, a filter and a feature map. The kernel or filter moves across the receptive fields of the image, checking if a feature is present. This process which is also known as convolution is better visualized in figure 1.2.

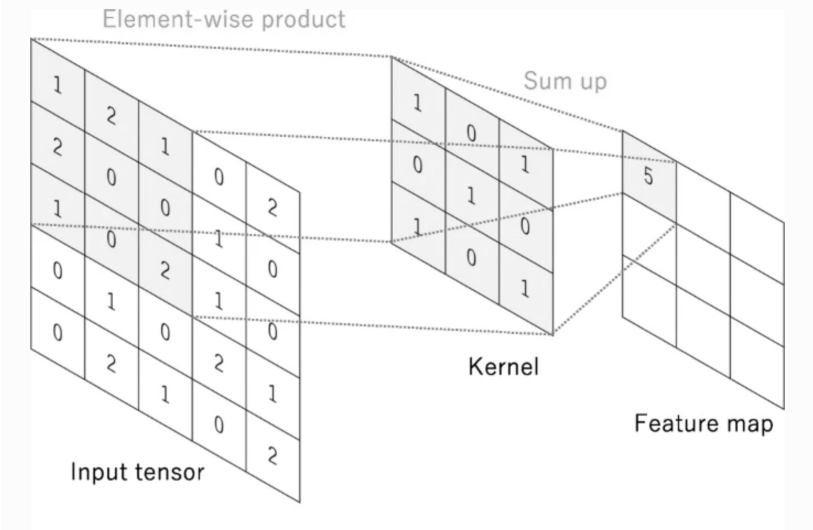


Figure 1.2: Convolution of a 5x5 image with a 3x3 filter, resulting in a smaller set of 3x3 feature. Image taken from [123]

Dimensionality Reduction is introduced in the pooling layer, reducing the number of parameters in the input. This is achieved by selecting the pixel with the maximum value to

to send to the output array across the input. This layer helps reduce complexity, improve efficiency and in some ways limit the risk of overfitting. This process is illustrated in figure 1.3.

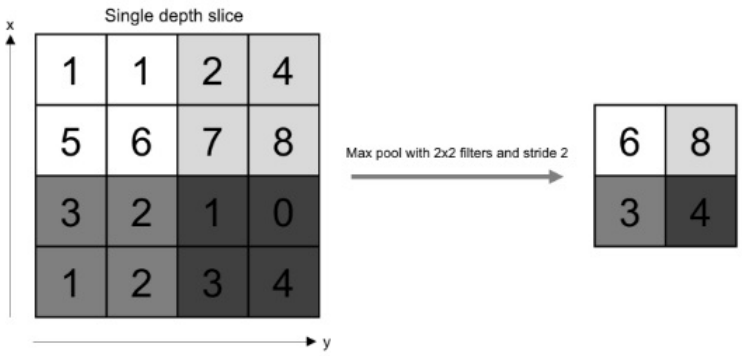


Figure 1.3: Max-pooling of a 4x4 input, reduced to a 2x2 set of max-pooled features

To introduce non-linearity to the previous linear activation layers, a nonlinear activation function such as rectified linear unit is employed. ReLU simply computes the function: $f(x) = max(0,x)$. This is done before it passes through another set of pooling layers until the matrix finally flattened to become a fully connected layer with the number of output nodes corresponding to the number of classes. To give you a better perspective in graphical form please see figure 1.4.

Training of CNN's is accomplished by use of backpropagation and stochastic gradient descent [109]. Like other deep neural networks, over the last few years CNN's have shown an amazing capacity to capture complex features. These networks have emerged as state-of-the-art approcahes to image recognition, often outperforming traditional approaches by a huge margin [115, 71].

## 1.2 Generative Adversarial Network

A generative model $G$ paremeterized by $\theta$ takes as input a random noise z and output a sample $G(z;\theta)$, so the output can be regarded as a sample drawn from a distribution: $G(z;\theta) \sim p_g$. Meanwhile we have a lot of training data x drawn from $p_data$, and the training objective for the generative model $G$ is to approximate $p_{data}$ using $p_g$.

Generative Adversarial Networks (GAN) [53] consists of two separate neural networks: a generator $G$ that takes a random noise vector $z$, and outputs synthetic data $G(z)$; a discriminator $D$ that takes an input $x$ or $G(z)$ and output a probability $D(x)$ or $D(G(z))$

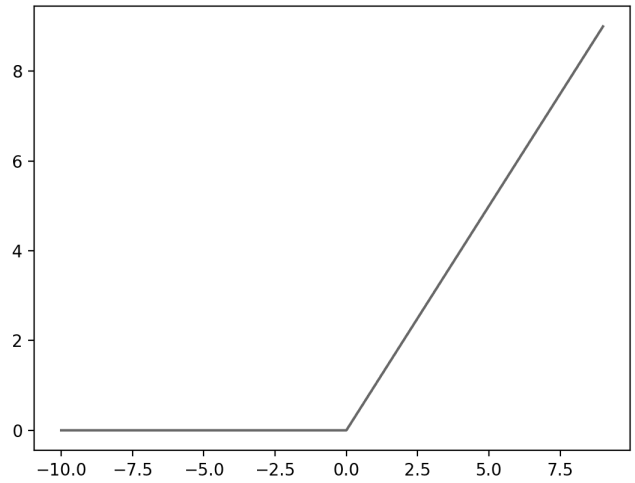


Figure 1.4: ReLU

to indicate whether it is synthetic or from the true data distribution, as shown in Figure 1.5. Both if the generator and the discriminator can be arbitrary neural networks. The first GAN [53] uses fully connected layer as its building block. Later, DCGAN [103] proposes to use fully convolutional neural networks which achieves better performance, and since then convolution and transposed convolution layers have become the core components in many GAN models. For more details on (transposed) convolution arithmetic, please refer to this report [40].

The original way to train the generator and discriminator is to form a two-player min-max game where the generator $G$ tries to generate realistic data to fool the discriminator while discriminator $D$ tries to distinguish between real and synthetic data [53]. The value function to be optimized is shown in Equation 1.1 where $p_{data}(x)$ denotes the true data distribution and $p_z(z)$ denote the noise distribution.

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1-D(G(z)))] \tag{1.1}$$

However, when the discriminator is trained much better than the generator,$D$ can reject the samples from $G$ with confidence close to 1, and thus the loss $log(1-D(G(z)))$ saturates and $G$ can not learn anything from zero gradient. To prevent this, instead of training G to minimize $log(1-D(G(z)))$, we can train it to maximize $logD(G(z))$ [53]. Although the new loss function for G gives a different scale of gradient than the original one, it still provides the same direction of gradient and does not saturate.

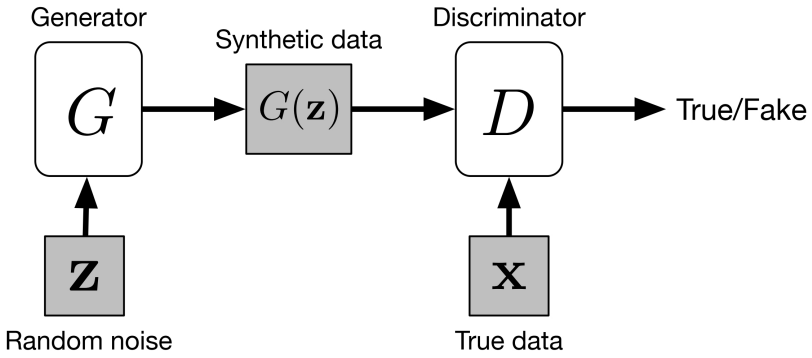


Figure 1.5: General structure of a GAN, where the generator $G$ takes a noise vector z as input and output synthetic sample $G(z)$, and the discriminator takes both synthetic input $G(z)$ and true sample x as inputs and predict whether they are real or fake.

### 1.2.1 Conditional GAN

In the original GAN, we have no control of what to be generated, since the output is only dependent on random noise. However, we can add a conditional input $c$ to the random noise $z$ so that the generated image is defined by $G(c;z)$ [88]. Typically, the conditional input vector $c$ is concatenated with the noise vector $z$, and the resulting vector is put into the generator as it is in the original GAN. Besides, we can perform other data augmentation on $c$ and $z$, as in [127]. The meaning of conditional input $c$ is arbitrary, for example, it can be the class of image, attributes of object [88] or an embedding of text descriptions of the image we want to generate [105] [106].

### 1.2.2 GAN with Auxillary Classifier

In order to feed more side-information and to allow for semi-supervised learning, one can add an additional task-specific auxiliary classifier to the discriminator, so that the model is optimized on the original tasks as well as the additional task [96] [111]. The architecture of such method is illustrated in Figure 1.6, where $C$ is the auxiliary classifier. Adding auxiliary classifiers allows us to use pre-trained models (e.g. image classifiers trained on ImageNet), and experiments in AC-GAN [96] demonstrate that such method can help generating sharper images as well as alleviate the mode collapse problem. Using auxiliary classifiers can also help in applications such as text-to-image synthesis [35] and image-to-image translation [126].

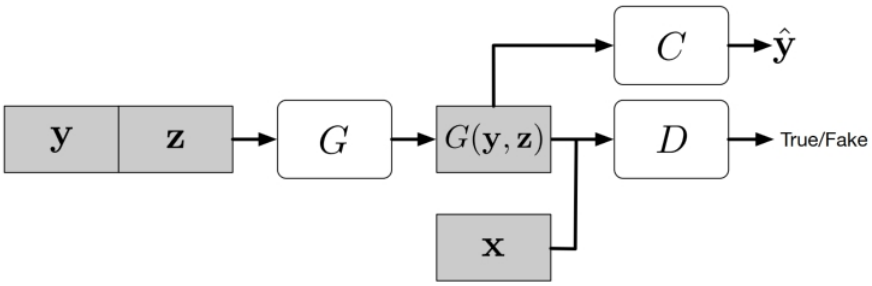


Figure 1.6: Architecture of GAN with auxiliary classifier, where $y$ is the conditional input label and $C$ is the classifier that takes the synthetic image $G(y;z)$ as input and predict its label $\hat{y}$

### 1.2.3 GAN with Encoder

Although GAN can transform a noise vector z into a synthetic data sample $G(z)$, it does not allow inverse transformation. If we treat the noise distribution as a latent3 feature space for data samples, GAN lacks the ability to map data sample x into latent feature $z$. In order to allow such mapping, two concurrent works BiGAN [38] and ALI [39] propose to add an encoder $E$ in the original GAN framework, as shown in Figure 1.7 . Let $\Omega_x$ be the data space and $\Omega_z$ be the latent feature space, the encoder $E$ takes $x \in \Omega_x$ as input and produce a feature vector $E(x) \in \Omega_z$ as output. The discriminator $D$ is modified to take both a data sample and a feature vector as input to calculate $P(Y|x,z)$, where $Y = 1$ indicates the sample is real and $Y = 0$ means the data is generated by $G$.

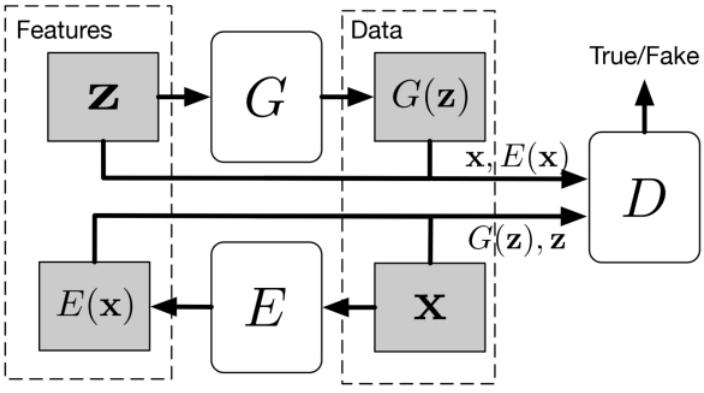


Figure 1.7: Architecture of BiGAN/ALI

The objective is thus defined as:

$$\min_{G,E}\max_{D} V(G,E,D) = E_{x\sim p_{data}(x)} logD(x,E(x)) + E_{z\sim p_z(z)}[log(1-D(G(z),z))] \quad (1.2)$$

VAE-GAN [78] proposes to combine Variational AutoEncoder (VAE) [73] with GAN [53] to exploit both of their benefits, as GAN can generate sharp images but often miss some modes while images produced by VAE [73] are blurry but have large variety. The architecture of VAEGAN is shown in Figure 1.8. The VAE part regularize the encoder $E$ by imposing a prior of normal distribution (e.g. $z \sim N(0;1)$), and the VAE loss term is defined as:

$$\mathscr{L}_{VAE} = -\mathbb{E}_{z\sim q(z|x)} log[p(x|z)] + D_{KL}(q(z|x)||p(x)), \quad (1.3)$$

where $z \sim E(x) = q(z|x), x \sim G(z) = p(x|z)$ and $D_{KL}$ is the Kullback-Leibler divergence.

Also, VAE-GAN [78] proposes to represent the reconstruction loss of VAE in terms of the discriminator $D$. Let $D_l(x)$ denotes the representation of the l-th layer of the discriminator, and a Gaussian observation model can be defined as:

$$p(D(x|z) = N(D(x)|D(\tilde{x}), I), \quad (1.4)$$

where $\tilde{x} \sim G(z)$ is a sample from the generator, and I is the identity matrix. So the new VAE loss is:

$$\mathscr{L}_{VAE} = -\mathbb{E}_{z\sim q(z|x)} log[p(D(x)|z)] + D_{KL}(q(z|x)||p(x)), \quad (1.5)$$

which is then combined with the GAN loss defined in Equation 1.2. Experiments demonstrate that VAE-GAN can generate better images than VAE or GAN alone.

However, when the discriminator is trained much better than the generator, $D$ can reject the samples from $G$ with confidence close to 1, and thus the loss $log(1-D(G(z)))$ saturates and $G$ can not learn anything from zero gradient. To prevent this, instead of training $G$ to minimize $log(1-D(G(z)))$, we can train it to maximize log $D(G(z))$ [53]. Although the new loss function for $G$ gives a different scale of gradient than the original one, it still provides the same direction of gradient and does not saturate.

### 1.2.4 Handling Mode Collapse

Although GAN is very effective in image synthesis, its training process is very unstable and requires a lot of tricks to get a good result, as pointed out in [53] [103]. Despite its instability in training, GAN also suffers from the mode collapse problem, as discussed in [53] [103] [36] .

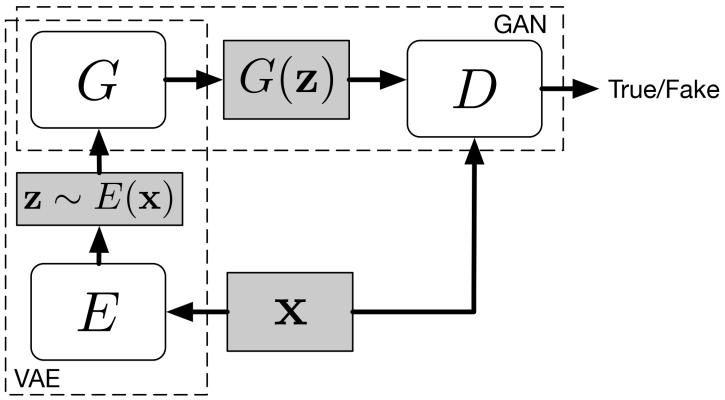


Figure 1.8: Architecture of VAE-GAN

In the original GAN formulation [53], the discriminator does not need to consider the variety of synthetic samples, but only focuses on telling whether each sample is realistic or not, which makes it possible for the generator to spend efforts in generating a few samples that are good enough to fool the discriminator. For example, although the MNIST [80] dataset contains images of digits from 0 to 9, in an extreme case, a generator only needs to learn to generate one of the ten digits perfectly to completely fool the discriminator, and then the generator stops trying to generate the other nine digits.

The absence of the other nine digits is an example of inter-class mode collapse. An example of intra-class mode collapse is, there are many writing styles for each of the digits, but the generator only learns to generate one perfect sample for each digit to successfully fool the discriminator. Many methods have been proposed to address the model collapse problem.

One technique is called minibatch features [6], whose idea is to make the discriminator compare an example to a minibatch of true samples as well as a minibatch of generated samples. In this way, the discriminator can learn to tell if a generated sample is too similar to some other generated samples by measuring samples' distances in latent space. Although this method works well, as discussed in [52], the performance largely depends on what features are used in distance calculation. MRGAN [29] proposes to add an encoder which transforms a sample in data space back to latent space, as in BiGAN [38].

The combination of encoder and generator acts as an auto-encoder, whose reconstruction loss is added to the adversarial loss to act as a mode regularizer. Meanwhile, the discriminator is also trained to discriminate reconstructed samples, which acts as another mode regularizer. WGAN [11] proposes to use Wasserstein distance to measure the similarity between true data distribution and the learned distribution, instead of using Jensen-

Shannon divergence as in the original GAN [53].Although it theoretically avoids mode collapse, it takes a longer time for the model to converge than previous GANs. To alleviate this problem, WGAN-GP [56] proposes to use gradient penalty, instead of weight clipping in WGAN. WGAN-GP generally produces good images and greatly avoid mode collapse, and it is easy to apply this training framework to other GAN models.

# Chapter 2

# Related Work

In this chapter we first introduce the involvement of deep learning in the mineral processing industry. We then discuss the how image is synthesized using GAN's. We cover the most common methods in the literature. We then wrap the chapter with learned loss functions and a few established neural network models that have paved the way to further research on image classification

## 2.1 Mineral Processing

The application of deep learning in the areas of ore and rock characterization, milling circuits and froth flotation have started to receive increasing attention in the recent studies with the advancement of deep learning. The subject under study is the froth image of arsenic mixture. However, before a froth image is produced it undergoes through a series of steps.

Firstly, in the mineral processing phase the ores need to be sorted and characterized for classification. It involves mineral content recognition and ore property estimation. Although it is not going to be discussed in this research. It is important to note that the distribution of particle sizes has a major impact on the remaining downstream processes. Image segmentation using recent successful techniques can therefore be explored in this area. Conventional methods such as watershed algorithms are usually contrast sensitive which can perform poor in low contrast settings. When segmenting particles with presence of minerals with similar colours in images it therefore generally tends to fail [69]. Traditionally, watershed algorithms have been used in the mining industry to classify images, in fact a more advance version of it has been reported which is called deep watershed algorithm [32, 14] to label frames.

It has been known that traditional approaches are generally time consuming and may even require considerable effort in manual adjustment of parameters for contrast adjust-

ment and noise reduction [61]. To address the difficulties met of these conventional medods, SegNet [12] a convolutional autoencoder network was used to segment digital rock images. A data augmentation technique, namely hybrid pattern and pixel-based simulation (HYPPS), was employed to generate sufficient images for training the network [68]. In this approach, the CNN performs pixel-to-pixel labelling (segmentation), i.e. every pixel in the original image is classified as either 'particle' or 'background'.

However, the potential application of CNNs extends beyond binary classification tasks [82], such as ore sorting [69], as some network architectures, such as SegNet [12] , UNet [108] and LinkNet [28], are designed for semantic segmentation.

With this novel approach, the networks can segment images with multi-categorical objects. For example, in the context of ore and rock characterization, an image may consist of particles of different metal grades (e.g. waste and ore), as well as different mineral grains. An effective way to identify and count these particles can be beneficial to many aspects in mining operations [59] and civil industry [62].

Characterisation of the complex flow of granular solids from hoppers, bins and silos is still an open research issue and [3] have recently made use of a convolutional neural network to extract features from mass flow measurements to enable better identification of avalanching phenomena in the flow.

After ore characterization grinding and comminution follows, this process produces the desirable size of ore particles.Since deep learning is superior in capturing the highly nonlinear relationship from complex data, several studies have appeared in the literature, e.g., missing data imputation using variational autoencoders (VAEs) [87], and estimation of mill load levels using soft sensor data modelling with CNN [121]. [13] have trained a five-hidden layer perceptron model to predict ore production and crusher utilization.

The operating states of grinding circuits are monitored from multivariate time-series signals [117] as they are or transformed to 2D images using distance matrices method [17], followed by feature learning using CNNs.

Deep learning has focused mostly on the development of more reliable sensor systems for online grade and reagent estimation in flotation, specifically based on froth image analysis. Traditionally, froth image analysis has focused on three related problems, namely the (i) recognition of changes in operation conditions from the appearance of the froth, often in combination with other variables, the (ii) estimation of bubble size distributions, as well as (iii) online estimators of grade or other chemical species being floated. Any of a number of methods can be used, often highly effectively to deal with (i), but (ii) and (iii) are more challenging.

As far as (i) is concerned [81] have made use of a pretrained CNN to extract features from an antimony froth that could then be used as inputs to a classifier to identify aberrant froth conditions, but the specific advantages of this approach is not clear.

Little has been done as far as (ii) is concerned, although Stone Three in South Africa

has patented an approach to estimate bubble sizes from froth images, based on the use of CNNs. This is an area where there is considerable scope for improvement over traditional methods and more work in this area is currently underway based on architectures and variants thereof similar to U-net [108].

Regarding (iii), in comparatively analyses with other multivariate image methods, [60, 46, 45, 44] and [129] have shown remarkable advantages in accuracy to be gained from using CNNs in froth image analysis. This is a major step forward towards the online implementation of image sensors in advanced online control systems in flotation.

While a variety of deep learning architectures are currently being used in the mining and metallurgical industries, convolutional neural networks have seen most use by a large margin to date. This is related to the current focus on sensor data analytics, particularly image-based sensors used in the characterisation of particulate feeds, drone inspection systems, and also the processing of hyperspectral images and multivariate time series data

The availability of large amounts of data is key to the development and deployment of deep learning systems in the industry. Although rapid development in sensor systems have led to the collection and storage of such data, these raw data are not necessarily useful for model development. For example, in flotation systems, while easy to collect prodigious numbers of images, these images also need to be labelled, which could be a major bottleneck in the development of the sensors.

While digitilisation of the mining industry is advancing in many ways, the industry tends to adopt technology, rather than leading in its development. Insight into future development can therefore be gained from recent developments in manufacturing. In this area, some novel machine learning methodologies closely tied to deep learning are emerging. This includes deep reinforcement learning and adversarial learning.

Deep reinforcement learning facilitates complex decision making with obvious applications in advanced control, but also beyond this to potentially higher levels of intelligence that could benefit haulage vehicle fleet management or plantwide control in mineral processing [113] . A prerequisite for this is the collection of sufficient data in real-world settings beyond simulated environments, which remains a significant challenge [2].

Unlike deep reinforcement learning, which can be seen as an extension of an existing approach, adversarial learning [31] improves learning efficiency through construction of a generator and a discriminator. By so doing, it is not necessary to specify a reward or loss function of the system. This enables learning tasks that were not previously possible, and has seen applications in image synthesis, monitoring and pattern recognition that would directly impact mining and automation.

## 2.2 Image Synthesis with GAN's

In this section, we summarize the three main approaches used in generating images, i.e. direct methods, iterative methods and hierarchical methods respectively, which form the basis of all applications mentioned in this paper.

### 2.2.1 Direct Methods

All methods under this category follows the philosophy of using one generator and one discriminator in their models, and the structures of the generator and the discriminator are straight-forward without branches. Many of the earliest GAN models fall into this category, like GAN [53], DCGAN [103], ImprovedGAN [111], InfoGAN [30], f-GAN [95] and GANINT-CLS [105]. Among them, DCGAN is one of the most classic ones whose structure is used by many later models such as [30] [105] [100] [130].

The general building blocks used in DCGAN are shown in Figure 2.1, where the generator uses transposed convolution, batch-normalization and ReLU activation, while the discriminator uses convolution, batchnormalization and Leaky ReLU activation.This kind of method is relatively more straight-forward to design and implement when compared with hierarchical and iterative methods, and it usually achieves good results.

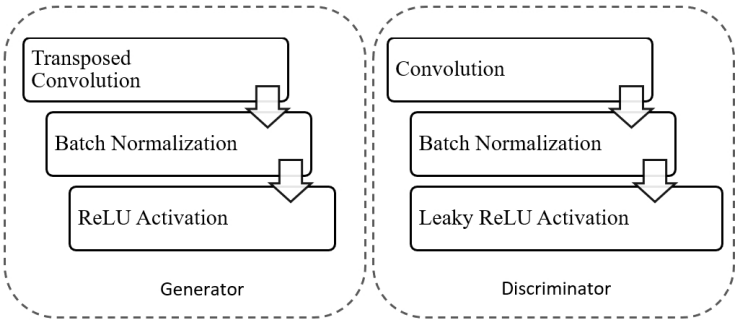


Figure 2.1: Building blocks of DCGAN, where the generator uses transposed convolution, batch-normalization and ReLU activation, while the discriminator uses convolution, batch-normalization and LeakyReLU activation

### 2.2.2 Hierarchical Methods

Contrary to the Direct Method, algorithms under the Hierarchical Method use two generators and two discriminators in their models, where different generators have different purposes. The idea behind those methods is to separate an image into two parts, like "styles and structure" and "foreground and background". The relation between the two genera-

tors may be either parallel or sequential. SS-GAN [120] proposes to use two GANs, a StructureGAN for generating a surface normal map from random noise $\hat{z}$ , and another Style-GAN that takes both the generated surface normal map as well as a noise $\hat{z}$ as input and outputs an image. The Structure-GAN uses the same building blocks as DCGAN [103], while the Style-GAN is slightly different. For Style-Generator, the generated surface normal map and the noise vector go through several convolutional and transposed convolutional layers respectively, and then the results are concatenated into a single tensor which will go through the remaining layers in Style-Generator.

As for the Style-Discriminator, each surface normal map and its corresponding image are concatenated at the channel dimension to form a single input to the discriminator. Besides, SS-GAN assumes that, a good synthetic image should also be used to reconstruct a good surface normal map. Under this assumption, SS-GAN designs a fully-connected network that transforms an image back to its surface normal map, and uses a pixel-wise loss that enforces the reconstructed surface normal to approximate the true one.

A main limitation of SS-GAN is that it requires to use Kinect to obtain groundtruth for surface normal maps. As a special example, LR-GAN [124] chooses to generate the foreground and background content using different generator, but only one discriminator is used to judge the images while the recurrent image generation process is related to the iterative method. Nonetheless, experiments of LR-GAN demonstrate that it is possible to separate the generation of foreground and background content and produce sharper images

### 2.2.3 Iterative Methods

This method differentiates itself from Hierarchical Methods in two ways. First, instead of using two different generators that perform different roles, the models in this category use multiple generators that have similar or even the same structures, and they generate images from coarse to fine, with each generator refining the details of the results from the previous generator.

Second, when using the same structures in the generators, Iterative Methods can use weight-sharing among the generators [124], while Hierarchical Methods usually can not. LAPGAN [36] is the first GAN that uses an iterative method to generate images from coarse to fine using Laplacian pyramid [24].

The multiple generators in LAPGAN perform the same task: takes an image from previous generator and a noise vector as input, and then outputs the details (a residual image) that can make the image sharper when added to the input image. The only difference in the structures of those generators is the size of input/output dimension, while an exception is that the generator at the lowest level only takes a noise vector as input and outputs an image. LAPGAN outperforms the original GAN [53] and shows that iterative method can generate sharper images than direct method. StackGAN [127], as an iterative method, has only two layers of generators. First generator takes an input $(z;c)$ and then outputs

a blurry image that can show a rough shape and blurry details of the objects, while the second generator takes $(z;c)$ and the image generated by the previous generator and then output a larger image with more photo-realistic details.

Another example of Iterative Methods is SGAN [21] which stacks generators that takes lower level feature as input and outputs higher level features, while the bottom generator takes a noise vector as input and the top generator outputs an image. The necessity of using separate generators for different levels of features is that SGAN associates an encoder, a discriminator and a Q-network [21] (which is used to predict the posterior probability $P(z_i|h_i)$ for entropy maximization, where $h_i$ is the output feature of the i-th layer) for each generator, so as to constrain and improve the quality of those features. An example of using weight-sharing is the GRAN [63] model, which is an extension to the DRAW [54] model which is based on variational autoencoder [73]. As in DRAW, GRAN generates an image in a recurrent way that feeds the output of the previous step into the model and the output of the current step will be fed back as the input in the next step. All steps use the same generator, so the weights are shared among them, just like classic Recurrent NeuralNetwork (RNN).

### 2.2.4 Other Methods

PPGN [92] produces impressive images in several tasks, such as class-conditioned image synthesis [88], text-to image synthesis [105] and image inpainting [125]. Different from other methods mentioned earlier, PPGN uses activation maximization [93] to generate images, and it is based on sampling with a prior learned with denoising autoencoder (DAE) [119]. To generate an image conditioned on a certain class label $y$, instead of using a feed-forward way (e.g. recurrent methods can be seen as feed-forward if unfolded through time), PPGN runs an optimization process that finds an input $z$ to the generator that makes the output image highly activate a certain neuron in another pretrained classifier (in this case, the neuron in the output layer that corresponds to its class label $y$). In order to generate better higher resolution images, ProgressiveGAN [70] proposes to start with training a generator and discriminator of 4×4 pixels, after which it incrementally adds extra layers that doubles the output resolution up to $1024 \times 1024$. This approach allows the model to learn coarse structure first and then focus on refining details later, instead of having to deal with all details at different scale simultaneously.

## 2.3 Discriminators: Learned Loss Functions

Generative adversarial network (GAN) is powerful and effective in that the discriminator acts as a learned loss function instead of a fixed one designed carefully for each specific task. This is particularly important for image synthesis tasks whose loss functions are hard to be explicitly defined in math. For example, in style transfer task, it is hard to write

down a math equation that evaluates how well an image matches a certain painting style. For image synthesis tasks, each input may have many legal outputs, but samples in training set cannot cover all situations. In this case, it is inappropriate to only minimize the distance between synthetic and ground-truth images, since we want the generator to learn the data distribution instead of remembering training samples. Although we can design feature-based losses that try to preserve feature consistency instead of at raw pixel level, as done in the perceptual loss [65] for image style, such losses are constrained by pre-trained image classification models they use, and it remains a question of which layers to pick for calculating feature loss when we switch to another pre-trained model. A discriminator, on the other hand, does not require explicit definition of the loss, since it learns how to evaluate a data sample as it trains against the generator. Thus the discriminator is able to learn a better loss function given enough training data. The fact that the discriminator acts as a learned loss function has significant meaning for general artificial intelligence. Traditional pattern recognition and machine learning require us to define what features to be used (e.g. SIFT [84] and HOG [34] descriptors), and we design specific loss functions and decide what optimization methods to be applied. Deep learning free us from carefully designing features, by learning low-level and high-level feature representations by itself during training (e.g. CNN kernels), but we still need to work hard at designing loss functions that work well. GAN takes us one step forward on our path towards artificial intelligence, in that it learns how to evaluate data samples instead of being told how to do so, although we still need to design the adversarial loss and combine it with other auxiliary losses. In other words, previously we design how to calculate how close an output is to the corresponding ground-truth $(\mathscr{L}(x;\hat{x}))$, but the discriminator learns how to calculate how well an output matches the true data distribution $(\mathscr{L}(x))$. Such property allows models to be more flexible and more likely to generalize well. Furthermore, with learn2learn [10] which allows neural networks to learn to optimize themselves, there is a possibility that we may no longer need to choose what optimizers (such as RMSprop [10], Adam [73] etc.) to use and let models handle everything themselves.

## 2.4 CNN Classifiers

### 2.4.1 AlexNet

AlexNet [77] is essentially an eight-layer convolutional neural network consisting of five convolutional layers and three fully connected layers. It was the winner of the ImageNet competition in 2012, with an error rate of approximately 16.4%. It is possible to use this network as a feature extractor, by replacing the original output layer with an appropriate layer relevant to the classification problem at hand. Figure 2.2 shows the simplified neural architecture.

### 2.4.2 VGG16

VGG16 consists of five blocks of convolutional and pooling layers, i.e. the first two blocks each has two convolutional layers, followed by a max pooling layer, while the next three blocks each has three convolutional layers followed by a max pooling layer (the sizes of the layers are indicated below and on top of layer blocks). This gives it 13 convolutional layers, as opposed to AlexNet's five. In addition, in VGG16 and VGG19, the large convolutional filters found in AlexNet are replaced with multiple smaller $3 \times 3$ kernel-sized filters.The stacks of smaller size filters increases the depth of the network, enabling it to learn more complex features [115]. Figure 2.3 shows the graphical perspective of this Architecture.

### 2.4.3 ResNet34

[57] have presented a residual learning framework for training of convolutional neural networks by explicit reformulation of the layers of the network as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. These residual networks are comparatively easy to optimise and allow construction of networks with a markedly increased depth and accuracy. As an example, on the ImageNet dataset [57] could apply residual networks with depths of up to 152 layers. With a 1st place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015 classification task (http://www.image-net.org/challenges/ LSVRC/), the network could achieve an error of 3.57% on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. Figure 2.4 exhibits the architecture in graphical form.

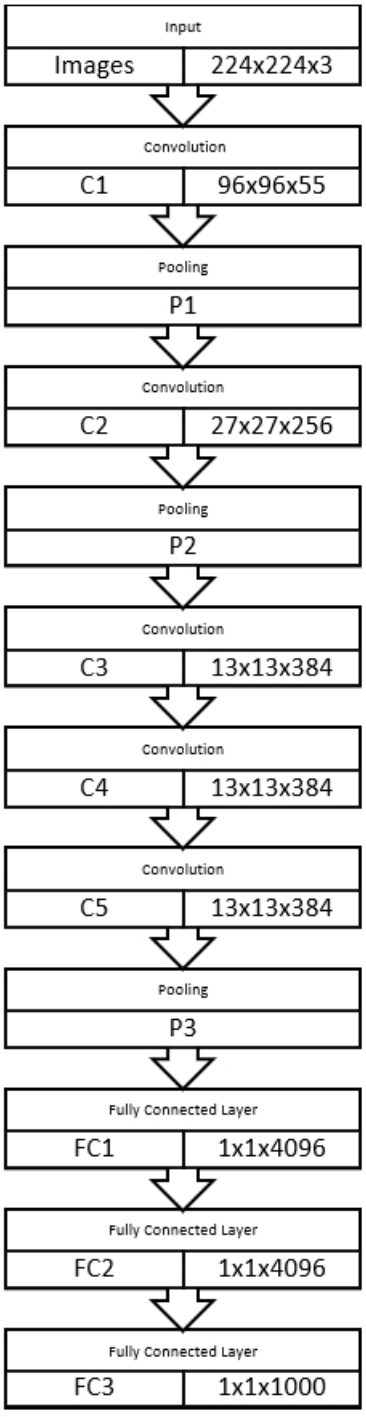


Figure 2.2: Simplified representation of the architecture of the AlexNet convolutional neural network, with five convolutional layers (C1, C2, ... C5), three pooling layers (P1, P2, and P3) and three fully connected layers (FC1, FC2 and FC3).

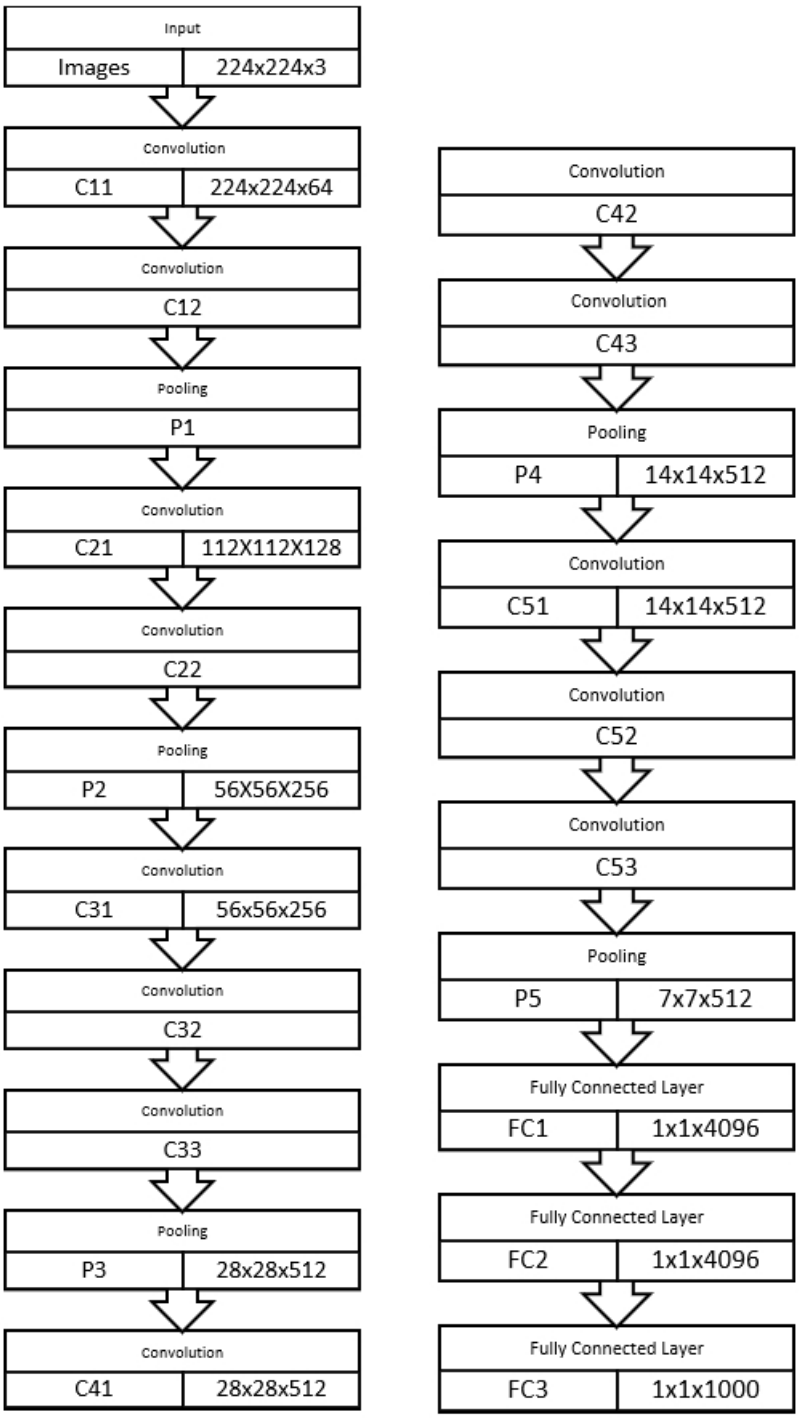


Figure 2.3: Simplified representation of the architecture of the VGG16 convolutional neural network, with 13 convolution layers (C11, C12, ... C53), five pooling layers (P1 to P5) and three fully connected layers (FC1, FC2 and FC3). The convolutional and pooling layers are arranged in five blocks, as indicated by the indices of the convolutional and pooling layers.

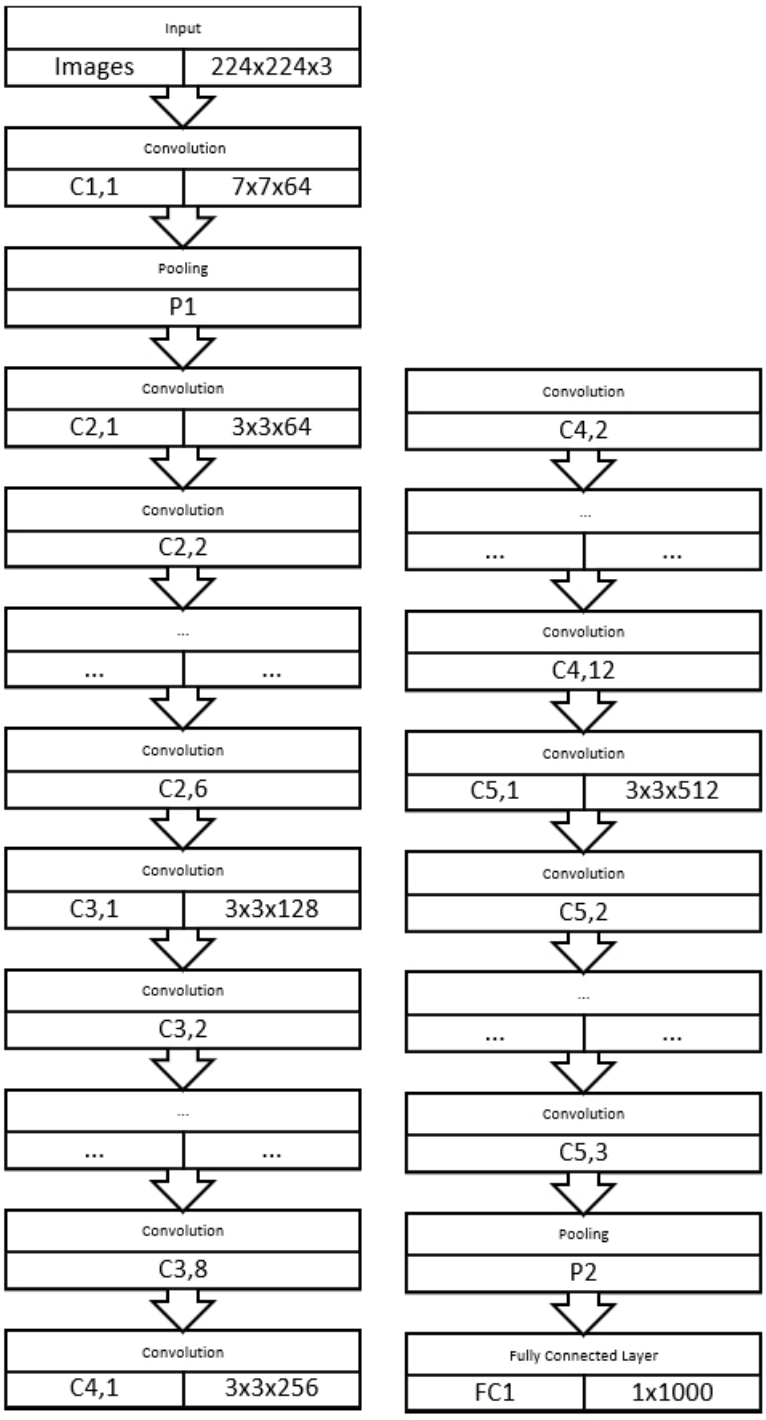


Figure 2.4: Simplified representation of the architecture of a 34-layer ResNet convolutional neural network showing convolutional layers (C1,1, C2,1, ... C5,3), pooling layers (P1 and P2) and a fully connected layer (FC1).

# Chapter 3

# Implementation

This chapter will discuss the methodology of the research in both hardware and software aspect. Split into two major sections, the first section describes the image synthesizer of the froth images which is the GAN and the second major section tackles the classifier which verifies the correctness of the generated synthetic froth images with respect to the corresponding class.

## 3.1 GAN: Image synthesis

### 3.1.1 Hardware and software

MATLABR2021b development environment was used to build both the GAN for the Image synthesis and DCNN for the classifier. The software was installed on a 64-bit windows PC, an Intel processor paired with a 16GB RAM, and an NVIDIA GeForce MX150 graphics card. The motivation for using MATLABR2021b was based on the existing tools that were already available for the lab technicians in Australia. But also for modelling Neural Networks, Mathworks inc. offers an Interoperability Framework for OpenSource Tools such as Tensorflow, Caffe2 and Pytorch. Allowing users to automate network design, training and experiment management in a seamless and easy way. In line with that the applications that are being deployed to embedded and production systems through automatic code generation in Matlab can benefit from the Automatic code generation. Automatic code generation generates optimized native code for Intel and ARM CPU's, FPGA's and SoC's and NVIDIA GPU's for Deep Networks along with pre-processing and post-processing, eliminating errors of transcription or interpretation.

### 3.1.2 Froth Images

The froth images that were used to train the neural networks were gathered from the mining industries in Australia in partnership with the Commonwealth Scientific and Industrial Research Organisation (CSIRO) through an internship program at Curtin University in Perth, Western Australia. Four images with a resolution of 1280 x 720 were presented to be synthesized, each image corresponds to a froth grade in an arsenic ore mixture using flotation method. Shown below are four of the original images which belongs to four different class.

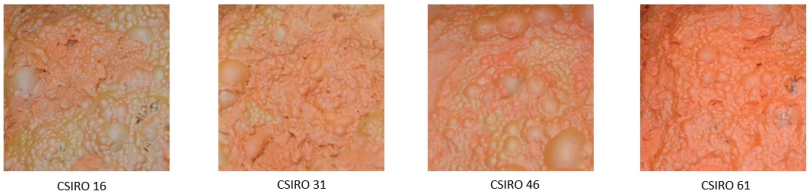


Figure 3.1: Generator network architecture

### 3.1.3 Data augmentation

Embark studios developed a light rust API for Multi-resolution Stochastic Texture Synthesis. This tool was used to help minimize the risk of overfitting which is a known problem with Neural Network models with limited data. The augmented images were generated by randomly rotating, shearing, shifting and flipping in the vertical and horizontal perspective.

### 3.1.4 Training the Image synthesizer

From the discussion earlier we already know that a GAN consists of two networks that are trained together

#### Generator

In the generator network, given a vector of random values as input, this network generates data with the same structure as the training data.To optimize the performance of the generator, the loss of the discriminator is maximized when given generated data. That is the objective of the generator is to generate data the discriminator classifies as "real".

In the network design from the above figure. Random vectors of size 100 are initialized at the beginning of the process these are converted to 7-by-7-by-128 arrays using a fully connected layer followed by a reshape operation. These are then upscaled to arrays

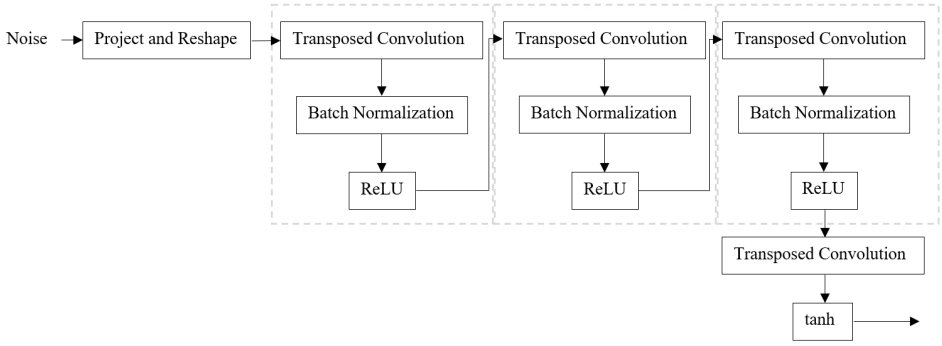


Figure 3.2: Generator network architecture

with 64-by-64-by-3 arrays using a series of transposed convolution layers with batch normalization and ReLU layers. For the transposed convolution layers, we specified 5-by-5 filters with a decreasing number of filters for each layer, a stride of 2, and cropping of the output on each edge.For the final transposed convolution layer, we specified three 5-by-5 filters corresponding to the three RGB channels of the generated images, and the output size of the previous layer. At the end of the network, we included a tanh layer. To project and reshape the noise input, we employed a fully connected layer followed by a reshape operation specifed as a function layer with a feature-to-image function.

**Discriminator**

In the discriminator network, it is fed with batches of data containing observations from both the training data, and generated data from the generator, this network attempts to classify the observations as "real" or "fake". To optimize the performance of the discriminator, the loss of the discriminator is minimized when given batches of both real and generated data. That is the objective of the discriminator is to not be "fooled" by the generator.Ideally, these strategies result in a generator that generates convincingly realistic data and a discriminator that has learned strong feature representations that are characteristic of the training data.

To create the discriminator network. A 64-by-64-by-3 images is taken which returns a scalar prediction score using a series of convolution layers with batch normalization and leaky ReLU layers. We added noise to the input image using dropout. To be more specific we specified the dropout layer with a dropout probability of 0.5. For the convolution layer we specified 5-by-5 filters with an increasing number of filters for each layer. Also the stride value is set to 2 with padding of the output across the layers.The leaky ReLU layers are specified a scale of 0.2. Finally for the final layer we specified a convolutional layer with one 4-by-4 filter. To output the probabilities in the range [0,1] we used the sigmoid

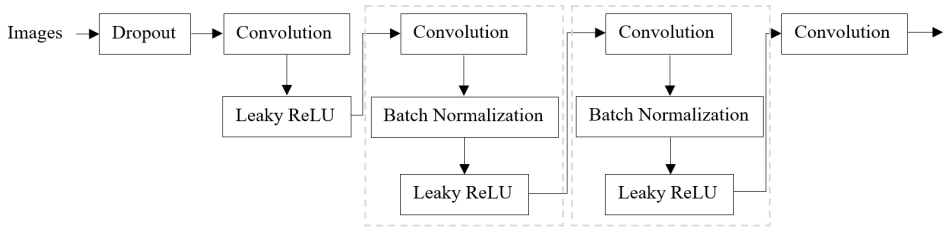


Figure 3.3: Discriminator network architecture

function. Figure 3.2 shows the design of the discriminator network.

### Loss Function and Scores

The objective of the generator is to generate data that the discriminator classifies as "real". To maximize the probability that images from the generator are classified as real by the discriminator, minimize the negative log likelihood function. Given the output $Y$ of the discriminator:

- $\hat{Y} = \sigma(Y)$ is the probability that the input image belongs to the class "real".
- $1 - \hat{Y}$ is the probability that the input image belongs to the class "generated".

Note that the sigmoid operation $\sigma$ happens in the modelGradients function. The loss function for the generator is given by

$$lossGenerator = -mean(log(\hat{Y}_{Generated})),$$

where $\hat{Y}_{Generated}$ contains the discriminator output probabilities for the generated images. The objective of the discriminator is to not be "fooled" by the generator. To maximize the probability that the discriminator successfully discriminates between the real and generated images, minimize the sum of the corresponding negative log likelihood functions. The loss function for the discriminator is given by

$$lossDiscriminator = -mean(log(\hat{Y}_{Real})) - mean(log(1 - \hat{Y}_{Generated})),$$

where $\hat{Y}_{Real}$ contains the discriminator output probabilities for the real images.

To measure on a scale from 0 to 1 how well the generator and discriminator achieve their respective goals, you can use the concept of score.

The generator score is the average of the probabilities corresponding to the discriminator output for the generated images:

$$scoreGenerator = mean(\hat{Y}_{Generated}).$$

| Parameter | value |
|---|---|
| numEpochs | 2000 |
| miniBatchSize | 128 |
| learnRate | 0.0002 |
| gradientDecayFactor | 0.5 |
| squaredGradientDecayFactor | 0.999 |
| flipFactor | 0.3 |
| validationFrequency | 100 |

Table 3.1: GAN Training Parameters

The discriminator score is the average of the probabilities corresponding to the discriminator output for both the real and generated images:

$$scoreDiscriminator = \frac{1}{2} mean(\hat{Y}_{Real}) + \frac{1}{2} mean(1 - \hat{Y}_{Generated}).$$

The score is inversely proportional to the loss but effectively contains the same information.

#### Training Parameters

The training parameters observed in this experiment are shown in Table 3.1. To avoid the discriminator from learning to discriminate from real and generated images too quickly, we added noise to the real data by randomly flipping the labels. This way there is a better balance in the learning of the discriminator and the generator.

## 3.2 CNN: Classifier

### 3.2.1 Training of the classifier

At the end of the process of generating synthetic images, a convolutional neural network is designed to verify the integrity of the newly created data. This neural network operates as a classifier which will test each images label with the correct class that it belongs to.

The input image passes through a set of convolution and pooling layers until it is flattened out and connected to a fully connected layer where the probability matrix will determine the class that the image belongs to. With a learning rate of 0.001 the network is trained for 500 epochs using Adam optimizer. For the purpose of easier name recall, we will call this FrothCNN.

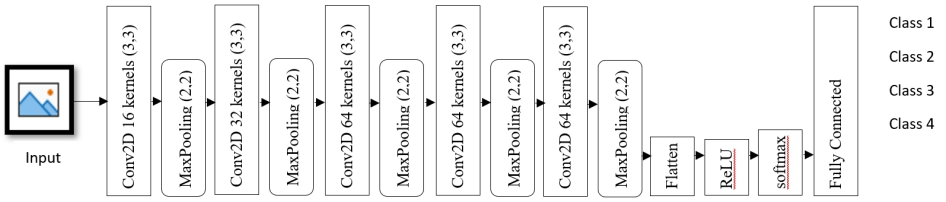


Figure 3.4: Convolutional Neural network Architecture of FrothCNN

### 3.2.2 Benchmarking

As a measure of benchmarking. The performance of the classifier is compared with the existing established Neural Network models. Discussed in their corresponding sections, these models have been pretrained on the ImageNet data base consisting of approximately 1.2 million images of 1000 common objects [110, 89]. As a result, they can be used on other image data bases without requiring retraining, except for modification of the final layer of the network in accordance with the specific problem under consideration. This technique is called transfer learning which is popular in the field of image classification.

#### Training of AlexNet

AlexNet was essentially used as a feature extractor, as described in [46] and further optimisation of the network was not attempted. That is, 4096 features were extracted from layer FC2 shown in figure 2.2. The final layer in the network was adapted to have four outputs – one for each of the classes in the froth image data base.

#### Training of VGG16

VGG16: As with AlexNet, VGG16 was used without modification, except for the final layer connecting to 1000 classes that was replaced with a new layer that connected to the four classes of froths in the database. As with AlexNet, this generated a 4096-dimensional feature vector for each image, which was then used to train the new fully connected layer.

# Chapter 4

# Results and Discussion

In this chapter we discuss the ouput of the neural network models we have designed in the previous chapters. We start by examining the qualities of an image and move on to the reason behind the design. We also present here the performance of the neural network when compared to human experts. The synthesized images and the classifier benchmarking results is also presented here.

## 4.1 Froth Image

Before we discuss the results of the trained neural networks. Let us investigate first the characteristics of the froth images which has driven the design specifics of the architecture of the neural networks.

When it pertains to images, a histogram is a graphical representation showing how frequently various color values occur in the image. Histogram is one of the simple ways to expose color changes between images and it is also quite handy as a preparatory step before performing Threshold or Edge Detection. We can see in figure 4.1 that the only noticeable mean comes from the red channel. The blue and green channels are almost the same. This is already obvious because the froth images vary on the color orange intensity which is highly affected by the red channel.

From the graph that we have shown in figure 4.1. We now have a clue on where to zone in. What we are actually interested on is the intensity of the color (how orange is the color) as opposed to the kind of color (e.g. blue , green or white) the froth image exhibits. By transforming the images into grayscale we can focus on the color intensity of each channel and by using the information we got from the histogram we should be able to expose which color channel has the huge effect on color reproduction of the images. Figure 4.2 shows that although there is a noticeable difference between the froth images as you go from top

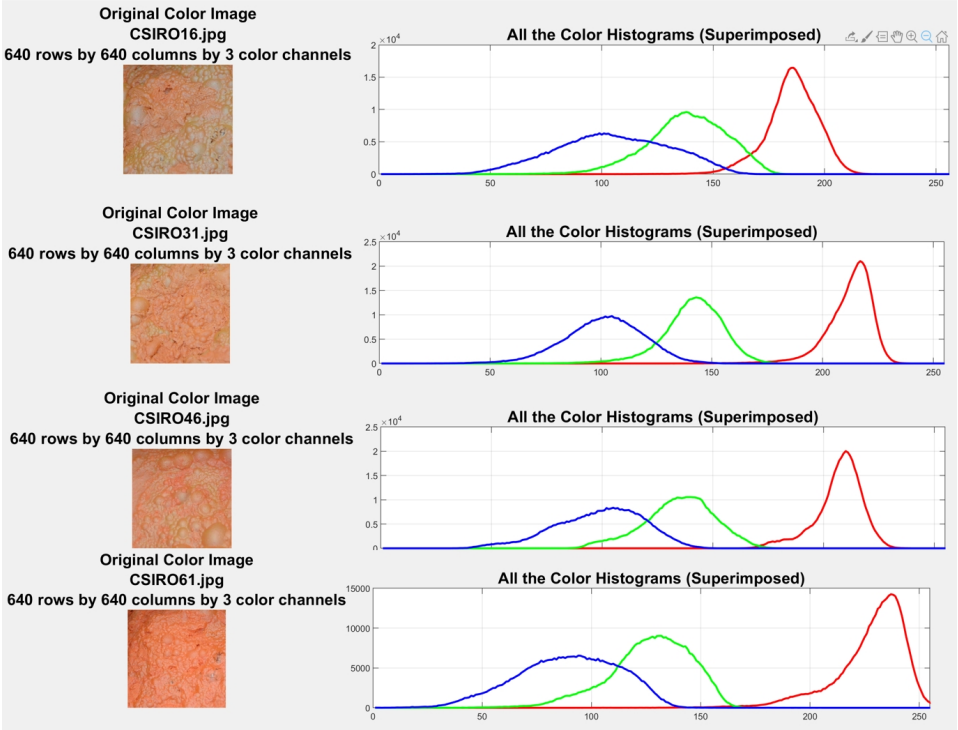


Figure 4.1: RGB Histogram

to bottom of the class in the red channel. The difference of the images from on the other class in green and blue channel is also noticeable. This should tell us that color alone is not a good unique feature which we can use to perfectly classify or synthesize these images. This is an established fact in literature.

Earlier, we investigated the color of the images using the RGB colorspace. In figure 4.3 we identified the colors of a froth image in l.a.b colorspace. Notice that instead of only three colors we are able to disect the image in 6 different colors. In l.a.b colorspace the colors are easier to distinguish from one another. The L*a*b color space is derived from the CIE XYZ tristimulus values. The L*a*b space consists of a luminosity ('L') or brightness layer, chromaticity layer 'a' indicating where color falls along the red-green axis, and chromaticity layer 'b' indicating where the color falls along the blue-yellow axis.The approach is to choose a small sample region for each color and to calculate each sample region's average color in 'a*b' space. The color markers are used to classify each pixel.

It is noticeable in figure 4.3 that the segmentation is not perfect and consistent. Although this is better than in the rgb colorspace we cannot use color alone as a feature to classify and synthesize this image because of the static nature of the threshold values to which this segmentation is highly dependent with.

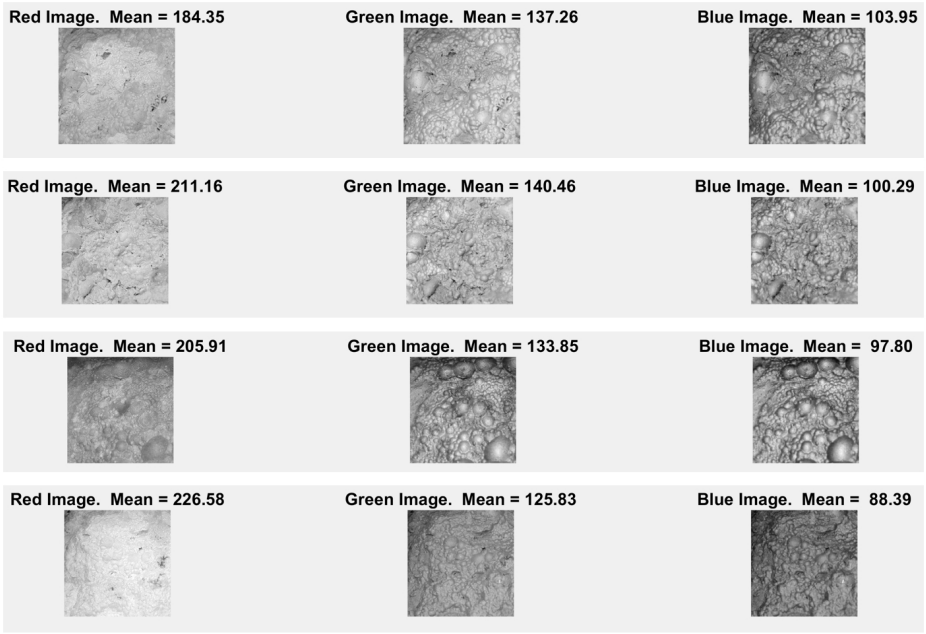


Figure 4.2: Grayscale image in R,G,B channels

Let us now look from a different perspective in understanding the unique features of the images being investigated. We can try to look at the validity of using texture as a unique feature to corretly classify or synthesize the image. By transforming the image into grayscale and use the intensity values of the RGB average. We can warp the image over the surface whose height is equal to the intensity of the image by specifying the number of graylevels. Figure 4.4 exhibits this 3D interpretation in grayscale. The 3D graph reveals the peaks and trenches of the texture plane. The peaks are the pixels nearing white and therefore values close to the 255 on the 0-to-255 range while the trenches or the dark regions are the complete opposite.

Although this information is helpful we don't have a unique way to associate the unique characteristics of each froth image belonging to a certain class. Features like: Physics of the bubbles, color of the bubbles, density of the bubbles in a region of interest, color consistency is not preserved and associated by techniques that are highly dependent on user specific threshold values. In fact, it is lost and cannot be learned in a meaningful way because it is a static implementation as opposed to a dynamic one. Which is one of the criteria we are looking for if we are going to synthesize these images. We do not want to synthesize the same image each time. So texture alone is also not a unique feature which we can use to classify or synthesize the froth images.

This led to the motivation to use Convolutional Neural Networks as they are great models in the literature known to preserve the unique characteristics of an image both

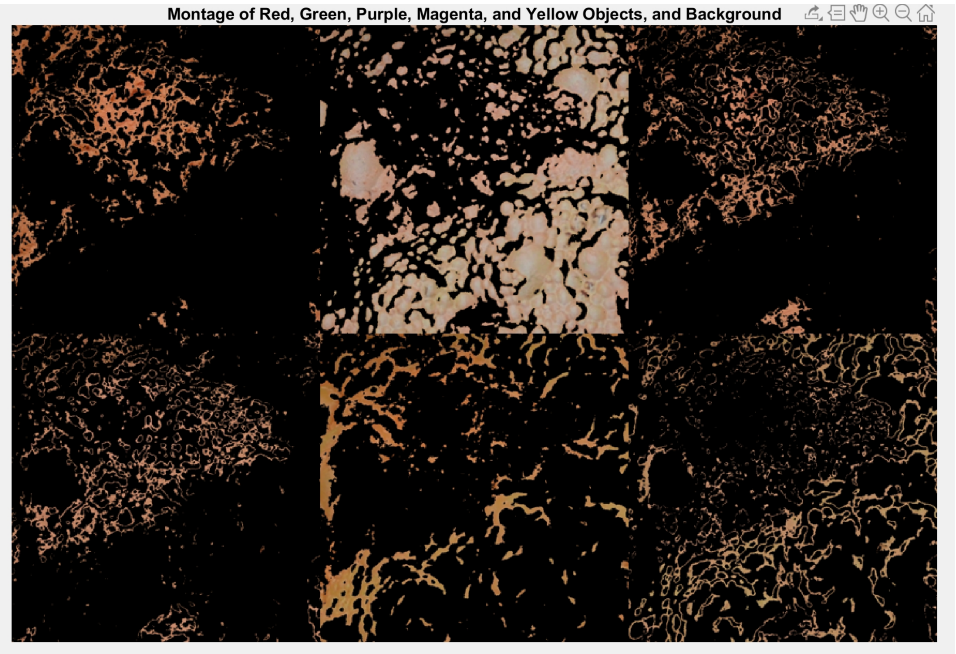


Figure 4.3: Froth image in L.a.b colorspace

from the color, texture and structure perspective. The results from the design and output of these networks are discussed in the next section.

## 4.2 GAN Results

### 4.2.1 Convergence

In the ideal case, both scores of the Discriminator and Generator would be 0.5. This is because the discriminator cannot tell the difference between real and fake images. However, in practice this scenario is not the only case in which you can achieve a successful GAN.

To monitor the training progress we can visually inspect the images over time and check if they are improving. If the images are not improving, then we can use a score plot to help us diagnose some problems. In some cases, the score plot can tell us there is no point continuing training, and we should stop, because a failure mode has occurred that training cannot recover from. The following sections tell us what to look for in the score plot and in the generated images to diagnose some common failure modes (convergence failure and mode collapse), and suggests possible actions we can take to improve the training. Convergence failure happens when the generator and discriminator do not reach a balance during training.

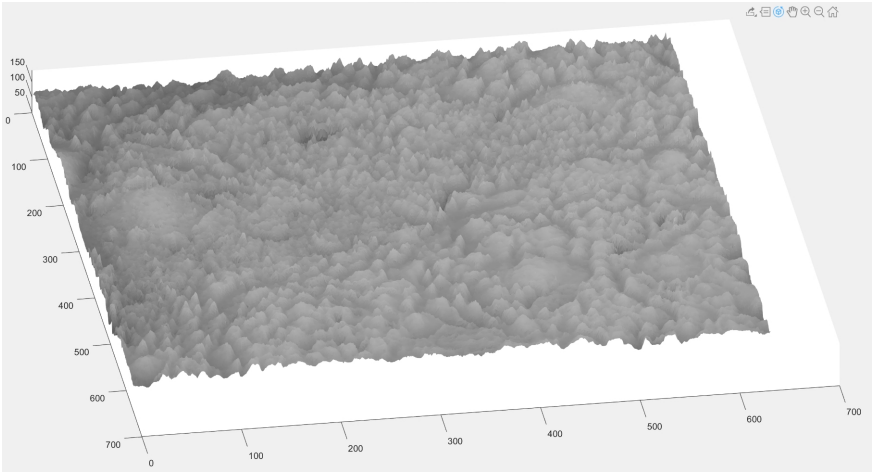


Figure 4.4: x- and y- coordinates of the surface were not specified. Thus defaulted to the image pixel indices.

### Discriminator Dominates

This scenario happens when the generator score reaches zero or near zero and the discriminator score reaches one or near one.

Figure 4.5 shows an example of the discriminator overpowering the generator. Notice that the generator score approaches zero and does not recover. In this case, the discriminator classifies most of the images correctly. In turn, the generator cannot produce any images that fool the discriminator and thus fails to learn.

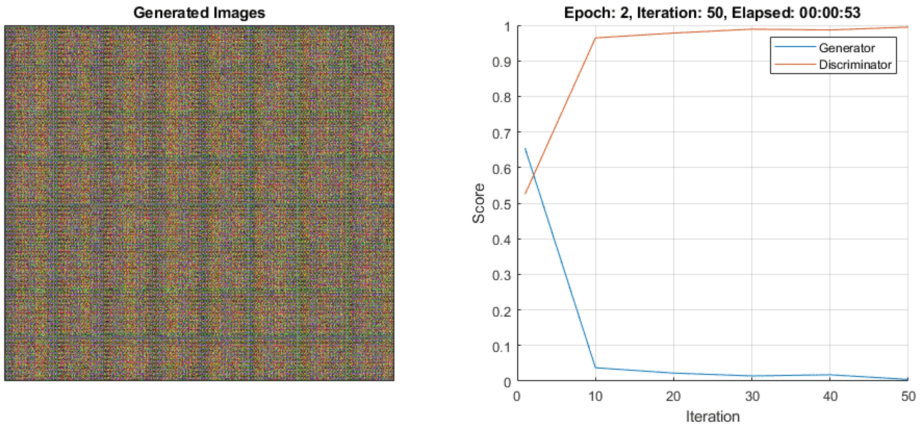


Figure 4.5: Discriminator Dominates

If the score does not recover from these values for many iterations, then it is better to stop the training. If this happens, then we can try balancing the performance of generator

and the discriminator by:

- Impairing the discriminator by randomly giving false labels to real images (one-sided label flipping).
- Impairing the discriminator by adding dropout layers.
- Improving the generator's ability to create more features by increasing the number of filters in its convolution layers.
- Impairing the discriminator by reducing its number of filters.

### Generator Dominates

This scenario happens when the generator score reaches one or near one. Figure 4.6 shows an example of the generator overpowering the discriminator. Notice that the generator score goes to one for a many iterations. In this case, the generator learns how to fool the discriminator almost always. When this happens very early in the training process, the generator is likely to learn a very simple feature representation which fools the discriminator easily. This means that the generated images can be very poor, despite having high scores. Note that in this example, the score of the discriminator does not go very close to zero because it is still able to classify correctly some real images. If the score does not

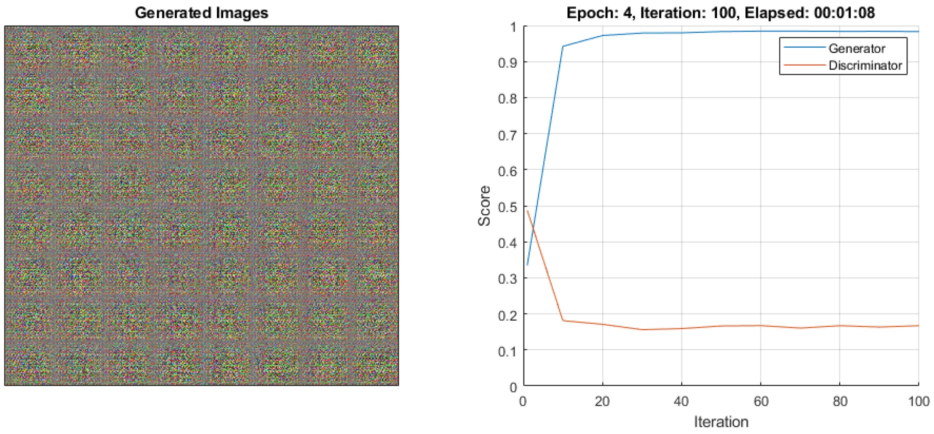


Figure 4.6: Generator Dominates

recover from these values for many iterations, then it is better to stop the training. If this happens, then try balancing the performance of generator and the discriminator by:

- Improving the discriminator's ability to learn more features by increasing the number of filters
- Impairing the generator by adding dropout layers

- Impairing the generator by reducing its number of filters

**Balanced Convergence**

Here, the discriminator has learned a strong feature representation that identifies real images among generated images. In turn, the generator has learned a similarly strong feature representation that allows it to generate images similar to the training data. Figure 4.7 shows the scores of the generator and discriminator networks

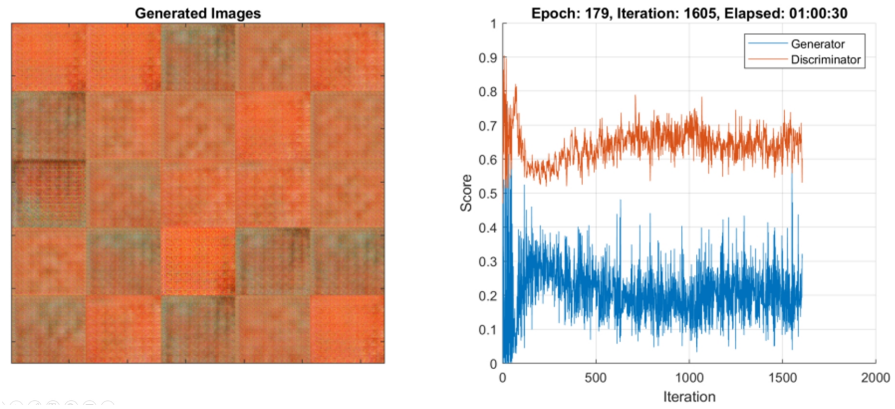


Figure 4.7: Balanced Convergence

### 4.2.2 Synthetic Image results

It is a given fact that it is difficult to assess the quality of synthetic images. RMSE is not a suitable metric since there is no absolute one-to-one correspondence between synthetic and real images. By visual inspection a commonly used subjective metric is to outsource Amazon Mechanical turk [9] that hires humans to score synthetic and real images according to how realistic they think the images are. However, this domain is expert-specific which requires a trained eye. To measure how good the synthetic images are, trained experts from the lab were given a set of images to evaluate. They were tasked to identify whether a froth image under examination is real or fake without reference to the correct answer. Tests show that by average the experts had difficulty in identifying the correct label of the images having an average accuracy of only 73%. This means that the generated synthetic images are convincing enough to the naked eye of a trained expert it confused them. We also wanted to know if given a set of images, can the experts correctly identify the class to which each image belongs to. It showed that by average the accuracy level is only at 67%.

We wanted to compare how our model would perform against a human expert using the same tests. For this purpose a classifier was designed. The results are shown in the next

section. For reference figures 4.8 and 4.9 show how close the original and the generated images are with each other in terms of visual inspection. The GAN model was able to generate a total of 4000 images, each class having 1000 samples. To build a classifier these are divided into a 70-30 ratio for training and testing.

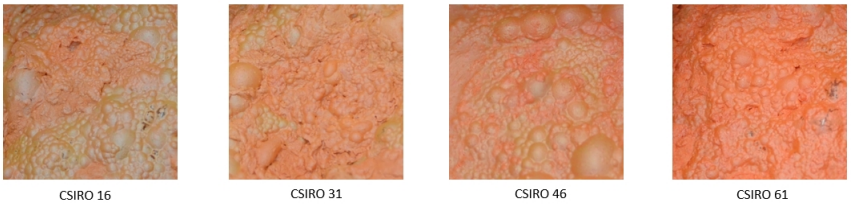


Figure 4.8: Original Froth Image

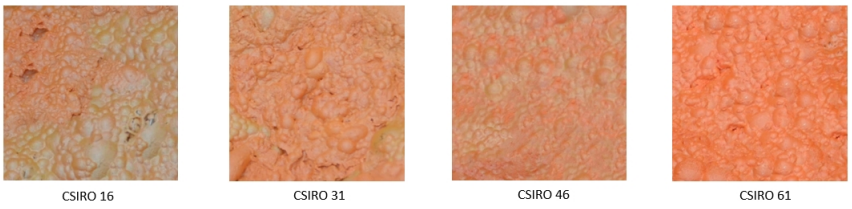


Figure 4.9: Generated Synthetic Froth Images

## 4.3 Classifier Results

The image features extracted by each network can be visualised by mapping them to a two-dimensional score space by use of linear discriminant analysis. As mentioned previously, these features were the outputs of the hidden nodes feeding into the final fully connected (output) layers of the networks, i.e. FC2 in AlexNet Fig 2.2 and VGG16 Fig 2.3. As can be seen in Fig 4.10, AlexNet and Fig 4.11,VGG16 perform similarly, with features that separate the froth classes well, although there are some overlapping among Class 2, Class 3 and Class 4.

However in figure 4.12, here we see the clear distinction between the different classes of our froth images. There is minimum overlapping among the data points and each chunk is significantly separated a good distance from each other. Figure 4.13 gives us a quantitative breakdown of the classification of the froth images in the test data set of FrothCNN. The model was able to classify 99%.

In light of comparison, the performance of the deep learning architectures against previously proposed methods for feature extraction from froth images are summarised in

Figure 4.14 as well. The two methods considered here are based on the use of a grey level co-occurrence matrices (GLCM) and local binary patterns (LBP). GLCMs are well-established in froth image analysis [18, 55, 128]. In contrast, have only more recently been considered in the context of froth image analysis by [74, 122], although they are used extensively in other domains in image recognition. The GLCM approach was based on the use of 256 grey levels, as well as the extraction of features based on five pixel distances (1, 2, 3, 4, 5) and four angles (0°, 45°, 90° and 135°). Six texture properties, namely contrast, dissimilarity, homogeneity, ASM, energy and correlation were computed from each GLCM matrix, in total yielding $4 \times 4 \times 6 = 120$ features for each image [112] With the LBP approach, the rotation invariant uniform method was selected along with a radius of 1 and the number of neighbouring points equal to 8, to yield 20 features [98]. The GLCM and LBP features were used as predictors with a random forest model([22] consisting of 300 trees. Like all the other models, the random forest models were trained and tested on the same training and test data sets. The results obtained with the different classifiers are summarised in figure 4.14.

## 4.4 Classifier Comparison

Three approaches have been explored to investigate the integrity of the generated synthetic froth images. Both GLCM and LBP use engineered froth features as predictor variables while pretrained networks AlexNet and VGG16 use learned features from a different domain and applied through transfer learning to the froth images. While the FrothCNN which learned exclusively from the generated synthetic froth images. In line with this it is quite obvious that learned features yielded better than engineered features. However this is not to say that the designed FrothCNN is better for all applications when compared to the mammoth-like AlexNet and VGG16. Learned features exclusively from the froth images helped in associating the correct label to the correct class compared to features learned from a different image dataset as applied to froth images. Even though FrothCNN has higher accuracy score it does not imply that it will perform the same in a different problem domain. The accuracy score is not the focus of the tests, rather it is a clear evidence that even though the froth images are synthesized or fake. They are classifiable into different class as evidenced by the results. This shows that the designed GAN is able to generate images that have meaning. That it is not just a three-dimensional matrix with random distribution.

The table below shows a summary of the performance of the synthetic image classifier compared with the established models in the literature.

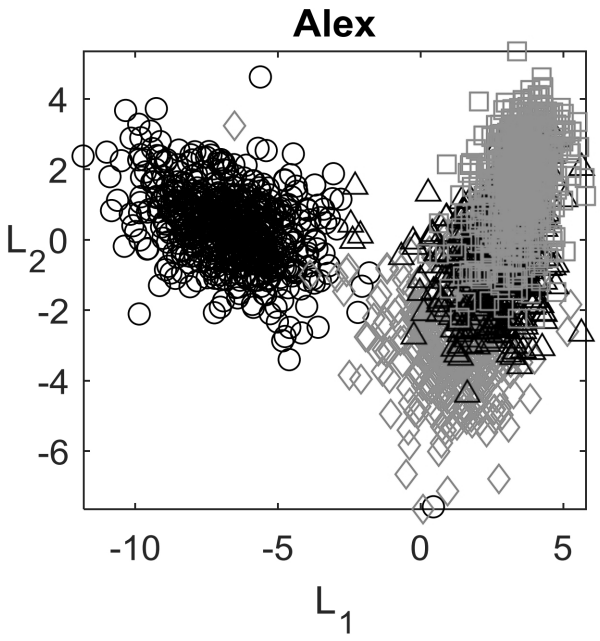


Figure 4.10: Linear Discriminant scatter plot of the features extracted by AlexNet, Class1(circle),Class2(diamond),Class3(triangle),Class4(square).

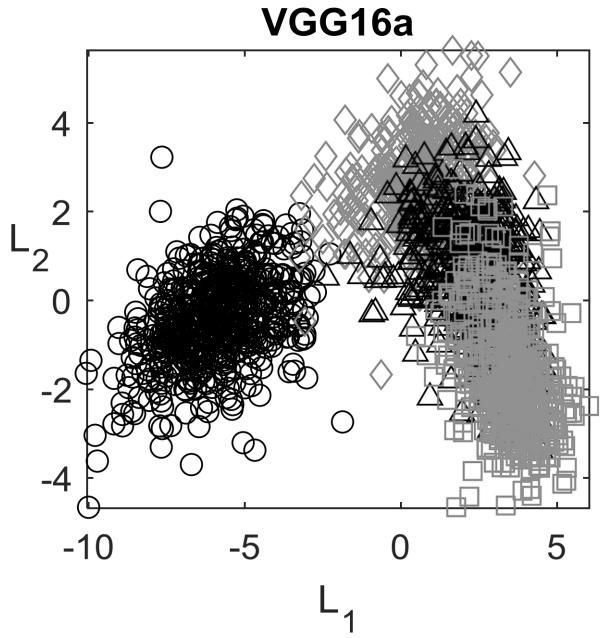


Figure 4.11: Linear Discriminant scatter plot of the features extracted by VGG16, Class1(circle),Class2(diamond),Class3(triangle),Class4(square).

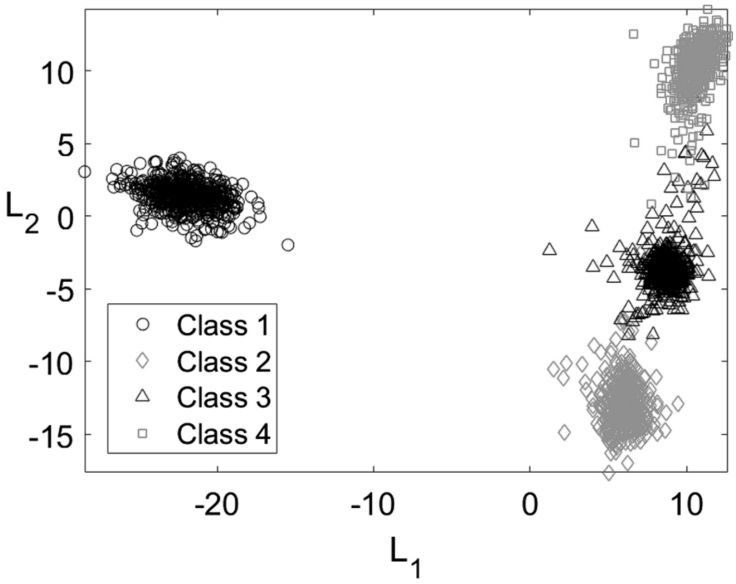


Figure 4.12: Linear Discriminant scatter plot of the features extracted by the FrothCNN, Class1(circle),Class2(diamond),Class3(triangle),Class4(square).

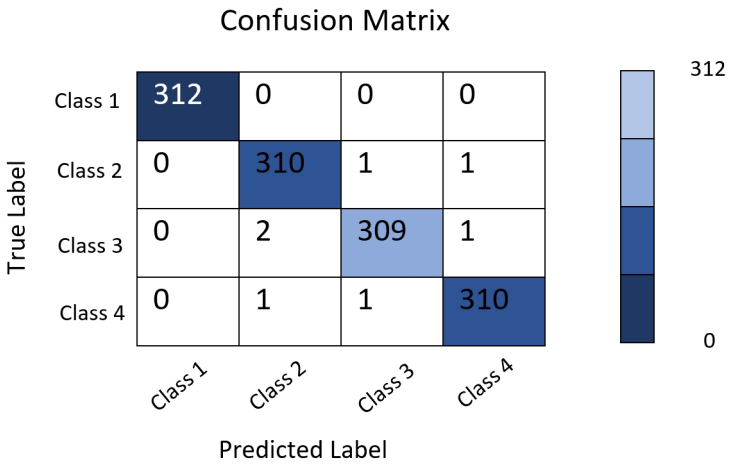


Figure 4.13: Classification of the fake froth images having an overall accuracy of 99% on the validation set.

| Classification Accuracy (%) of models on training and validation data sets | | | | | |
|---|---|---|---|---|---|
| Model | GLCM | LBP | AlexNet | VGG16 | FrothCNN |
| No. of Features | 120 | 20 | 4096 | 4096 | 512 |
| Training Data | - | - | 81.3 | 77.5 | 99.1 |
| Test Data | 73.6 | 69.1 | 82.1 | 74.2 | 99 |

Figure 4.14: Summary of results.

# Chapter 5

# Conclusion

Image synthesis from a limited set of data was considered in this research. The generated synthetic image were used to create a classifier. As a measure of benchmarking the performance of the the classifier was compared with the established neural network in the literature using transfer learning technique. The following conclusions are made from this study:

- The synthesis of fake froth images of arsenic ore mixtures from a limited source of data is possible using generative adversarial networks.
- The synthesized froth images were convincing enough to the eye of a trained expert, it was able to fool them. This only means that the GAN that was developed had above human expert accuracy level and was consistent in generating fake images.
- The methodology described in this research worked particularly well for this domain since the images does not really have a defined proper structure placement like that of a human face, a building or parts of an animal. Although this is just a hypothesis and not conclusive.
- The development of a classifier from a set of synthetic froth images was indeed successful in two ways:
  - It showed that the generated synthetic images were not just random three dimensional distribution. As clearly evidenced, they exhibit distinct features to their own class that they are classifiable. This better highlights the integrity of the generated synthetic froth image dataset produced by GAN as meaningful.
  - It has a better accuracy rate compared to a human expert in the domain by average.
  - It also has better overall accuracy when compared to the established models in the literature for this specific problem domain. Although some authors may

claim that an accuracy above 95% is considered borderline overfitting. However the parameters of the model can actually be tweaked to have better score on images not encountered during the training phase.

- Although the accuracy level is high, the classifier has not been tested on froth images acquired from other production lines. This is set as a recommendation for further research.
- Pre-trained networks that is available in the literature are actually not a bad choice in creating a classifier for a specific problem domain. In fact, it reported better accuracy compared to a trained expert.

Although the results are quite promising, further work is required to validate these results on different flotation systems which is currently hampered by a lack of suitable data sets. As 'big data' tools, deep convolutional neural networks require massive amounts of data. While the collection of froth image data could be readily accomplished on industrial plants, labelling of the data required for training may be more challenging . The researchers recommend synthesis of froth image on these systems to generate more data and to examine where it will take this area of research.

# Part II

# A Language for Declarative Data Validation in Answer Set Programming

# Chapter 6

# Introduction

A popular Latin saying starts with *errare humanum est* (translated, to err is human), and clarifies how making mistakes is part of human nature. Computer programmers, being humans, are inclined and not indifferent to errors [75, 76]. Whether a typo in notation, a misspelled word, or a wrong or fragile representation of data, errors in source code files may result in substantial delays in developing an application. Even worse, errors may stay unknown for a long time, until something happen that stimulates the error to cause a crash of the system or some other unwanted and unexpected behavior. In the worst scenario, unknown errors may lead to wrong results that are used to take some business decision, which in turn may ruin a company. (Refer to [91] for examples of typical errors in software systems.)

Fail-fast systems are designed to break as soon as an unexpected condition is detected (refer to [99] for an example of fail-fast type checking in Java). As often it is the case, the idea is not limited to computer programming, and definitely not originated in software design. For example, many electrical devices have fail-fast mechanisms to provide over-current protection — it is better to melt an inexpensive fuse, than to burn a control board. Even if technically an electrical devise operating with a fuse replaced by a wire works as expected in more cases, no professional electrician would suggest to tamper with the system in this way. In fact, in case the replaced fuses melt again and again, it is usually evidence that the system has some malfunction that must be detected and fixed.

Computer programming should follow a similar fail-fast approach. Errors must be detected as soon as possible, and reported to the programmer in a non-skippable way, so that the malfunction can be quickly detected and fixed. Data validation is the process of ensuring that data conform to some specification, so that any process in which they are involved can safely work under all of the assumptions guaranteed by the specification. In particular, immutable objects are usually expected to be validated on creation, so that their consistency can be safely assumed anywhere in the system — this way, in fact, an invalid

immutable object simply cannot exist because its construction would fail. Guaranteeing validity of mutable objects is usually more difficult and tedious, and almost impossible if there is no invariant on the validity of immutable objects. (Refer to [66] and [118] for details on how to tackle the complexity of a business domain in terms of domain primitives and entities.)

Answer Set Programming (ASP; [51, 94, 86]) should not be an exception when it comes at errors. However, the language offers very little in terms of protection mechanisms. No static or dynamic type checking are available, and programmers can rely on a very limited set of primitive types, namely integers, strings and alphanumeric constants, with no idiomatic way to enforce that the values of an argument must be of one of these types only — refer to [26] for details of the ASP-Core-2 format. More structured data types are usually represented by uninterpreted function symbols [83, 42, 19, 25], but again there is no idiomatic way to really validate such structures. Even *integrity constraints* may be insufficient to achieve a reliable data validation, as they are cheerfully satisfied if at least one of their literals is false; in fact, this is a very convenient property to discard unwanted solutions, but not very effective in guaranteeing data integrity — invalid data in this case may lead to discard some wanted solution among thousands or more equally acceptable solutions, a very hard to spot unexpected outcome.

The lack of data validation in ASP is likely due to the quest for better and better performance. After a significant effort to optimize the search algorithms that are one of the main reasons of the success of ASP systems, to sacrifice a few machine instructions just to validate data that are almost always valid sounds like a blasphemy. Someone may argument that ASP is not intended to be a general purpose programming language, and therefore input and output are eventually validated by external tools. However, this is a very simplistic assumption, and invalid data may appear in other portions of the program, causing the lost of otherwise acceptable solutions. Everyone is free to follow their own path, but at some point in their life, perhaps after spending hours looking for a typo in the name of a function, any programmer will regret not having had an idiomatic way to specify the format of their data. Quoting the Latins, *errare humanum est, perseverare autem diabolicum* — to err is human, but to persist (in error) is diabolical.

This thesis aims at rescuing ASP programmers from some problems due to data validation by proposing a framework called VALASP, written in Python and available at `https://github.com/alviano/valasp`. The proposed approach is to specify the format of some data of interest, leaving open the possibility to work with unspecified data types (Section 8). Moreover, the specification can be separated from the ASP program, and the fail-fast approach is achieved by injecting constraints that are guaranteed to be immediately satisfied when grounded, unless data are found to be invalid and some exception is raised. Such a behavior is obtained thanks to *interpreted functions*, briefly recalled in Section 7, where their use for data validation is also hinted. The specification itself can

be given in terms of annotations for Python classes, in the style of the recent Python dataclasses (`https://docs.python.org/3/library/dataclasses.html`); in this case, exceptions for invalid data can be raised manually, or by taking advantage of any off-the-shelf validation library (Section 8.1). An additional level of abstraction is given by a YAML-based format, which can be conveniently used for many common use cases, but also leaves the possibility to write arbitrary checks in Python methods that are called before or after the grounding procedure (Section 8.2). A few use cases are discussed in Section 9, and related work from the literature is discussed in Section 10 — in particular, differences with sort typed systems like IDP [27] and SPARC [85].

# Chapter 7

# Background

ASP programs are usually evaluated by a two-steps procedure: first, object variables are eliminated by means of *intelligent grounding* techniques, and after that stable models of the resulting propositional program are searched by means of sophisticated non-chronological backtracking algorithms. Details of this procedure, as well as on the syntax and semantics of ASP, are out of the scope of this work. Therefore, this section only recall the minimal background required to introduce the concepts presented in the next sections.

ASP is not particularly rich in terms of primitive types, and essentially allows for using integers and (double-quoted) strings. (We will use the syntax of CLINGO; [48].) More complex types, as for example dates or decimal numbers, can be represented by means of (non-interpreted) functions, or by the so called *@-terms*; in the latter case, the @-term must be associated with a Python function mapping different objects to different symbols in a repeatable way — for example, by populating a table of symbols or by using a natural enumeration.

**Example 7.0.1** (Primitive types and @-terms)**.** Dates can be represented by strings, functions (tuples as a special case) or @-terms, among other possibilities. Hence, `"1983/09/12"`, `(1983,9,12)` and `@date(1983,9,12)` can all represent the date 12th September 1983, where the @-term is associated with the following Python function:

```
def date(year, month, day):
    res = datetime.datetime(year.number, month.number, day.number)
    return int(res.timestamp())
```

Each representation comes with its pros and cons, discussed later in Section 8. ■

Intelligent grounding may process rules in several order, and literals within a rule can also be processed according to different ordering. A safe assumption made in this thesis is that all object variables of an @-term must be already bound to some ground term before the grounder can call the associated Python function.

**Example 7.0.2** (@-term invocation in the grounding process)**.** Consider the following program:

```
birthday(sofia, (2019,6,25)).
birthday(bigel, (1982,123)).  % Oops! I missed a comma, but where?!?
:- birthday(Person,Date), @is_triple_of_integer(Date) != 1.
```

The Python function associated with the @-term is called two times (at most, unless the system fails), with arguments `(2019,6,25)` and `(1982,123)`, so that some invariant can be enforced on the second argument of every instance of `birthday/2`. ■

Data validation is the process of ensuring that data conform to some specification, so that any process in which they are involved can safely work under all of the assumptions guaranteed by the specification. Data can be found invalid because of an expected error-prone source (for example, user input from a terminal), or due to an unexpected misuse of some functionality of a system (this is usually the case with bugs). While in the first case it is natural to ask again for the data, in the second case failing fast may be the only reasonable choice, so that the problem can be noticed, traced, and eventually fixed. The fail-fast approach is particularly helpful at coding time, to avoid bug hunting at a later time, but it may also pay off after deploy if properly coupled with a recovery mechanism (for example, restart the process).

**Example 7.0.3** (Data validation)**.** The @-term from Example 7.0.2 can be associated with the following Python code:

```
def is_triple_of_integer(value):
    if value.type != Function:
        raise ValueError('wrong type')
    if value.name:
        raise ValueError('not a tuple')
    if len(value.arguments) != 3:
        raise ValueError('not a triple')
    if any(arg for arg in value.arguments if arg.type != Number):
        raise ValueError('not a tuple of integers')
    return 1
```

Indeed, the presence of `birthday(bigel, (1982,123))` will be noticed because of abrupt termination of the grounding procedure. Adopting a fail-fast approach is the correct choice in this case, and any attempt of sanification is just a dangerous speculation on the invalid data — should it be `(1982,1,23)`, or `(1982,12,3)`? ■

# Chapter 8

# A data validation framework for ASP

Data validation can be used in ASP programs thanks to @-terms. However, the resulting code is likely to be less readable due to aspects that are not really the focus of the problem aimed to be addressed. We will illustrate our proposal to accomplish data validation without cluttering your ASP encoding in this section, but let us first stress why we believe that data validation may save the day of ASP programmers.

In Example 7.0.1, all three representations of dates can be used with built-in comparators, but within this respect the string representation is very fragile: `"1983/09/12" == "1983/9/12"` is (mistakenly) false because even if both terms represent the same date, they use different forms; `"1983/9/12" > "1983/10/12"` is (mistakenly) true because again one of the two dates is not in canonical form — if you opt for this representation, you better validate those strings. The tuple representation preserves more structure, which is very convenient if elements of the date have to be extracted, as in this case object variables or constants can be used as usual — if you opt for this representation, you better validate those tuples to check the number of arguments and their values. Extracting the year from the @-term representation instead requires further @-terms:

```
def year(timestamp): return
    datetime.date.fromtimestamp(timestamp.number).year
```

Hence, `@year(@date(1983,9,12)) == 1983` is (correctly) true. On the other hand, @-terms are likely the only easy way for computing the difference (in days) between two dates:

```
def days_between(a, b):
    a = datetime.date.fromtimestamp(a.number)
    b = datetime.date.fromtimestamp(b.number)
```

```
    return (b - a).days
```

For example, `@days_between(@date(2019,6,25), @date(2018,2,1))` returns `509`. An important point here, which may not be noticed at first, is that (at least in this case) the additional effort required by the @-term representation gives also some implicit data validation. In fact, while the use of strings and tuples do not prevent the creation of an invalid date like 29th February 2019, an exception is raised by `@date(2019,2,29)` to inhibit the uncontrolled release of an invalid object.

The proposed framework is made of several layers, and use Python at its core to inject constraints like the one in Example 7.0.3 to implement validation concerns (Section 8.1). As a level of abstract, the framework can automatically produce the Python code according to a YAML-based format designed to express validation criteria that are common for ASP programs (Section 8.2).

## 8.1 The Python layer

The framework relies on two main concepts, namely the *context* object and the *valasp* decorator. The context object has methods to register classes and @-terms, and to blacklist unwanted predicates; it is designed to be coupled with an ASP system, but also provides methods to simplify the binding at coding time. The decorator can be applied to a Python class to be interpreted as a specification of valid data; it is designed to inject in the context object all required code to validate ASP elements. The framework reserves the prefix `valasp` for internal usage at all levels.

**Context object.** Registered classes can be used by the code compiled by the context object, while the registration of an @-term requires to specify the signature of the function and its code as a list of strings. Generalizing the idea hinted in Example 7.0.2, we distinguish three types of *constraint validators*, referred to as *forward*, *implicit*, and *tuple* constraint validators, depending on how terms are grouped and passed to @-terms: implicit constraint validators use functions (with the same name of the validated predicate); tuple constraint validators use tuples (or unnamed functions); forward constraint validators are applicable only to unary predicates and use their unique terms. For a predicate `pred` of arity $n \geq 1$, the constraint validators are

```
:- pred(X1), @valasp_validate_pred(X1) != 1.       % FORWARD n = 1
:- pred(X1,...,Xn),
   @valasp_validate_pred(pred(X1,...,Xn)) != 1.    % IMPLICIT
:- pred(X1,...,Xn),
   @valasp_validate_pred((X1,...,Xn,)) != 1.       % TUPLE
```

to be associated with the Python function

```
def valasp_validate_pred(value):
    Pred(value)
    return 1
```

where `Pred` is a class whose name is obtained by capitalizing the first lowercase letter of `pred`, and whose constructor raises an exception if the provided data are invalid. Such a class have to be previously registered in the context (usually with the help of the decorator). Blacklisting a predicate name `pred` with a given arity $n \geq 1$ amounts to inject the constraint

```
:- pred(X1,...,Xn),
   @valasp_error("pred/n is blacklisted", (X1,...,Xn)) != 1.
```

where the @-term rises an exception with the provided message. In the following, a context object is assumed to be stored in the Python variable `ctx`.

**Decorator.** The format of predicates, values, functions, and tuples can be specified by applying the following decorator to Python classes:

```
@ctx.valasp(validate_predicate, with_fun, auto_blacklist)
```

Parameter `validate_predicate`, true by default, can be set to false to inhibit the creation of validator constraints, which is useful to decompose and reuse validation rules (see Example 8.1.1). The type of validator constraint is determined by parameter `with_fun`, and by default the framework uses forward validator constraints for unary predicates, and implicit validator constraints otherwise. Predicates with the same name but different arities are automatically blacklisted, but such a default mechanism can be disabled by setting parameter `auto_blacklist` to false. (Note that the framework does not forbid to use multiple arities for the same predicate name, but it does not allow to specify validation rules for two predicates with the same name but different arities.)

The decorator can be applied to a class with annotated fields, that is, a class specifying a list of pairs of the form `field: Type` — primitive types are named `Integer`, `String` and `Alpha`. Such annotations are used to enrich the decorated class with a constructor that assigns and validates the type of all fields from a symbol provided by the grounder (via the validator constraint); if the class has a `__post_init__` method, it is called before leaving the constructor. Other common *magic methods* are also added (if not already defined in the class) to provide a string representation and comparison operators.

**Example 8.1.1** (That's not a birthday!)**.** A predicate `bday` whose arguments represent a person and a date can be validated by

```
@ctx.valasp(validate_predicate=False, with_fun=Fun.TUPLE)
class Date:
    year: Integer
    month: Integer
```

```
    day: Integer

    def __post_init__(self):
        datetime.datetime(self.year, self.month, self.day)

@ctx.valasp()
class Bday:
    name: Alpha
    date: Date
```

Given the above specification, and the program

```
bday(sofia, (2019,6,25)).
bday(leonardo, (2018,2,1)).
bday(bigel, (1982,123)).
```

the framework raises an exception because of the third fact. ■

The framework automatically detects overflows for `Integer` instances, and can perform bulk calls to all methods with a given prefix for all registered classes in the context. It is a convenient way to initialize class level variables before starting the grounding procedure, and to check their values after the input program is grounded — a common use case is overflow detection on sums.

**Example 8.1.2** (Overflow detection on sum of integers)**.** Let `income` be a predicate representing incomes of companies, which are summed up in an ASP program comprising the facts `income("Acme ASP",1500000000)` and `income("Yoyodyne YAML",1500000000)`. Better to validate the sum than to go ahead with a meaningless `-1294967296`.

```
@ctx.valasp()
class Income:
    company: String
    amount: Integer

    def __post_init__(self):
        if not(self.amount > 0):
            raise ValueError("amount must be positive")
        self.__class__.amount_sum += self.amount

    @classmethod
    def before_grounding_init_amount_sum(cls):
        cls.amount_sum = 0

    @classmethod
    def after_grounding_init_amount_sum(cls):
        if cls.amount_sum > Integer.max():
```

```
            raise OverflowError()
```

Note that the class variable `amount_sum` is initialized in a `before_grounding*` class method, and updated after every new instance of `Income` (here we are taking advantage by the fact that Python implements arbitrary precision integers). Eventually, an `after_grounding*` class method detects the overflow — note that the overflow could be detected earlier in `__post_init__`. ■

## 8.2 The YAML layer

YAML is a human friendly data serialization standard, whose syntax is well-suited for hiding many details of the Python layer. The YAML files processed by our framework are essentially dictionaries of blocks defining types. The key of such a block is the name of a *user-defined type*, and the value is a dictionary of *field declarations*.

A field declaration must specify the `type` (either primitive or user-defined), and may specify one or more *facets*. The facets of `Integer` are `enum` to specify a list of acceptable values, `min` (by default $-2^{31}$) and `max` (by default $2^{31}-1$) to specify (inclusive) bounds, and finally `count`, `sum+` and `sum-` to specify bounds on the number of values, the sum of positive values and negative values. The facets of `String` and `Alpha` are `enum` and `count` as before, `min` and `max` to bound the length, and `pattern` to specify a regular expression. For a user-defined type, and for the wildcard `Any`, the only facets is `count`. When no facets are needed, it is sufficient to specify the type immediately after the name of the field.

User-defined types may also use the reserved `valasp` key to specify parameters for the decorator, and Python code to be executed (i) at the end of the `__post_init__` method (`after_init` key), and (ii) before and after the grounding process (`before_grounding` and `after_grounding` keys). Python imports and any additional global code can be specified at the root level using the reserved `valasp` key.

**Example 8.2.1** (Yet another time, not a birthday!)**.** Below is a YAML file to validate predicate `bday` from Example 8.1.1.

```
valasp:
    python: |+
        import datetime
    asp: |+
        #include "sofia_leonardo_bigel_bdays.asp".

date:
    year: Integer
    month: Integer
    day: Integer
```

```
    valasp:
        validate_predicate: False
        with_fun: TUPLE
        after_init: |+
            datetime.datetime(self.year, self.month, self.day)

bday:
    name: Alpha
    date: date
```

Note that Python and ASP code is specified with *literal style*, by using the `|+` marker, to preserve extra block indentations (not really needed for a single line, but important for multiple lines). In this case, if the included file contains `bday(bigel, (1982,123))`, an exception is raised. ■

**Example 8.2.2** (Overflow detection with the YAML format)**.** Below is a YAML representation of Example 8.1.2.

```
income:
    company: String
    amount:
        type: Integer
        min: 0
        sum+: Integer
```

Here, `sum+: Integer` is syntactic sugar for specifying that the sum of positive values must fit into a 32-bits integer — nicer than writing `max: 2147483647` inside `sum+`. ■

The `valasp` section of a symbol declaration can also include an `having` list of triples, where the first and third elements are field names, and the second element is an operator among `==`, `!=`, `<`, `<=`, `>=`, and `>`. The value associated with any of these keys must be a list of pairs of field names. All these comparison will be part of the `__post_init__` method. For example,

```
ordered_triple:
    first: Integer
    second: Integer
    third: Integer

    valasp:
        having:
            - first < second
            - second < third
```

specifies that `first < second < third` must hold. Note that YAML lists can be written as multiple lines starting with a dash, or in square brackets.

# Chapter 9

# Use cases and assessment

This section reports a few use cases on three encodings from ASP competitions. Each use case focuses on the validation of parts of an encoding, showing how the proposed framework can identify invalid data. (Tuning of the encoding is out of the scope of this work.) Moreover, the overhead introduced by data validation is empirically assessed.

## 9.1 Video streaming — 7th ASP competition (Gebser et al. 2020)

Video streaming amounts to selecting an optimal set of video representations, in terms of resolution and bitrate, to satisfy user requirements. User requirements and solution are respectively encoded by `user(USERID, VIDEOTYPE, RESOLUTION, BANDWIDTH, MAXSAT, MAXBITRATE)` and `assign(USER_ID, VIDEO_TYPE, RESOLUTION, BITRATE, SAT)`. The overall satisfaction of users is maximized by the following weak constraint:

```
:∼ assign(USER_ID,_,_,BITRATE,SAT_VALUE), user(USER_ID,_,_,_,BEST_SAT,_).
   [BEST_SAT-SAT_VALUE@1, USER_ID, assign]
```

According to the official description, available online at `http://aspcomp2015.dibris.unige.it/Video_Streaming.pdf`, instances of `user/6` can be validated with the following YAML specification:

```
user:
    userid:
        type: Integer
        min: 0
    videotype:
        type: String
        enum: [Documentary, Video, Cartoon, Sport]
    resolution:
```

```
        type: Integer
        enum: [224, 360, 720, 1080]
    bandwidth:
        type: Integer
        min: 0
    maxsat:
        type: Integer
        min: 0
    maxbitrate:
        type: Integer
        min: 150
        max: 8650
    valasp:
        after_init: |+
           if self.value % 50 != 0:
               raise ValueError("unexpected bitrate")
```

According to the above specification, the arguments `userid`, `bandwidth` and `maxsat` are non-negative integers; `videotype` is a string among `Documentary`, `Video`, `Cartoon`, and `Sport`; argument `resolution` is an integer among 224, 360, 720, and 1080; and `maxbitrate` is an integer between 150 and 8650, and it is divisible by 50.

The official encoding and instances do not have errors, as expected. However, the encoding is quite fragile and relies on several assumptions on the input data and on ASP internals — ASP systems use 32-bits integers for everything but the cost of a solution. To show how dangerous are such assumptions, consider a decision problem where a partial solution and a target satisfaction are given. Accordingly, the weak constraint is replaced by the following constraint:

```
:- target(T), #sum{BEST_SAT-SAT_VALUE, USER_ID :
    assign(USER_ID,_,_,BITRATE,SAT_VALUE), user(USER_ID,_,_,_,BEST_SAT,_)}
    > T.
```

In this case, the execution of CLINGO on the instances of the competition may lead to the error message `"Value too large to be stored in data type: Integer overflow!"`, produced while simplifying the sum. However, whether the message is shown or not depends on the partial solution provided in input. In fact, if the overflow is only due to the `assign/5` instances in input, the subsequent simplification step cannot notice the problem and a wrong answer is produced. The following YAML specification can help to detect these overflows:

```
target:
    value:
      type: Integer
      min: 0
```

```
sum_element:
    value:
        type: Integer
        min: 0
        sum+: Integer
   userid: Integer
valasp:
    asp: |+
        sum_element(BEST_SAT-SAT_VALUE,UID) :-
            assign(UID,_,_,BITRATE,SAT_VALUE), user(UID,_,_,_,BEST_SAT,_).
```

## 9.2 Solitaire — 4th ASP Competition (Alviano et al. 2013)

Solitaire represents a single-player game played on a 7x7 board where the 2x2 corners are omitted. We focus on the following rules defining the board:

```
range(1).
range(X+1) :- range(X), X < 7.
location(1,X) :- range(X), 3 <= X, X <= 5.
location(2,X) :- range(X), 3 <= X, X <= 5.
location(Y,X) :- range(Y), 3 <= Y, Y <= 5, range(X).
location(6,X) :- range(X), 3 <= X, X <= 5.
location(7,X) :- range(X), 3 <= X, X <= 5.
```

Those rules are interesting since an error in this point might be propagated all over the encoding. The YAML specification of `range` and `location` is the following:

```
range:
    value:
        type: Integer
        enum: [1, 2, 3, 4, 5, 6, 7]
location:
    x: range
    y: range
    valasp:
        after_grounding: |+
            pos = [1,2,6,7]
            if self.x.value in pos and self.y.value in pos:
                raise ValueError("Invalid position")
```

## 9.3 Qualitative spatial reasoning — 4th ASP Competition (Alviano et al. 2013)

Qualitative spatial reasoning consists of deciding whether a set of spatial and temporal constraints is consistent with respect to a composition table. Membership in qualitative relations is encoded by 169 rules, similar to the following:

```
label(X,Z,rp) :- label(X,Y,rp), label(Y,Z,rp).
label(X,Z,req) | label(X,Z,rp) | label(X,Z,rpi) | label(X,Z,rd) |
    label(X,Z,rdi)
    | label(X,Z,rs) | label(X,Z,rsi) | label(X,Z,rf) | label(X,Z,rfi)
    | label(X,Z,rm) | label(X,Z,rmi) | label(X,Z,ro) | label(X,Z,roi)
    :- label(X,Y,rp), label(Y,Z,rpi).
```

The third argument of `label/3` is a qualitative relation. The following YAML specification can be used to validate such rules:

```
rel:
    value:
        type: Alpha
        enum: [req, rp, rpi, rd, rdi, ro, roi, rm, rmi, rs, rsi, rf, rfi]
node:
    value:
        type: Integer
        min: 0
        max: 49
    valasp:
        validate_predicate: False
label:
    x: node
    y: node
    l: rel
    valasp:
        having: [x < y]
```

## 9.4 Empirical assessment

The overhead introduced by VALASP to validate instances of the discussed problems was measured by running CLINGO with and without validation. The experiment was run on a 2.4 GHz Quad-Core Intel Core i5 with 16 GB of memory. VALASP was executed with the command-line option `--valid-only`, and CLINGO was executed with the command-line option `--mode=gringo` (and redirecting output to `/dev/null`); both options disable the

computation of stable models since VALASP has no impact on the solving procedure. We remark here that the running time of VALASP includes grounding time. For each benchmark, we considered all available instances.

Concerning video streaming, the average running time of CLINGO is 0.08 seconds, and the average running time of VALASP is 0.18 seconds. As for solitaire, the average running time of CLINGO and VALASP is respectively 0.10 and 0.13 seconds. Finally, on qualitative spatial reasoning, the average running time of CLINGO and VALASP is respectively 4.13 and 3.42 seconds; in this case VALASP is even faster than CLINGO because VALASP does not print the ground program. We can conclude that no overhead is eventually introduced by VALASP on these testcases.

# Chapter 10

# Related work

The use of types in programming languages eases the representation of complex knowledge, favors the early detection of errors and provides an implicit documentation of source codes [101]. For example, by stating that the arguments of predicate `bday` are of types `person_name` and `date`, there is no need to document the way these elements are represented, and any attempt to instantiate this predicate with different types is blocked. ASP-Core-2 [26], on the other hand, is untyped: there is no way to state that arguments of a predicate must be of a specific type, the language offers a very limited set of primitive types, and there is no idiomatic way to declare user-defined types. This work targets ASP-Core-2, and its extensions implemented by CLINGO [48] and DLV2 [1], aiming at providing the missing idioms to specify types and to validate data.

Types are not new in logic-based languages, and in particular order-sorted logic has been formalized as first-order logic with sorted terms, where sorts are ordered to build a hierarchy [67]. IDP3 [23, 27] and SPARC [15, 107, 85] are two systems with languages close to ASP-Core-2 and supporting sorted terms. There are many differences between these systems and the framework proposed in this work. First of all, VALASP is designed to be smoothly integrated with ASP-Core-2 projects: the programmer is free to choose what to validate and what to leave unchecked, and the original encoding can still be used as it is in case validation is not required in the deployed software. Sorted terms are also used to bound object variables in rules, while this is not possible with VALASP because it only deals with the aspect of data validation.

The framework uses @-terms to perform data validation by means of Python functions that are called during the grounding process. In the literature, @-terms and non-Herbrand functions [16] were used to enrich ASP with functionality that are otherwise not viable (if not in the Turing tarpit). External atoms in HEX [41] extend the notion of externally interpreted function to externally interpreted relations, and can be also used to achieve some form of data validation [104]. Hence, external atoms can be used as an alternative to

@-terms for implementing the validation constraints defined in Section 8.1.

Finally, there are works in the literature that introduce data validation in Prolog systems [72] and that implement data validation for Constraint Logic Programming by means of Prolog systems [58, 102]. The goal of those works is clearly related to this part of the thesis, but they differ on the way data validation is specified, on the target language and on the underlying implementation. Similarly, debugging techniques for ASP [43, 50, 97, 37] share the goal to identify errors, but with a different approach. VALASP aims at blocking data validation errors in a very early stage, at coding time and by implementing fail-fast techniques to point to the source of the problem. Debugging techniques instead are useful to localize the origin of unattended behavior, and usually requires interaction with the programmer. If VALASP is properly used, a debugger is still a useful software in the tool belt of an ASP programmer, but on the other hand it is likely that the number of debugging sessions will be reduced.

# Chapter 11

# Conclusion

ASP programmers do mistakes, there is no shame in this. VALASP aims at early detection of data validity errors, and promotes a fail-fast approach so that the origin of the problem can be quickly identified and fixed. The proposed approach follows the separation of concerns design principle: validation rules are specified in Python or YAML, and are separated from the business logic represented in ASP encodings. Such a design is useful to smoothly introduce data validation in ASP, as validation rules can be specified externally without the need to deeply change the way programs are written. If after deploy data can be safely assumed valid, VALASP can be easily discharged because the original encoding stays unchanged. The framework is open source, and can be used to validate data for both CLINGO and DLV2.

# Bibliography


[1] Weronika T. Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. The ASP system DLV: advancements and applications. *KI*, 32(2-3):177–179, 2018.

[2] Charu C Aggarwal. Deep reinforcement learning. In *Neural Networks and Deep Learning*, pages 373–417. Springer, 2018.

[3] Chris Aldrich and Jacques Olivier. Nonlinear time series analysis of avalanching granular flow data. *IFAC-PapersOnLine*, 52(14):237–242, 2019.

[4] Earl Ryan M Aleluya, Arnel D Zamayla, and Shayne Lyle M Tamula. Decision-making system of soccer-playing robots using finite state machine based on skill hierarchy and path planning through bezier polynomials. *Procedia Computer Science*, 135:230–237, 2018.

[5] Mario Alviano, Francesco Calimeri, Günther Charwat, Minh Dao-Tran, Carmine Dodaro, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Johannes Oetsch, Andreas Pfandler, Jörg Pührer, Christoph Redl, Francesco Ricca, Patrik Schneider, Martin Schwengerer, Lara Katharina Spendier, Johannes Peter Wallner, and Guohui Xiao. The fourth answer set programming competition: Preliminary report. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2013.

[6] Mario Alviano, Carmine Dodaro, and Arnel Zamayla. Data validation meets answer set programming. In José F. Morales and Dominic A. Orchard, editors, *Practical Aspects of Declarative Languages - 23rd International Symposium, PADL 2021, Copenhagen, Denmark, January 18-19, 2021, Proceedings*, volume 12548 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 2021.

[7] Mario Alviano, Carmine Dodaro, and Arnel Zamayla. Valasp: a tool for data validation in answer set programming. *Theory Pract. Log. Program.*, 2022.

[8] Mario Alviano and Arnel Zamayla. A speech about generative datalog and non-measurable sets. In Joaquín Arias, Fabio Aurelio D'Asaro, Abeer Dyoub, Gopal Gupta, Markus Hecher, Emily LeBlanc, Rafael Peñaloza, Elmer Salazar, Ari Saptawijaya, Felix Weitkämper, and Jessica Zangari, editors, *Proceedings of the International Conference on Logic Programming 2021 Workshops co-located with the 37th International Conference on Logic Programming (ICLP 2021), Porto, Portugal (virtual), September 20th-21st, 2021*, volume 2970 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

[9] https://www.mturk.com/ Amazon Mechanical Turk.

[10] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[11] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[12] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[13] Jieun Baek and Yosoon Choi. Deep neural network for ore production and crusher utilization prediction of truck haulage system in underground mine. *Applied Sciences*, 9(19):4180, 2019.

[14] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5221–5229, 2017.

[15] Evgenii Balai, Michael Gelfond, and Yuanlin Zhang. Towards answer set programming with sorts. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 2013.

[16] Marcello Balduccini. ASP with non-herbrand partial functions: a language and system for practical use. *Theory Pract. Log. Program.*, 13(4-5):547–561, 2013.

[17] Jason P Bardinas, Chris Aldrich, and Lara FA Napier. Predicting the operating states of grinding circuits by use of recurrence texture analysis of time series data. *Processes*, 6(2):17, 2018.

[18] Gianni Bartolacci, Patrick Pelletier Jr, Jayson Tessier Jr, Carl Duchesne, Pierre-Alexandre Bossé, and Julie Fournier. Application of numerical image analysis to process diagnosis and physical parameter measurement in mineral processes—part i: flotation control based on froth textural characteristics. *Minerals Engineering*, 19(6-8):734–747, 2006.

[19] Sabrina Baselice and Piero A. Bonatti. A decidable subclass of finitary programs. *Theory Pract. Log. Program.*, 10(4-6):481–496, 2010.

[20] Alex Bewley and Ben Upcroft. From imagenet to mining: Adapting visual object detection with minimal supervision. In *Field and Service Robotics*, pages 501–514. Springer, 2016.

[21] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[22] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[23] Maurice Bruynooghe, Hendrik Blockeel, Bart Bogaerts, Broes De Cat, Stef De Pooter, Joachim Jansen, Anthony Labarre, Jan Ramon, Marc Denecker, and Sicco Verwer. Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with *IDP3*. *Theory Pract. Log. Program.*, 15(6):783–817, 2015.

[24] Peter J Burt and Edward H Adelson. The laplacian pyramid as a compact image code. In *Readings in computer vision*, pages 671–679. Elsevier, 1987.

[25] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Finitely recursive programs: Decidability and bottom-up computation. *AI Commun.*, 24(4):311–334, 2011.

[26] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309, 2020.

[27] Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Predicate logic as a modeling language: the IDP system. In Michael Kifer

and Yanhong Annie Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, pages 279–323. ACM / Morgan & Claypool, 2018.

[28] Abhishek Chaurasia and Eugenio Culurciello. Linknet: Exploiting encoder representations for efficient semantic segmentation. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2017.

[29] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

[30] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2180–2188, 2016.

[31] Lefeng Cheng and Tao Yu. A new generation of ai: A review and perspective on machine learning technologies applied to smart energy and electric power systems. *International Journal of Energy Research*, 43(6):1928–1973, 2019.

[32] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Convolutional nets and watershed cuts for real-time semantic labeling of rgbd videos. *The Journal of Machine Learning Research*, 15(1):3489–3511, 2014.

[33] Siddharth Dadhich, Ulf Bodin, and Ulf Andersson. Key challenges in automation of earth-moving machines. *Automation in Construction*, 68:212–222, 2016.

[34] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[35] Ayushman Dash, John Cristian Borges Gamboa, Sheraz Ahmed, Marcus Liwicki, and Muhammad Zeshan Afzal. Tac-gan-text conditioned auxiliary classifier generative adversarial network. *arXiv preprint arXiv:1703.06412*, 2017.

[36] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

[37] Carmine Dodaro, Philip Gasteiger, Kristian Reale, Francesco Ricca, and Konstantin Schekotihin. Debugging non-ground ASP programs: Technique and graphical tools. *Theory Pract. Log. Program.*, 19(2):290–316, 2019.

[38] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[39] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

[40] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[41] Thomas Eiter, Stefano Germano, Giovambattista Ianni, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. The DLVHEX system. *KI*, 32(2-3):187–189, 2018.

[42] Thomas Eiter and Mantas Simkus. Bidirectional answer set programs with function symbols. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 765–771, 2009.

[43] Jorge Fandinno and Claudia Schulz. Answering the "why" in answer set programming - A survey of explanation approaches. *Theory Pract. Log. Program.*, 19(2):114–203, 2019.

[44] Y Fu and C Aldrich. Flotation froth image recognition with convolutional neural networks. *Minerals Engineering*, 132:183–190, 2019.

[45] Y Fu and Chris Aldrich. Using convolutional neural networks to develop state-of-the-art flotation froth image sensors. *IFAC-PapersOnLine*, 51(21):152–157, 2018.

[46] Yihao Fu and Chris Aldrich. Froth image analysis by use of transfer learning and convolutional neural networks. *Minerals Engineering*, 115:68–78, 2018.

[47] BP Gautham, Sreedhar Reddy, Venkataramana Runkana, et al. Future of mining, mineral processing and metal extraction industry. *Transactions of the Indian Institute of Metals*, 72(8):2159–2177, 2019.

[48] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Patrick Lühne, Philipp Obermeier, Max Ostrowski, Javier Romero, Torsten Schaub, Sebastian Schellhorn, and Philipp Wanko. The potsdam answer set solving collection 5.0. *KI*, 32(2-3):181–182, 2018.

[49] Martin Gebser, Marco Maratea, and Francesco Ricca. The seventh answer set programming competition: Design and results. *Theory Pract. Log. Program.*, 20(2):176–204, 2020.

[50] Martin Gebser, Jörg Pührer, Torsten Schaub, and Hans Tompits. A meta-programming technique for debugging answer-set programs. In Dieter Fox and

Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 448–453. AAAI Press, 2008.

[51] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[52] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[53] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[54] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pages 1462–1471. PMLR, 2015.

[55] Weihua Gui, Jinping Liu, Chunhua Yang, Ning Chen, and Xi Liao. Color co-occurrence matrix based froth image texture extraction for mineral flotation. *Minerals Engineering*, 46:60–67, 2013.

[56] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[58] Manuel V. Hermenegildo, Germán Puebla, Francisco Bueno, and Pedro López-García. Program debugging and validation using semantic approximations and partial specifications. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 69–72. Springer, 2002.

[59] Huichao Hong, Lixin Zheng, Jianqing Zhu, Shuwan Pan, and Kaiting Zhou. Automatic recognition of coal and gangue based on convolution neural network. *arXiv preprint arXiv:1712.00720*, 2017.

[60] ZC Horn, L Auret, JT McCoy, Chris Aldrich, and BM Herbst. Performance of convolutional neural networks for feature extraction in froth flotation sensing. *IFAC-PapersOnLine*, 50(2):13–18, 2017.

[61] Zeshan Hyder, Keng Siau, and Fiona Fui-Hoon Nah. Use of artificial intelligence, machine learning, and autonomous technologies in the mining industry. *MWAIS 2018 Proceedings*, 43, 2018.

[62] Julio César Álvarez Iglesias, Richard Bryan Magalhaes Santos, and Sidnei Paciornik. Deep learning discrimination of quartz and resin in optical microscopy images of minerals. *Minerals Engineering*, 138:79–85, 2019.

[63] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016.

[64] Adam Jacobson, Fan Zeng, David Smith, Nigel Boswell, Thierry Peynot, and Michael Milford. Semi-supervised slam: Leveraging low-cost sensors on underground autonomous vehicles for position tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3970–3977. IEEE, 2018.

[65] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[66] Dan Bergh Johnsson, Daniel Deogun, and Daniel Sawano. *Secure by Design*. Manning Publications, 2019.

[67] Ken Kaneiwa. Order-sorted logic programming with predicate hierarchy. *Artif. Intell.*, 158(2):155–188, 2004.

[68] Sadegh Karimpouli and Pejman Tahmasebi. Image-based velocity estimation of rock using convolutional neural networks. *Neural Networks*, 111:89–97, 2019.

[69] Sadegh Karimpouli and Pejman Tahmasebi. Segmentation of digital rock images using deep convolutional autoencoder networks. *Computers & geosciences*, 126:142–150, 2019.

[70] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[71] Saeed Reza Kheradpisheh, Masoud Ghodrati, Mohammad Ganjtabesh, and Timothée Masquelier. Deep networks can resemble human feed-forward vision in invariant object recognition. *Scientific reports*, 6(1):1–24, 2016.

[72] R Kiel and M Schader. A tool for validating prolog programs. In *Classification, Data Analysis, and Knowledge Organization*, pages 183–188. Springer, 1991.

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

[74] Melissa Kistner, Gorden T Jemwa, and Chris Aldrich. Monitoring of mineral processing systems by using textural image analysis. *Minerals Engineering*, 52:169–177, 2013.

[75] A. J. Ko and Brad A. Myers. Development and evaluation of a model of programming errors. In *2003 IEEE Symposium on Human Centric Computing Languages and Environments (HCC 2003), 28-31 October 2003, Auckland, New Zealand*, pages 7–14. IEEE Computer Society, 2003.

[76] A. J. Ko and Brad A. Myers. A framework and methodology for studying the causes of software errors in programming systems. *J. Vis. Lang. Comput.*, 16(1-2):41–84, 2005.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[78] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR, 2016.

[79] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[80] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[81] Zhong-mei Li, Wei-hua Gui, and Jian-yong Zhu. Fault detection in flotation processes based on deep learning and support vector machine. *Journal of Central South University*, 26(9):2504–2515, 2019.

[82] Zhengyu Liang, Zhihong Nie, Aijun An, Jian Gong, and Xiang Wang. A particle shape extraction and evaluation method using a deep convolutional neural network and digital image processing. *Powder Technology*, 353:156–170, 2019.

[83] Yuliya Lierler and Vladimir Lifschitz. One more decidable class of finitely ground programs. In Patricia M. Hill and David Scott Warren, editors, *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, volume 5649 of *Lecture Notes in Computer Science*, pages 489–493. Springer, 2009.

[84] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[85] Elias Marcopoulos and Yuanlin Zhang. OnlineSPARC: A programming environment for answer set programming. *Theory Pract. Log. Program.*, 19(2):262–289, 2019.

[86] Victor W. Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[87] John T McCoy, Steve Kroon, and Lidia Auret. Variational autoencoders for missing data imputation with application to a simulated milling circuit. *IFAC-PapersOnLine*, 51(21):141–146, 2018.

[88] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[89] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017.

[90] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[91] Roberto Natella, Stefan Winter, Domenico Cotroneo, and Neeraj Suri. Analyzing the effects of bugs on software interfaces. *IEEE Trans. Software Eng.*, 46(3):280–301, 2020.

[92] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4467–4477, 2017.

[93] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29:3387–3395, 2016.

[94] Ilkka Niemelä. Logic programming with stable model semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

[95] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 271–279, 2016.

[96] Augustus Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[97] Johannes Oetsch, Jörg Pührer, and Hans Tompits. Catching the ouroboros: On debugging non-ground answer-set programs. *Theory Pract. Log. Program.*, 10(4-6):513–529, 2010.

[98] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[99] Rohan Padhye and Koushik Sen. Efficient fail-fast dynamic subtype checking. In Daniele Bonetta and Yu David Liu, editors, *Proceedings of the 11th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages, VMIL@SPLASH 2019, Athens, Greece, October 22, 2019*, pages 32–37. ACM, 2019.

[100] Guim Perarnau, Joost Van De Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016.

[101] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

[102] Germán Puebla, Francisco Bueno, and Manuel V. Hermenegildo. A generic processor for program validation and debugging. In Pierre Deransart, Manuel V. Hermenegildo, and Jan Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming, Constrain Debugging (DiSCiPl project)*, volume 1870 of *Lecture Notes in Computer Science*, pages 63–107. Springer, 2000.

[103] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[104] Christoph Redl. Extending answer set programs with interpreted functions as first-class citizens. In Yuliya Lierler and Walid Taha, editors, *Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2017, Paris, France, January 16-17, 2017, Proceedings*, volume 10137 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 2017.

[105] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *International Conference on Machine Learning*, pages 1060–1069. PMLR, 2016.

[106] Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. *Advances in neural information processing systems*, 29:217–225, 2016.

[107] Christian Reotutar, Mbathio Diagne, Evgenii Balai, Edward Wertz, Peter Lee, Shao-Lon Yeh, and Yuanlin Zhang. An online logic programming development environment. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 4130–4131. AAAI Press, 2016.

[108] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[109] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[110] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[111] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242, 2016.

[112] K Sam Shanmugan, Venkatesh Narayanan, Victor S Frost, Josephine Abbot Stiles, and Julian C Holtzman. Textural features for radar image analysis. *IEEE Transactions on Geoscience and Remote Sensing*, GE-19(3):153–156, 1981.

[113] William J Shipman and Loutjie C Coetzee. Reinforcement learning and deep neural networks for pi controller tuning. *IFAC-PapersOnLine*, 52(14):111–116, 2019.

[114] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[115] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[116] Godfred Somua-Gyimah, Samuel Frimpong, Wedam Nyaaba, and Eric Gbadam. A computer vision system for terrain recognition and object detection tasks in mining and construction environments. In *SME Annual Conference*, 2019.

[117] JR Van Duijvenbode and MWN Buxton. Use of time series event classification to control ball mill performance in the comminution circuit–a conceptual framework. *Real Time Mining*, page 114, 2019.

[118] Vaughn Vernon. *Domain-Driven Design Distilled*. Addison-Wesley, 2016.

[119] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[120] Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *European conference on computer vision*, pages 318–335. Springer, 2016.

[121] Jie Wei, Lei Guo, Xinying Xu, and Gaowei Yan. Soft sensor modeling of mill level based on convolutional neural network. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 4738–4743. IEEE, 2015.

[122] Degang Xu, Xiao Chen, Yongfang Xie, Chunhua Yang, and Weihua Gui. Complex networks-based texture extraction and classification method for mineral flotation froth images. *Minerals Engineering*, 83:105–116, 2015.

[123] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[124] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. Lr-gan: Layered recursive generative adversarial networks for image generation. *arXiv preprint arXiv:1703.01560*, 2017.

[125] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5485–5493, 2017.

[126] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S Paek, and In So Kweon. Pixel-level domain transfer. In *European conference on computer vision*, pages 517–532. Springer, 2016.

[127] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[128] Jin Zhang, Zhaohui Tang, Jinping Liu, Zhen Tan, and Pengfei Xu. Recognition of flotation working conditions through froth image statistical modeling for performance monitoring. *Minerals Engineering*, 86:116–129, 2016.

[129] Jin Zhang, Zhaohui Tang, Yongfang Xie, Mingxi Ai, and Weihua Gui. Convolutional memory network-based flotation performance monitoring. *Minerals Engineering*, 151:106332, 2020.

[130] Wei Zhao, Wei Xu, Min Yang, Jianbo Ye, Zhou Zhao, Yabing Feng, and Yu Qiao. Dual learning for cross-domain image captioning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 29–38, 2017.

# Appendix A

# Publications

## A.1 Knowledge Representation

### Data Validation Meets Answer Set Programming

Presented at *PADL 2021: 23rd International Symposium on Practical Aspects of Declarative Languages, Copenhagen, Denmark* [6].

Data validation may save the day of computer programmers, whatever programming language they use. In fact, processing invalid data is a waste of resources at best, and a drama at worst if the problem remains unnoticed and wrong results are used for business. Answer Set Programming is not an exception, but the quest for better and better performance resulted in systems that essentially do not validate data in any way. Even under the simplistic assumption that input and output data are eventually validated by external tools, invalid data may appear in other portions of the program, and go undetected until some other module of the designed software suddenly breaks. This paper formalizes the problem of data validation for ASP programs, introduces a declarative language to specify data validation, and presents a tool to inject data validation in ordinary programs. The proposed approach promotes fail-fast techniques at coding time without imposing any lag on the deployed system if data are pretended to be valid. Additionally, the proposed approach opens the possibility to take advantage of ASP declarativity for validating complex data of imperative programming languages.

### A speech about Generative Datalog and Non-measurable Sets

Presented at *ASPOCP 2021: 14th Workshop on Answer Set Programming and Other Computing Paradigms, Porto, Portugal* [8].

Generative Datalog is the first component of PPDL (short for Probabilistic-Programming Datalog), a recently proposed probabilistic programming language. Specifically, genera-

tive Datalog provides constructs to refer to parameterized probability distribution, and is used for the specification of stochastic processes. Possible outcomes of such a stochastic process are possibly filtered according to logical constraints, which constitute the second component of PPDL. This paper is about generative Datalog, and hints on the possibility to represent non-measurable sets by combining generative Datalog constructs with addition over real numbers and a single, atomic, ground constraint.

### ValAsp: a tool for data validation in Answer Set Programming

Accepted for publication in *Theory and Practice of Logic Programming* [7].

The development of complex software requires tools promoting fail-fast approaches, so that bugs and unex- pected behavior can be quickly identified and fixed. Tools for data validation may save the day of computer programmers. In fact, processing invalid data is a waste of resources at best, and a drama at worst if the problem remains unnoticed and wrong results are used for business. Answer Set Programming (ASP) is not an exception, but the quest for better and better performance resulted in systems that essentially do not validate data. Even under the simplistic assumption that input/output data are eventually validated by external tools, invalid data may appear in other portions of the program, and go undetected until some other module of the designed software suddenly breaks. This paper formalizes the problem of data validation for ASP programs, introduces a declarative language to specify data validation, and presents VALASP , a tool to inject data validation in ordinary programs. The proposed approach promotes fail-fast techniques at coding time without imposing any lag on the deployed system if data are pretended to be valid. Validation can be specified in terms of statements using YAML, ASP and Python. Additionally, the proposed approach opens the possibility to use ASP for validating data of imperative programming languages.

## A.2 Neural Networks

Over the last decade, autonomous haulage vehicles used in mining operations have been developed by several manufacturers [47]. Fortunately, vehicles that move inside the mining area operate in a controlled environment compared to private vehicles in public highways. In effect, the operation of self-driving haulage trucks, loaders, dozers and excavators [61] has been well established over the last few years. However, there is a strong need for a robust localization and navigation, vision-based sensing, proximity detection and collision avoidance as highlighted by [33] who also reported the potential use of reinforcement learning for intelligent automatic control. A work done by [4] about robotic systems can be tweaked to fit the requirements needed to design autonomous vehicles in the mining area.

Particularly in the path planning and decision-making mechanism. Although the accuracy rate is not that high from the reported test bed. This is because soccer is a complicated game when compared to a network of vehicles travelling to accomplish a much simpler task of hauling from point A to point B in the mining area. The researchers recognize the latency introduced with the use of Bluetooth suffering from lost or delayed data transmission and the poor computing power with the meager budget when the research was performed. The researcher would like to explore the feasibility of implementing an even more advanced algorithm with the advent of deep learning. A major bottleneck would be the system described was dependent on a camera setup on a birds eye view for vision. However, in the Pilbara region in Western Australia, large fleets of autonomous haulage vehicles are steadily replacing manually driven vehicles. Likewise, the use of unmanned arial vehicles (UAV) or drones have also become a routine, alleviating this problem. But in underground environment this will not do so even the most-popular vision-based localization method SLAM [90] as reported by [64] due to the dust and varying light conditions introducing high to low contrast condition which affects color perception and accuracy. A major issue for the research works done mentioned in this section is that human presence is still required in many applications hence not being fully autonomous [20]. The CNN based [20] system, detects light vehicles and personnel in an active open-pit mine site which reduced the false positive rate. Imagine inanimate objects triggering false alarms for autonomous trucks leading them to stop which ultimately trigger high risk and expensive sever collisions. [116] furthered the purpose of collision avoidance by proposing a similar network architecture for object recognition and terrain recognition in an open pit mining environment.