UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

GENERATIVE ADVERSARIAL NETWORK BASED MODEL FOR MULTI-DOMAIN FAULT DIAGNOSIS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

JUAN PABLO CABEZAS RODRÍGUEZ

PROFESOR GUÍA:
ENRIQUE LÓPEZ DROGUETT

MIEMBROS DE LA COMISIÓN:
VIVIANA MERUANE NARANJO
EDUARDO SALAMANCA HENRÍQUEZ

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: JUAN PABLO CABEZAS RODRÍGUEZ
FECHA: JUNIO 2019
PROF. GUÍA: ENRIQUE LÓPEZ DROGUETT

GENERATIVE ADVERSARIAL NETWORK BASED MODEL FOR MULTI-DOMAIN FAULT DIAGNOSIS

Con el uso de las redes neuronal profundas ganando terreno en el área de PHM, los sensores disminuyendo progresivamente su precio y mejores algoritmos, la falta de datos se ha vuelto un problema principal para los modelos enfocados en datos. Los datos etiquetados y aplicables a escenarios específicos son, en el mejor de los casos, escasos.
El objetivo de este trabajo es desarrollar un método para diagnosticas el estado de un rodamiento en situaciones con datos limitados.
Hoy en día la mayoría de las técnicas se enfocan en mejorar la precisión del diagnóstico y en estimar la vida útil remanente en componentes bien documentados. En el presente, los métodos actuales son ineficiente en escenarios con datos limitados.
Se desarrolló un método en el cual las señales vibratorias son usadas para crear escalogramas y espectrogramas, los cuales a su vez se usan para entrenar redes neuronales generativas y de clasificación, en función de diagnosticar un set de datos parcial o totalmente desconocido, en base a uno conocido.
Los resultados se comparan con un método más sencillo en el cual la red para clasificación es entrenada con el set de datos conocidos y usada directamente para diagnosticar el set de datos desconocido.
El Case Western Reserve University Bearing Dataset y el Machine Failure Prevention Technology Bearing Dataset fueron usados como datos de entrada. Ambos sets se usaron como conocidos tanto como desconocidos.
Para la clasificación una red neuronal convolucional (CNN por sus siglas en inglés) fue diseñada. Una red adversaria generativa (GAN por sus siglas en inglés) fue usada como red generativa. Esta red fue basada en una introducida en el paper *StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation.*
Los resultados fueron favorables para la red CNN mientras que fueron -en general- desfavorables para la red GAN.
El análisis de resultados sugiere que la función de costo es inapropiada para el problema propuesto.
Las conclusiones dictaminan que la traducción *imagen-a-imagen* basada en la función ciclo no funciona correctamente en señal vibratorias para diagnóstico de rodamientos.

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: JUAN PABLO CABEZAS RODRÍGUEZ
FECHA: JUNIO 2019
PROF. GUÍA: ENRIQUE LÓPEZ DROGUETT

GENERATIVE ADVERSARIAL NETWORK BASED MODEL FOR MULTI-DOMAIN FAULT DIAGNOSIS

With the use of deep neural networks gaining notoriety on the prognostics & health management field, sensors getting progressively cheaper and improved algorithms, the lack of data has become a major issue for data-driven models. Data which is labelled and applicable for specific scenarios is scarce at best.
The purpose of this works is to develop a method to diagnose the health state of a bearing on limited data situations.
Now a days most techniques focus on improving accuracy for diagnosis and estimating remaining useful life on well documented components. As it stands, current methods are ineffective on limited data scenarios.
A method was developed were in vibration signals are used to create scalograms and spectrograms, which in turn are used to train generative and classification neural networks with the goal of diagnosing a partially or totally unknown dataset based on a fully labelled one.
Results were compared to a simpler method in which a classification network is trained on the labelled dataset to diagnose the unknown dataset.
As inputs the Case Western Reserve University Bearing Dataset (CWR) and the Society for Machine Failure Prevention Technology Bearing Dataset. Both datasets are used as labelled and unknown.
For classification a Convolutional Neural Network (CNN) is designed. A Generative Adversarial Network (GAN) is used as generative model. The generative model is based of a previous paper called *StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation.*
Results were favourable for the CNN network whilst generally negative for the GAN network. Result analysis suggests that the cost function is unsuitable for the proposed problem.
Conclusions state that cycle based image-to-image translation does not work correctly on vibration signals for bearing diagnosis.

One must imagine Sisyphus happy - Albert Camus

Agradecimientos

First and foremost I want to thank my parents and family, for they are the ones responsible of my unquenchable thirst for knowledge. I grew up in an environment where my curiosity was nurtured and my questions would not be left unsolved. I would not be the person I am happy to be if it were not for that as long as I can remember I've been encouraged to pursuit the answers of that which I ponder and to chase that which I crave, no matter how many broken noses I end up with.
To Hans, Javi, Jota, Grace, Rodri and Seba for every sleepless night, every coffee bean and every last sip of beer that got us here, it's not about where you're going is about who you're going with.
Thanks to my commission and my guide professor for the time and patience to guide me through this project.
I want to thank everyone at *Diablos*, tackles hurt the least and tries feel the best when you're on the field with friends rather than teammates.
To my freshman year friends, specially to Juan Pablo, not only for his remarkable name but for greeting me as an old friend on our very first day here.
Thanks to my high school friends, no matter when we meet again we shall always feel the same.
Finally I'd like to thank every mechanic, the dream team willy, the people at CDI, every classmate, group project partner, teacher, bike travelling companion, fellow teaching assistant, barista and every one who I met on the way.
Some might say happiness is only real when shared.

Contents

List of Tables **xi**

List of Figures **xii**

Introduction **1**
0.1 Motivation . . . 3
0.2 Objectives . . . 4
0.2.1 Scope . . . 5

1 Background and literature review **6**
1.1 Maintenance vibration analysis and bearing . . . 6
1.1.1 Lubricant Analysis . . . 7
1.1.2 Vibration Analysis . . . 7
1.2 Machine Learning, Deep Neural Networks, MLPs and GANs . . . 9
1.2.1 CNN . . . 9
1.2.2 Generative Adversarial Networks . . . 10
1.2.3 StarGAN . . . 12
1.2.4 Other Networks . . . 15

2 Methodology **18**
2.1 Procedures . . . 18
2.2 Datasets . . . 18
2.3 Classification . . . 20
2.4 Signal Pre-processing . . . 21
2.5 GAN Network . . . 22
2.6 CNN Network . . . 23
2.7 Evaluation Criteria . . . 24

3 Results **25**
3.1 CWR to MFPT . . . 26
3.1.1 Scalograms . . . 26
3.1.2 Spectrograms . . . 27
3.2 MFPT to CWR . . . 28
3.2.1 Scalograms . . . 29
3.2.2 Spectrograms . . . 30

4 Result Analysis **32**

5 Discussion & Future Works **34**

Conclusion **35**

6 Bibliography **38**

7 Annex **40**
7.1 GAN 40
7.2 CNN 57

List of Tables

3.1 Parameter evaluation 26
3.2 Proposed Model vs Simple Model 26
3.3 Confusion Matrix for classification based off of fake images 26
3.4 Confusion Matrix for classification without GAN network 27
3.5 Proposed model vs Simple model 27
3.6 Confusion Matrix for classification with GAN network 28
3.7 Proposed Model vs Simple Model 29
3.8 Confusion Matrix for classification with GAN network 29
3.9 Confusion Matrix for classification without GAN network 29
3.10 Proposed Model vs Simple Model 30
3.11 Confusion Matrix for classification with GAN network 30
3.12 Confusion Matrix for classification without GAN network 30

List of Figures

1.1 Typical signals and envelope signals in roller element bearings, [11]p.49 . . . 8
1.2 Envelope analysis procedure, [11]p.201 . . . 9
1.3 Example of a convolutional layer learning to recognise features from a simple key symbol. Learnt features allow the layer to output a high value if the feature is present and a low value when is not. . . . 10
1.4 Example of a max-pooling layer, for each section only higher values are kept 11
1.5 StarGAN's advantage relies on the reduced number of networks needed to translate in between multiple datasets. On the left the approach taken by traditional image-translation networks. On the right a graphical representation for StarGANs concept. image from [1] . . . 12
1.6 StarGAN's novel approach relies on changes to the discriminators roll and to the generators methodology . . . 14
1.7 StarGAN vs other image translation networks . . . 14
1.8 StarGAN Example Results . . . 15
1.9 CycleGAN examples . . . 17
1.10 CycleGAN faulty examples. . . . 17

2.1 a sketch of the methodology. 1. corresponds to pre-processing. 2 corresponds to training of the GAN network.3 is the implementation of the GAN. 4 is training and implementation of the CNN. MFPT and cwr are in arbitrary stages for demonstration . . . 19
2.2 An image of the test rig used in the CWR Dataset[17] . . . 19
2.3 Data is sorted according to the signal and the transformation . . . 22
2.4 Target labels inclusion in the networks . . . 22

3.1 Scalogram Discriminators loss with the MFPT scalograms as training images. 27
3.2 Scalogram Generator loss with the MFPT scalograms as training images. . . 27
3.3 Spectrogram Discriminators loss with the MFPT spectrograms as training images. . . . 28
3.4 Spectrogram Generator loss with the MFPT spectrograms as training images. 28
3.5 Scalogram Discriminators loss with the CWR scalograms as training images. 29
3.6 Scalogram Generator loss with the CWR scalograms as training images. . . . 30
3.7 Spectrogram Discriminators loss with the CWR spectrograms as training images. 31
3.8 Spectrograms Generator loss with the CWR spectrograms as training images. 31

5.1 On the left this works comprehension of StarGAN, on the right the author conceptualisation of StarGAN . . . 34

Introduction

Over the last few years the use of deep neural network has gained notoriety on the prognostics health management field[7]. Based on a concept first proposed in the seventies by Seppo Linnainmaa and popularized in 1986 by Geoffrey Hinton and David Rumelhart in their paper "Learning representations by back-propagating errors"[13].
Neural networks have seen a resurgence since advances in technology -both hardware and software- have made their widespread implementations feasible. Specially, the development of ever more powerful Graphic Processing Units and improved learning algorithms have paved the way for a deep learning algorithms and data analytic apogee.
In prognostics although traditional model-based approaches, are still considered to be more accurate and effective [7], data-driven approaches -such as the use of neural networks- present themselves as enticing alternatives for both complex systems where the amount of data makes traditional methods non-feasible and models in which a physical representation is hard to portray.
Conventionally data-driven methods focus on Condition Monitoring and Anomaly Detection, where the system is monitored and the data extracted is examined periodically in search of abnormalities.
Further methods incorporate diagnosis aspects to determine the source and/or implications of such abnormalities.
As with most neural network applications one of the key limitations on the use of data-driven approaches to maintenance is the scarcity of relevant data, specially labelled data, where signals from every possible state of each specific part are available to work with.
This work "Generative Adversarial Network Based Model for Multi-domain Fault Diagnosis" tackles this issue by providing a method where relevant data can be forged and labelled using previous data and computational methods.
On the following chapters motivation for this study as well as the scope and the objectives will be outlined. Afterwards background and literary reviews on the topics hereby treated will be presented. The next chapter will introduce the methodology followed. Then results shall be presented and discussion on them will follow. Finally conclusions on the study will be drawn and suggestions on further work will be included.
To better comprehend the purpose of this research a clarifying example will be made using pumps as an ever so slightly more tangible parallel to bearings. The hypothetical situation is that of a pumping station in which a working pump has been thoroughly studied and its vibration signal is well known and documented for all possible fault conditions .
As time would have it the hypothetical plant is upgraded in order to expand its capacity. Due to changing market conditions and availability a different pump is acquired for the expansion.

Starting from the assumption that the valuable data collected on the original pumps has been used to train and implement a self diagnosing system. The incorporation of the new pumps presents a few options for the plants maintenance. The first option is to abandon the self diagnosing system and deploy a crew specifically for the new pump, this approach goes against the hypothetical company's cutting edge, state of the art, future focused mentality and thus will not be considered.
In this scenario two options seem plausible, the first would be to use the algorithm tailored for the original pumps to try and diagnose the new pumps, with a limited effectiveness. This will be considered as the base line approach.
The second option as proposed in this work, is to use the data from the previous pumps along with the knowledge of the baseline state of the new pumps and advanced deep learning algorithms to effectively diagnose the unknown component's fault states.

0.1 Motivation

In an ever more global marketplace where suppliers are faced with a price scenario in which cost and production capacity are dictated by widely available manufacturing technology; maintenance and reliability, their cost, and their effect on availability became critical differentiating factor for generating revenue.
All though promising results have been shown: optimised availability, reducing excessive component replacement, and early detection, to mention a few. As the academic approach to deep learning algorithms and their applications in fault diagnosis develops, the hurdles that maintain it's applications to the industry constrained become ever more apparent.
To effectively provide advantages as diagnostics methods within maintenance programs 3 data-related issues must be sorted:

1. Lack of data
2. Unavailable or insufficient labelled data
3. Limited cases on available data

In this particular work's case, bearing use is quite widespread. It's failure modes have been thoroughly studied and documented. Despite this, under different conditions key parameters will change, previous data will be render useless and training a proper machine learning model will be an unfathomable task.
Bearings are an interesting case study for automated fault diagnosis methods, as mentioned they are common place for most manufacturing equipment, and -for reasons which will be later explored- are not good candidates for preventive maintenance. To develop an efficient diagnostics method with the use of machine learning techniques would allow for better maintenance plan, saving on excessive component replacement, enhanced plant security with early fault detection and -if compared with other predictive techniques such as vibration analysis- it would allow for a complex system to be developed by specialist but implemented by a less trained workforce.
This considerations make the proposal for a multi-domain fault diagnosis system based off of generative adversarial networks and Convolutional neural networks, an interesting proposal.

0.2 Objectives

- General Objective
  The global objective pursued by the present work is to develop and evaluate a model to diagnose the health state of a bearing vibration signal sample on a multi-domain approach with the use of a generative adversarial network and a Convolutional Neural Network.
- Specific Objectives
  - Build a GAN network capable of translating vibration signals across different domains.
  - Build a CNN that effectively classifies vibrational data when trained on images provided by the aforementioned GAN network.
  - Study the effect of different parameter on the effectiveness of the model (e.g. train vs test datasets, Scalograms vs Spectrograms, etc.)
  - Compare the effectiveness of the novel approach to a simplified deep learning approach

0.2.1 Scope

The reach of this work is constrained to the research and study of generative adversarial networks and simple classification networks and its possible applications to multi-domain fault diagnosis in bearing vibration signals.
The work will only be concerned with the MFPT bearing dataset and the CWR bearing dataset, no other mechanical elements nor different condition monitoring variables will be within the reach of this investigation.

Additional research on statistical analyses to quantify the effectiveness of the method or comparison to other similar works may be included, as well as suggestions for future works to further complement this investigation. In any case those are not in the intended plan.
The scope of this research focuses on the elaboration of a novel technique for fault diagnosis in limited data situations, based on multi domain generative networks and deep learning networks, specifically to determine if it is a suitable path to pursue in order to overcome previously stated difficulties in the application of deep neural networks to maintenance.

Chapter 1

Background and literature review

1.1 Maintenance vibration analysis and bearing

Condition based maintenance is currently considered the most efficient strategy for maintenance work in various industries [11]. Within the scope of 'Predictive Maintenance', is the concept of building a maintenance plan on the perceived condition of operating machines, rather than running them until break-down, or in a time-based fashion -Corrective and preventive maintenance respectively-.
A corrective maintenance task is "a maintenance task, performed to identify, isolate and rectify a fault so that the failed equipment, machine, or system can be restored to an operational condition"[4]; hence a Corrective Maintenance plan is based on this *Ex-Post* type of maintenance task. A corrective approach often maximises times between shut-downs, yet it can often imply larger times to repair, higher costs in spare parts, and catastrophic failure. Catastrophic failure refers to failure of a specific part or machine results in severe damage to itself, other components and even human costs. Due to such inconveniences corrective maintenance has fallen out of fashion, being used almost exclusively on low threat, non critical processes and industries where the temporary loss of one machine does not have a meaningful impact on production levels.

Preventive Maintenance, also referred to as Calendar-based, is maintenance done at regular intervals. Usually set in accordance with statistical information of elements, the concept runs on replacing the components when reliability starts to decay.
On the other hand preventive strategies allow for maintenance work to be planned in advance and greatly reduced catastrophic failure occurrence, however a small number of unforeseen failures can still occur and replacement is often times done in excess consuming unnecessary amounts of spare part components. Some parts with large statistical spread around the mean failure time can be replaced with half or more of it's useful life still remaining. Still, preventive maintenance is widely used in most industries as maintenance strategy. It's persistence can be in part attributed to the high capital costs preventive maintenance requires.

Predictive strategies aim to perform maintenance at optimum time, therefore reducing overall costs. Based on regularly monitoring machine condition with -primarily- two techniques, vibration analysis and lubricant analysis. All though other techniques are available such as termography and performance monitoring which are more application-specific, the two aforementioned ones are the main ones when monitoring internal conditions [11].

1.1.1 Lubricant Analysis

Lubricant analysis is the routine analysis of a machine lubricant health state, contamination and additive availability through sampling of the oil and posterior analysis.
Oil analysis can be divided into multiple categories:

- Chip Detectors
- Spectrographic Analysis Procedures
- Ferrography

Chip Detectors corresponds to filters and magnetic Plugs which study debris contained in oil samples, usually from worn parts or degraded oil.
Spectrographic analysis Procedures look for trace elements of wear as well as amount of additives, contamination or leaks of other fluids into the lubricant's circuit.
Ferrography is a microscopic investigation of magnetic debris -typically smaller than what is detected with chip detectors- and other non-magnetic elements that get carried by ferrous elements.
A typical oil analysis would also incorporate tests for viscosity, spectrometric analysis and moisture.
It's main drawbacks are the relative difficulty to identify the specific part of the system which is originating the abnormal results and the discrete nature of the technique. Oil analysis implies sampling, changing and refilling of the lubricant. Lubricant analysis are usually reserved for oil lubricants, all though grease sampling kits are available, it is much more difficult to apply this type of analysis to grease lubricants.

1.1.2 Vibration Analysis

Vibration analysis studies what is referred to as vibration signatures. Even in good conditions machine generate vibrations. Many of them are linked to periodic events, particular events from the machine's operation, hence the frequency with which such events repeat can provide strong evidence on the source of the event. This tight correlation is exploited by powerful diagnostics tools based on frequency analysis. Even when vibrations are not directly related to periodic events, if the correct techniques are used, they can act as powerful indicators to help diagnosis. vibrations is an extensive terms but -when talking about vibration analysis- it will *frequently* refer to signals as measured by an accelerometer.
'Rolling element Bearings are on of the most widely used elements in machines and their failure on of the most frequent reasons for machine breakdown' [11]. Furthermore they have

a very high statistical spread on their mean time to failure. Anyhow the vibration signals from faults in this particular type of bearing have been widely studied and well-proven techniques are available for their diagnostic. Figure 1.1 exemplifies acceleration signals with

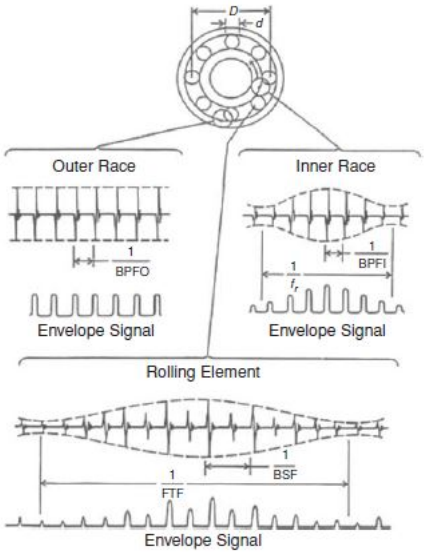

Figure 1.1: Typical signals and envelope signals in roller element bearings, [11]p.49

faults present in the different components of a ball bearing. The referenced frequencies from the image correspond to the modulation patterns from faults originating in this different components which are as follow:

- Ballpass Frequency
  - Outer Race
$$BPFO = \frac{nf_s}{2}\left(1 - \frac{\mathrm{d} \cdot cos(\phi)}{D}\right) \tag{1.1}$$
  - Inner Race
$$BPFO = \frac{nf_s}{2}\left(1 + \frac{\mathrm{d} \cdot cos(\phi)}{D}\right) \tag{1.2}$$
- Fundamental Train Frequency
$$FTF = \frac{fs}{2}\left(1 - \frac{\mathrm{d} \cdot (cos\phi)}{D}\right) \tag{1.3}$$
- Ball Spin Frequency
$$BSF = \frac{D}{2\mathrm{d}}\left[1 - \left(\frac{\mathrm{d} \cdot (cos\phi)}{D}\right)^2\right] \tag{1.4}$$

On this frequencies it is important to note that they correspond to kinematic frequencies assuming no slip, this is the cause for the advantage of envelope analysis over raw frequency analysis.
Envelope Analysis correspond to the application of a band-pass filtered in a high frequency band to the signal. With this, fault impulses are amplified by structural resonances. This is subsequently amplitude demodulated to produce the envelop signal. This procedure is demonstrated in the figure1.2.

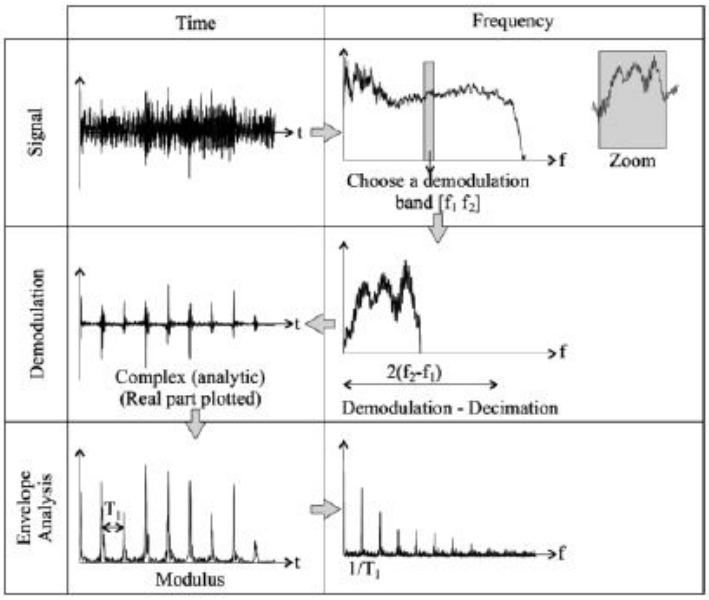

Figure 1.2: Envelope analysis procedure, [11]p.201

It is important at this point to note the difference between fault and failure. From now on fault shall be defined as 'the inability or incorrect functioning of an entity. It can be an inherent weakness of the design or implementation,and can result in a failure'. Whilst A failure is defined as 'the state or condition of not meeting a desired or intended objective, e.g. a service stops performing its required operation'[7]p.246.

1.2 Machine Learning, Deep Neural Networks, MLPs and GANs

Arthur Samuel in 1959 defined Machine Learning as the 'field of study that gives computers the ability to learn without being explicitly programmed to do so' as quoted in [5] Machine Learning algorithms work by recognising underlying patterns on training data. They allow for reduced workload to the people responsible and can reveal unsuspected or otherwise ignored relationships between parameters.

1.2.1 CNN

Convolutional Neural Networks are networks specially designed to work with images. Their architecture is composed of 3 distinct layers: Input layers, convolutional layers, and pooling layers. Dropout layers are often incorporated as well.

- Input Layers: They correspond to the first layer of the network and their objective is to incorporate information as it is to the network

- Convolutional layers are the heart of this type of network, they corresponds to filters swept over the image. this filters detect the presence of particular features in the image, having higher outputs when the learnt feature is present and lowers when is not, as shown in Figure 1.3.
- Pooling layers are downsampling layers which is to mean that they reduce the size of the image, they do this by outputting a smaller sample than what they take in. Maxpooling is the most commonly used pooling layer, it takes an input of a given size -such as 4x4- and outputs the highest value among them. Depicted by Figure 1.4
- Dropout layers are meant to reduce computational cost for the network and increase redundancy inside the network, they work by "dropping" a series of activations setting them to 0.

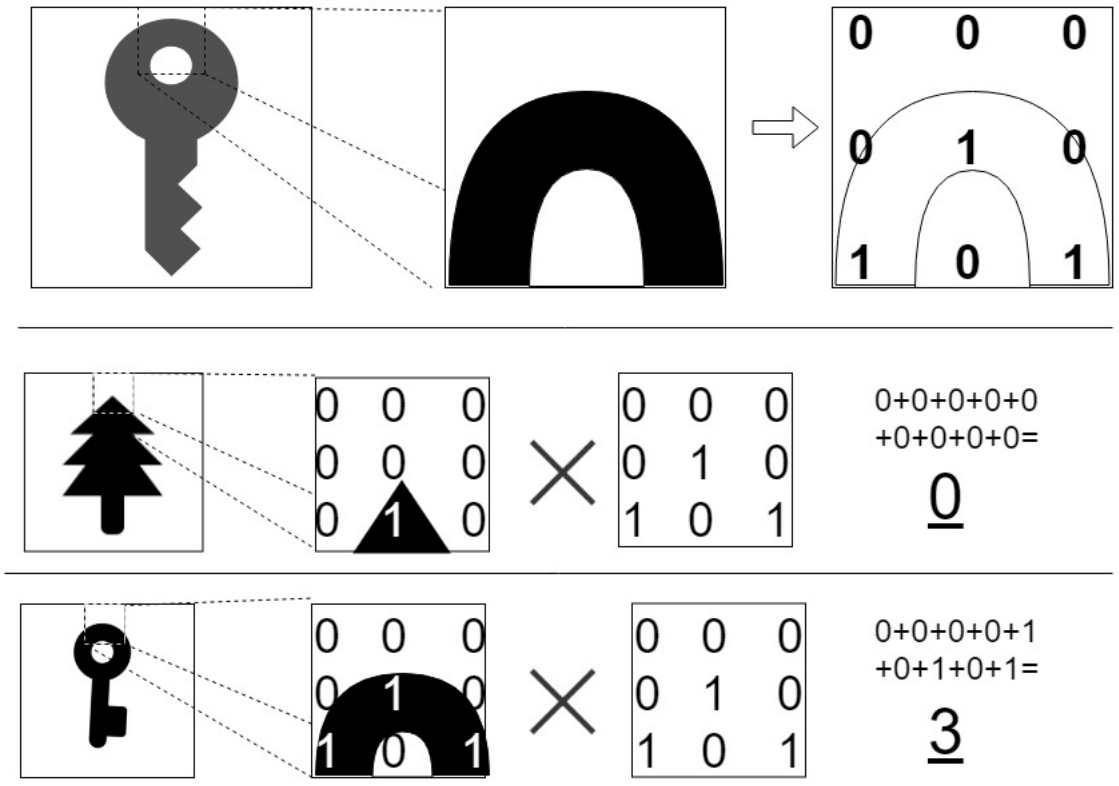

Figure 1.3: Example of a convolutional layer learning to recognise features from a simple key symbol. Learnt features allow the layer to output a high value if the feature is present and a low value when is not.

A Deep CNN is a neural network with several convolutional and pooling layers and a fully connected output.
Some examples of the application of CNN to vibrational data can be found on [8] and [18].

1.2.2 Generative Adversarial Networks

First introduced by Goodfellow in [3] Generative Adversarial Networks are ' generative models' i.e. a model that 'take[s] a training set, consisting of samples drawn from a distribution

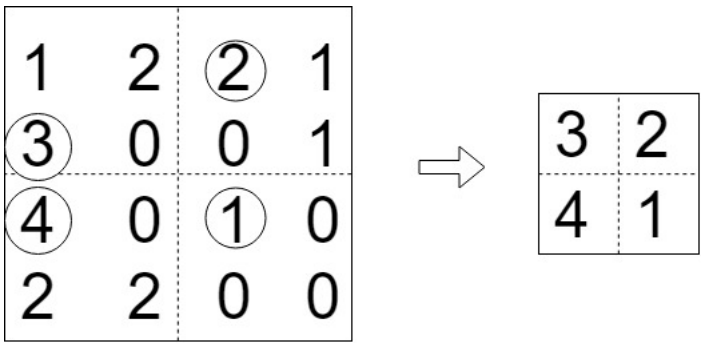

Figure 1.4: Example of a max-pooling layer, for each section only higher values are kept

p data and learns to represent an estimate of that distribution somehow. The result is a probability distribution p model' [2]p.2. Based on the idea of a game between two players, a generator and a discriminator who compete against each other. The parallel to a forger and a policeman is often drawn to clarify the role of the generator to create images as close as possible to the distribution from which the sample images are drawn and the discriminators role in discerning which images are real and which are "fakes".
On technical terms GANs are a structured probabilistic model with latent and observed variables. The two "players" in the game are two functions with inputs and parameters, which must be differentiable. Cost functions are independent for both players yet they depend on parameters from both players. In other words the Generator tries to minimize a function dependent on parameters P_G and P_D whilst only controlling P_G and the Discriminator vice-versa. The solution to the game is a Nash equilibrium which corresponds to the tuple of parameters that optimize the "global" cost function. GANs' posses two cost functions one for the generator and one for the Discriminator.
The discriminators loss is almost always a somewhat standard cross entropy.

$$J_{(D)}(P_{\mathrm{d}}, P_g) = \frac{1}{2}\mathbb{E}_{x\sim p_{\mathrm{d}ata}}[LogD(x)] + \frac{1}{2}\mathbb{E}_z[1 - LogD(G(Z))] \tag{1.5}$$

This corresponds to the the sum of 2 cross entropy functions, one with value 0 for fake images and one with value 1 for real images.
On the other hand different approaches are commonplace for the generators cost, the first and most simple of which is a zero-sum game. This approach presents the generator loss as the additive inverse of the discriminators loss:

$$J_{(G)} = -J_{(D)} \tag{1.6}$$

A second approach which seems to perform better in practice is :

$$J_{(G)} = -\mathbb{E}_z[LogD(G(Z))] \tag{1.7}$$

A strong conceptual difference sets this two approaches apart, the first one minimises the probability of the discriminator guessing when the image comes from the generator, whilst the second one maximises the chances of the discriminator being wrong.
Some other cost functions have also been applied in literature, such as a maximum likelihood with the use of a logistic sigmoid function, as presented in [3]. In any case, and despite theory suggesting otherwise, there does not seem to be a generator's cost functions which

yields overall better results according to [2].
It is worth mentioning that there are some issue with generative adversarial networks which are ubiquitous throughout their implementations. The most notable of which is non-convergence, where the network fails to advance towards optimisation. this can be expressed as never stabilising losses, failing to produce legitimate images or on **mode collapse**, amongst others. Mode collapse refers to the case when a subset (or all of) the inputs result in similar (o completely identical) outputs.So far mini-batch discrimination seems to be the best solution for this problem.
Other issues GANs seem to struggle with:

- Counting: an excessive amount of parts such as eyes or paws.
- perspective: when dimensionality is misunderstood by the network.
- global structure: figures which contain all relevant parts of objects in unnatural or impossible ways
- *Cherry-Picking*: When good results are a small subset of a much larger not-so-good pool of results
- Cost v/s Quality: Most times there does not seem to be a strong correlation between the networks cost and the qualitative results.

1.2.3 StarGAN

StarGAN is a neural network model first presented in [1]. Its focus is to find a solution to the limitation of current generative neural networks to handle more than two domains in image to image translation. Centred around the idea of building an infrastructure capable of handling multiple domains without the need of specific neural network for each combination. For this purpose, three major changes are introduced with respect to the original GAN:

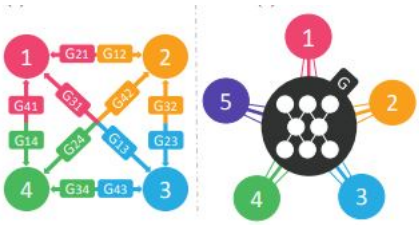

Figure 1.5: StarGAN's advantage relies on the reduced number of networks needed to translate in between multiple datasets. On the left the approach taken by traditional image-translation networks. On the right a graphical representation for StarGANs concept. image from [1]

The first of this changes is the incorporation of the target domain label into the image prior to the first layer of the neural network. The label is configured as a concatenation of one-hot labels and each position is tessellated and incorporated as a layer of depth to the image.
The second change is the addition of a feedback loop to the generator. When training, after an image has been translated to the target domain it is passed through the generator once more to create what they refer to as "reconstructed image". the reconstructed images

will serve as a metric of preservation of the original images characteristics other than the translated feature.
Finally, the last major change -with respect to the original GAN- is a mask vector which is "a simple but effective approach that enables joint training between domains of different datasets by adding a mask vector to the domain label. [Their] proposed method ensures that the model can ignore unknown labels and focus on the label provided by a particular dataset"[1]. The mask vector is a one-hot label with length equal to the number of datasets used for training. This vector is part of the concatenated label mentioned in the first point. Also relevant is the second role for the discriminator in classification.
The loss for StarGAN is divided in: Adversarial Loss, Domain Loss and Reconstruction Loss.

- Adversarial loss

$$\mathcal{L}_{adv} = \mathbb{E}_x[logD_{src}(x)] + \mathbb{E}_{x,c}\Big[log(1 - D_{src}(G(x,c)\Big] \tag{1.8}$$

- Domain Loss

$$\mathcal{L}^R_{c,s} = \mathbb{E}_{x,c'}\Big[- logD_{cls}(c'|x)\Big] \tag{1.9}$$

$$\mathcal{L}^F_{c,s} = \mathbb{E}_{x,c}\Big[- logD_{cls}(c|G(x,c)\Big] \tag{1.10}$$

- Reconstruction Loss

$$\mathcal{L}_{rec} = \mathbb{E}_{x,c,c'}\Big[||x - G(G(x,c),c')||_1\Big] \tag{1.11}$$

Here x : Real Images, c' : Original Domain,c :Target Label, D_{src} : discriminator's source output(Real or Fake), D_{cls} : discriminator's class output, $G(a,b)$: generated image from input 'a' and target label 'b', F: superindex for *Fake*, R: superindex for *Real*.
Hence the full objective is:

- Full Objective:

$$\mathcal{L}_D = \mathcal{L}_{\text{adc}} + \lambda_{cls} \cdot \mathcal{L}^R_{cls}\mathcal{L}_D = \mathcal{L}_{\text{adc}} + \lambda_{cls} \cdot \mathcal{L}^F_{cls} + \lambda_{\text{rec}} \cdot \mathcal{L}_{\text{rec}} \tag{1.12}$$

The λ factors correspond to hyper-parameters to tune the importance of each loss.
The architecture of StarGAN is as follows: Generator:

1. (3 x) Convolutional Layer
   - Instance Normalization
   - ReLU
2. (6 x) Resnet Blocks
   - Convolutional Layer
   - Instance Normalization
   - ReLU
   - Convolutional Layer
   - Instance Normalization → Result
   - Result + Input
3. (2 x) Deconvolutional Layer

- Instance Normalization
- ReLU

4. tanh

Discriminator:

1. (6x) Convolutional Layer
   - Leaky ReLU
2.
   - Class: Convolutional Layer
   - Source: Convolutional Layer

In StarGAN's architecture the discriminator has two outputs, one with the size of the number of classes which corresponds to the one-hot label classifying the image as one specific class. The other is a one vector output which corresponds to the classic role of the discriminator identifying the source of the image as either real or generator-made.

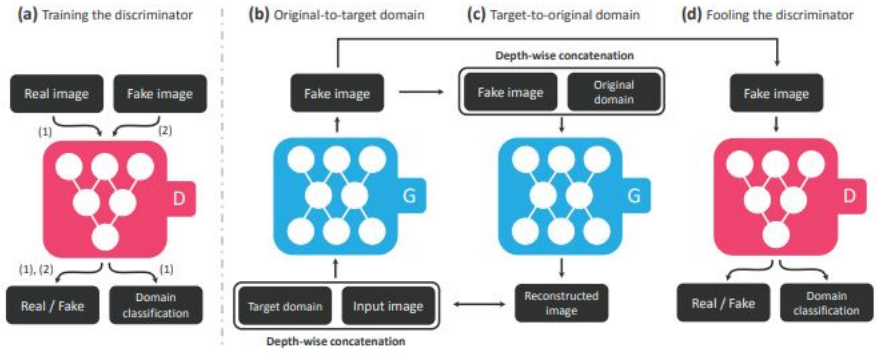

Figure 1.6: StarGAN's novel approach relies on changes to the discriminators roll and to the generators methodology

Some example results from [1] are presented in figures 1.7 and 1.9.

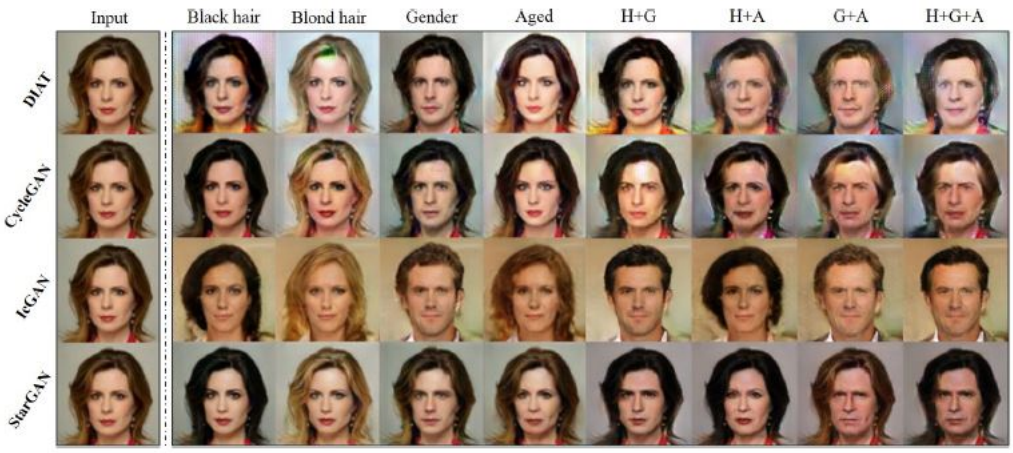

Figure 1.7: StarGAN vs other image translation networks

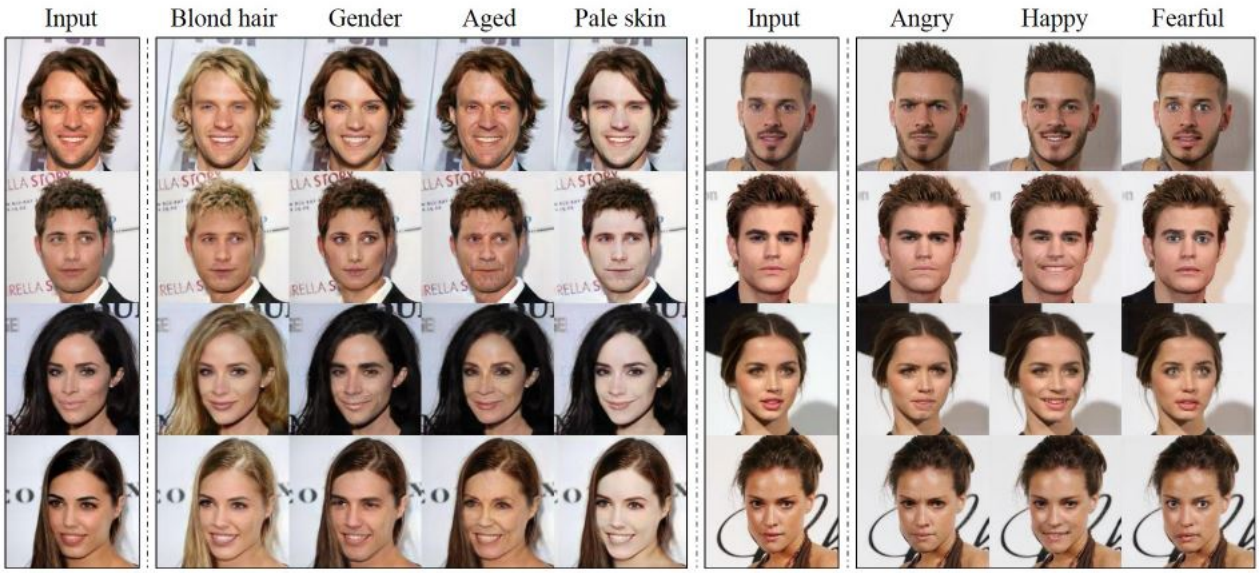

Figure 1.8: StarGAN Example Results

1.2.4 Other Networks

CycleGAN is a neural network proposed in [19] it was proposed as a response to other image translation networks limitations in regards to datasets without paired images. Paired images refers to datasets which contain the same image or an equivalent image for each domain: [Example] It was the network responsible for introducing the cyclic loss previously mentioned in StarGAN. It's concept is that of training to 'collaborative' pairs of GAN, implying two generators and to discriminators, one for the translation $F : Y \rightarrow X$ and $G : Y \leftarrow X$. In this configuration cyclic loss is dependent on the two networks, so it is imperative to train both, they cannot by disengaged.
Loss is defined by the authors as

$$\mathcal{L}^{y}_{\text{adv}} = \mathbb{E}_{y}\Big[logD_{y}(y)\Big] + \mathbb{E}_{x\sim p_{\text{data}}}\Big[log(1 - D_{y}(G(x))\Big] \tag{1.13}$$

$$\mathcal{L}^{x}_{\text{adv}} = \mathbb{E}_{x}\Big[logD_{x}(x)\Big] + \mathbb{E}_{y\sim p_{\text{data}}}\Big[log(1 - D_{x}(G(y))\Big] \tag{1.14}$$

$$\mathcal{L}_{Cyc} = \mathbb{E}_{x\sim p_{\text{data}}}\Big[||F(G(x)) - x||_{1}\Big] + \mathbb{E}_{y\sim p_{\text{data}}}\Big[||G(F(y)) - y||_{1}\Big] \tag{1.15}$$

And the full objective becomes:

$$\mathcal{L}_{full} = \mathcal{L}_{Cyc} + \mathcal{L}^{y}_{\text{adv}} + \mathcal{L}^{x}_{\text{adv}} \tag{1.16}$$

The models architecture is as follows: Generator:

1. (3 x) Convolutional Layer
   - Instance Normalization
   - ReLU
2. (9 x) Resnet Blocks

    - Convolutional Layer
    - Instance Normalization
    - ReLU
    - Convolutional Layer
    - instance Normalization → Result
    - ReLU(Input + Result)
3. (3 x) Deconvolutional Layer
4. tanh

Discriminator:

1. Convolutional Layer
    - ReLU
2. (2 x) Convolutional Layer
    - Instance Normalization
    - ReLU
3. Convolutional Layer

While training, the negative log-likelihood was replaced by the least-square loss, and the discriminator was trained on a history of generated images rather than the ones produced by the last generator.
It is Mentioned by the authors that in some cases, specially when translating *art-styles* the addition of identity mapping which is a loss of function:

$$\mathcal{L}_{\text{id}} = \mathbb{E}_{x \sim p_{data}(y)} \Big[||G(y) - y|| \Big] \tag{1.17}$$

This helps avoid unnecessary changes such as background colour.
Some limitations to this network are:

- All though it performs well when texture and colour change, it does not perform well changing the shape of objects
- When new elements not present in the dataset appear on the testing images it does not react well.
- Performance is way behind that of networks trained on paired data

CycleGAN applications are abundant, some note worthy examples are shown below as well as some particularly faulty images, all images are taken from [19]. Unable to be reproduced in this written work are videos made with this network which work by translating all the frames from the original video.

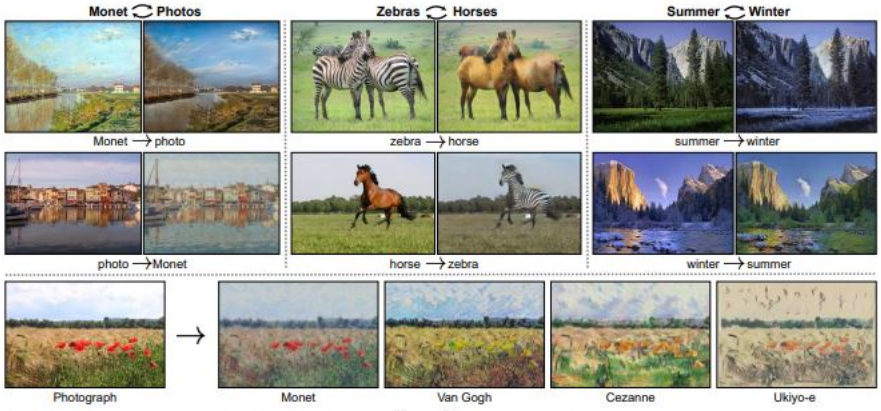

Figure 1.9: CycleGAN examples

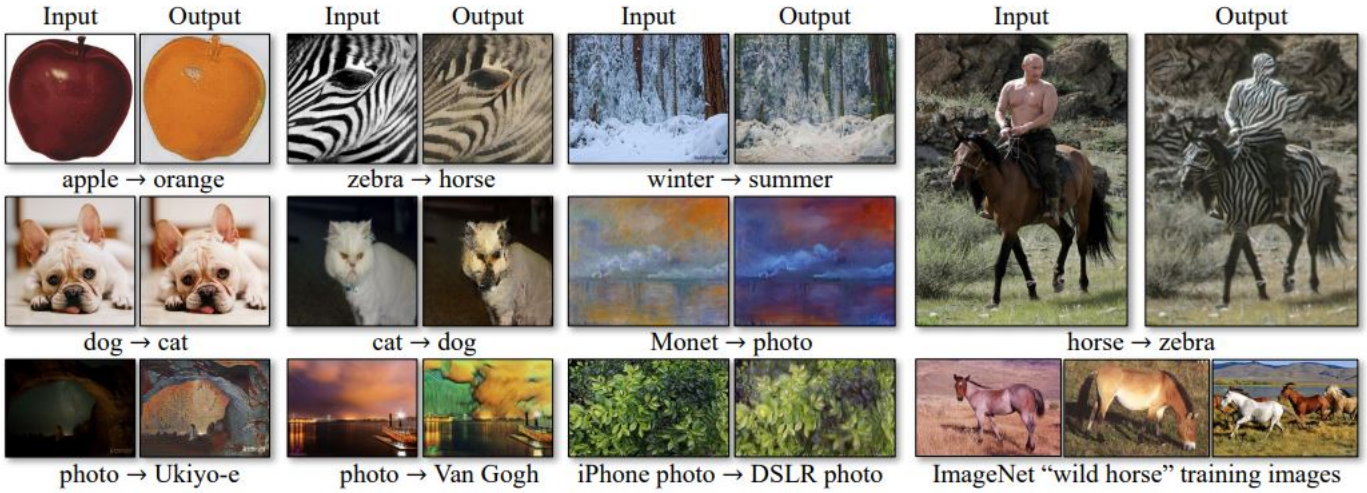

Figure 1.10: CycleGAN faulty examples.

Chapter 2

Methodology

In order for the proposed objectives to be carried out the following methodology is proposed. A process of six steps to go from an accelerator vibration signal to a predictive model is implemented.
The process begins with a segregation of the original data in categories which will later be the domains. Signals are then transformed into images through the application of mathematical methods. The output of this step are images of a 256 x 256 pixels size. Such images are either subsequently fed as training input to the GAN-like neural network or left out for the later stages. Another portion is provided to the previously trained generative network and the outcome is utilised to train a CNN classifier. Finally the remaining portion of the signal images is used to evaluate the results. The evaluation of the result is performed through a confusion matrix and other relevant indicators. Figure 2.1 illustrates the procedure using arbitrary datasets for clarifying purposes.

The procedure is further discussed below.

2.1 Procedures

2.2 Datasets

Limited availability of robust datasets implied reduced decision making in selection.
Used datasets are Case Western Reserve University Bearing Data Center (CWR) and the one from the Society of Machine Failure Prevention Technology (MFPT):

- CWR: Case Western Reserve University provides a bearing dataset originated from the set-up shown in figure 2.3. With Samples taken at 12,000 Samples per second and 48,000 Samples per second, this set of data was generated using a 2 hp motor, a torque transducer, a dynamometer and control electronics. The test bearing work supporting the motor shaft.

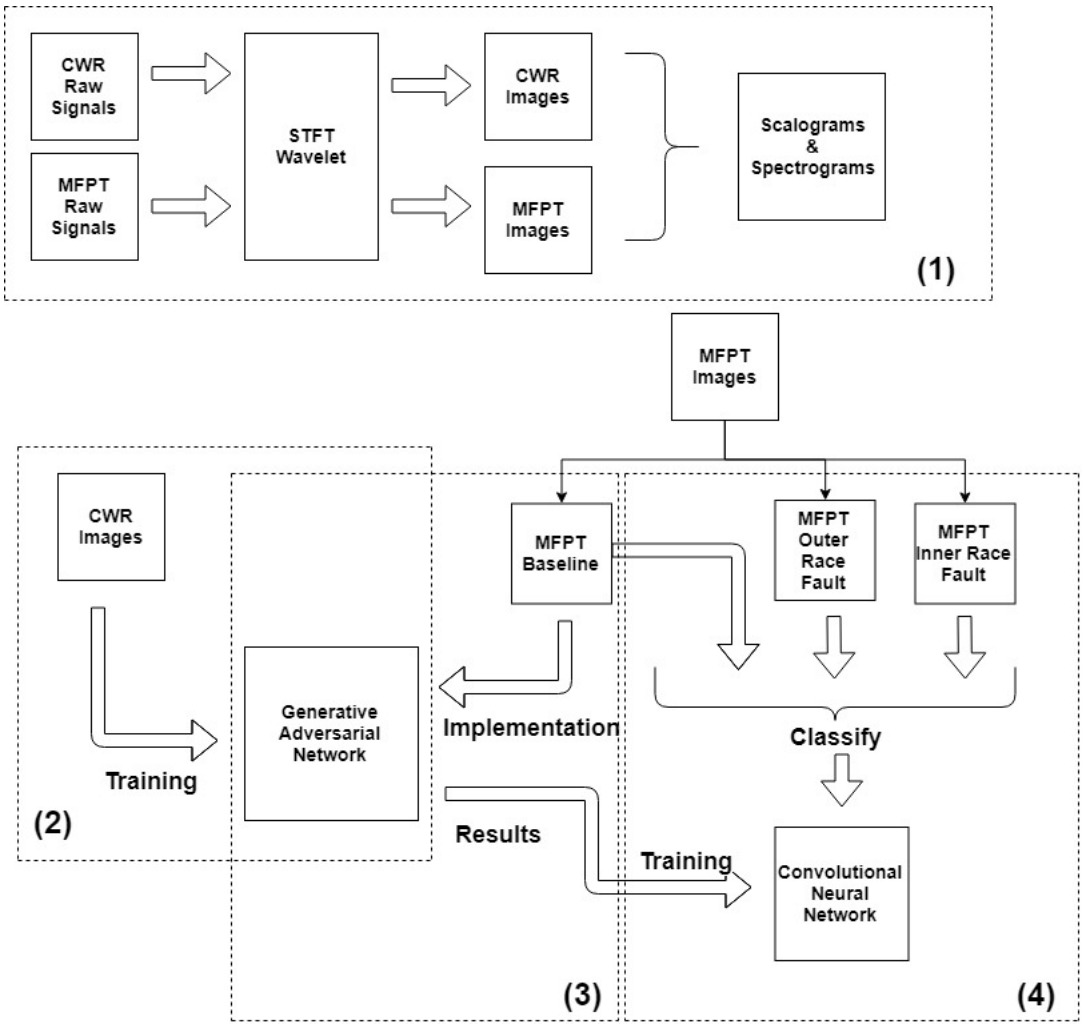

Figure 2.1: a sketch of the methodology. 1. corresponds to pre-processing. 2 corresponds to training of the GAN network.3 is the implementation of the GAN. 4 is training and implementation of the CNN. MFPT and cwr are in arbitrary stages for demonstration

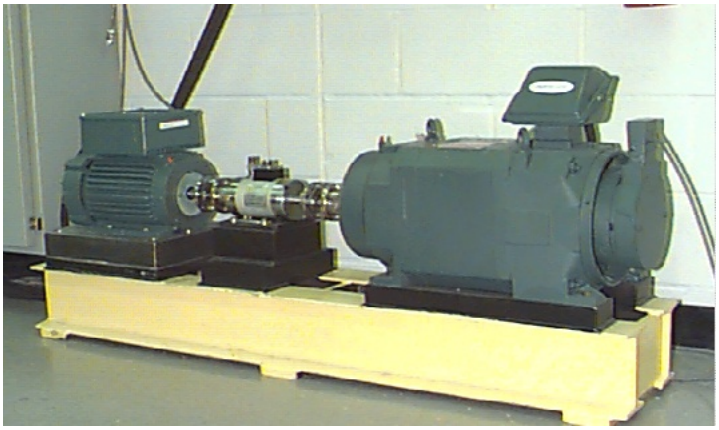

Figure 2.2: An image of the test rig used in the CWR Dataset[17]

Using electrodischarge machining single point faults were introduced with diameters of

7,14,21,28 and 40 mils. **1 mill** is equivalent to **0.001 inches**.
Data was collected from an accelerometer placed at 12 o'clock position with a magnetic base.
Taking in considerations the stationary nature of outer race faults, experiments were made with the fault positioned at 3 o'clock, 6 o'clock and o'clock.
Bearing Specifications:

- Inside Diameter = 0.9843 inches
- Outside Diameter = 2.0472 Inches
- Thickness = 0.5906 Inches
- Ball Diameter = 0.3126
- Pitch Diameter = 1.537
- Elements= 9

Where inside and outside diameters correspond to the dimensions of the outer and inner casing of the bearing, respectively.

- MFPT: The Society for Machine Failure Prevention technology makes available a bearing dataset from what is described as "a normal test rig" with "the hope that researchers and CBM practitioners will improve upon techniques, and consequently, mature CBM systems, faster"[16].
  samples where taken at
  - 48,828 samples per second and at 0,50,100,150,200,250 and 300 lbs of load for inner race fault.
  - 48,828 samples per second and 25, 50, 100 150 200, 250 and 300 lbs of load for outer race fault.
  - 97,656 samples per second and 270 lbs of load for another outer race fault set
  - 97,656 samples per second and 270 lbs of load for baseline condition

  Samples are taken at 48,828 samples per second and at 0,50,100,150,200,250 and 300 lbs of load for inner race fault, at 48,828 samples per second and

Bearing Specifications:

- Thickness = 0.5906 Inches
- Ball Diameter = 0.235
- Pitch Diameter = 1.245
- Elements = 8

No information is provided on the casing sizes of the bearing used for the MFPT Dataset.

2.3 Classification

Classification necessarily begins with an examination of the information available for every dataset. The CWR Dataset was categorised according to Fault Diameter, Motor Load, Motor Speed, sampling frequency and fault condition. MFPT on the other hand was categorised by

load, sampling frequency and by fault condition. The most robust of the parallels is the fault condition. The sampling frequency was considered as well but for another purposes. The CWR motor loads were considered at some point but were dropped for divergence issues.

2.4 Signal Pre-processing

The first step hat must be taken is to eliminate the differences in sampling frequencies. This is down through down-sampling, where points samples from the original signals are taken sparsely so that the time difference between them becomes the same. The specific type of neural networks proposed for this study is image-centred, hence the data had to be pre-processed in order to make images out of the accelerometers vibration signals. An image size of 256x256 was established.
Data pre-processiong was carried out through two different methods: Short Time Fourier Transform Spectrograms and Wavelet Analysis Scalograms.

Short Time fourier transform: A short time fourier transform is a method in which signal is segmented in the time dimension and a fourier transform is applied on each segment.
This transform is described by the formula:

$$s(f,\lambda) = \int_{-\infty}^{\infty} x(t) \cdot w(t-\lambda) \cdot exp(-j\pi ft)\mathrm{d}t \tag{2.1}$$

where w corresponds to a zero centred window. The discrete case is homologous.The size of the window necessarily implies a trade-off between time resolution and frequency resolution The squared amplitude is used to plot a time-frequency diagram, referred to as *Spectrogram.*

$$|s(f,\lambda)^2| = Squared Amplitude \tag{2.2}$$

a window length equal to three revolutions was used to create each spectrogram.
Wavelet Transform: Wavelet analysis is a type of time frequency analysis. The signal is decomposed in a "family" of wavelets, ergo, a fixed shaped wavelet which is shifted and dilated in time.

$$W(a,b) = \frac{1}{\sqrt{a}} \int_{\infty}^{\infty} x(t)\psi^*(\frac{t-b}{a})\mathrm{d}t \tag{2.3}$$

Where $\psi(t)$ corresponds to the "mother" wavelet translated by b and dilated by a. a which is usually referred to as "scale" is an inverse representation of the log frequency.
Wavelets can be thought of as impulse response filters with zero phase shift and constant percentage bandwidth properties.
Wavelets are particularly good at time localisation at high frequencies which is a key factor in their use for detecting local events in a signal. They have been used for detecting local faults in gears and bearings as shown in [12][10][15].
In general wavelets should be chosen to have the greatest similarity to the features in the signal which are to be extracted. Often the information contained in the graphical representation of the transform, *Scalogram* is not obvious and image analysis techniques are needed to characterise them.
To compensate for unbalanced datasets, salt and pepper noised images and horizontally flipped images where generated as data augmentation.

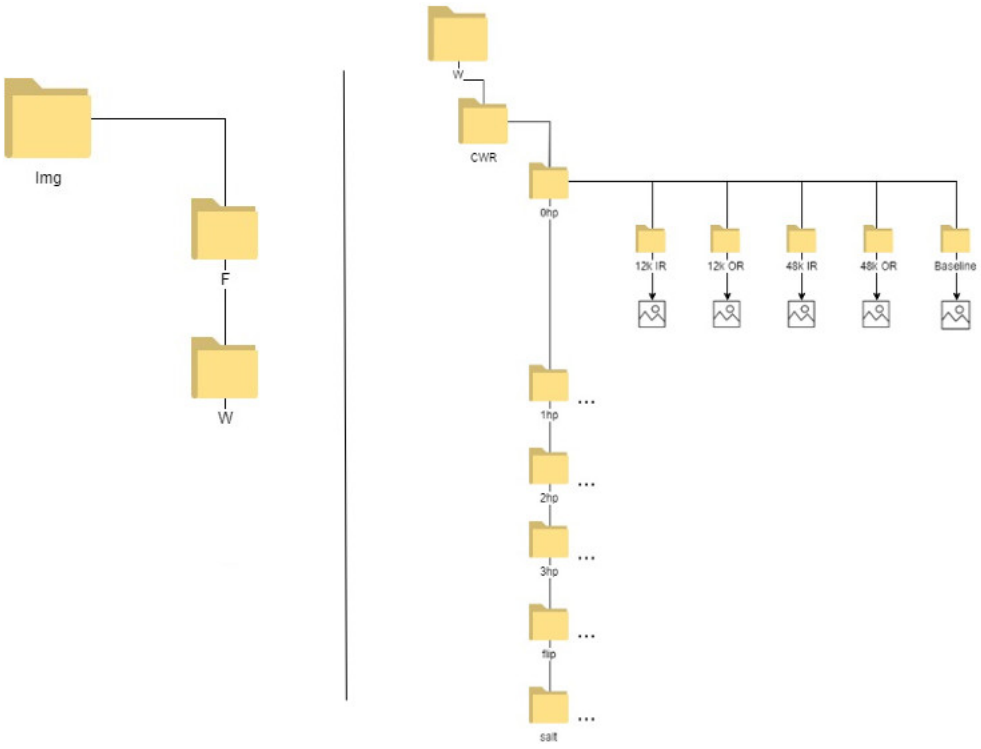

Figure 2.3: Data is sorted according to the signal and the transformation

2.5 GAN Network

For The Generative Adversarial Network, the model was based off of the StarGAN paper mentioned in Chapter 1.
The network loads the images in batches of 8 and for each one, the image is read in black and white. A random target is designated for training purposes. Just as in StarGAN the target label is separated, teselated and concatenated in dimensional space as figure 2.4 illustrates. As the final version for the network only includes images from one dataset for training, there is no mask label. The Generator and discriminator performs similarly to the starGAN network

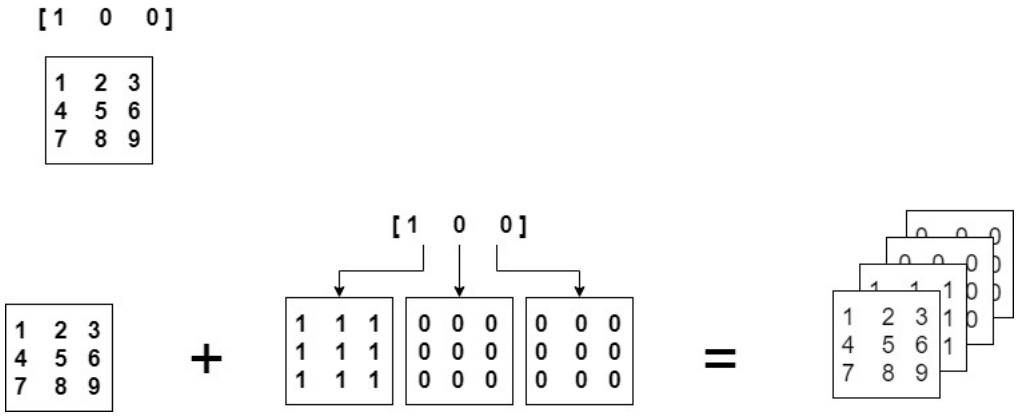

Figure 2.4: Target labels inclusion in the networks

with a discriminator/classifier and a generator with a cyclic loss. The main differences are:

Loss Function, optimiser and architectures.
The Loss function for all components is changed from cross entropy to least-square loss, as per CycleGAN. The change allows for a more stable training. This change helps avoiding fast divergence on the early iterations-to which the network is prone-. To avoid mode collapse mini batch discrimination and identity discrimination are added to the loss function of the generator with a λ factor.
Generator Loss:

$$\mathcal{L}_G = \mathcal{L}_{cyc} \cdot \lambda_{cyc} + \mathcal{L}_{\mathrm{adv}} \cdot \lambda_{\mathrm{adv}} + \mathcal{L}_{class} \cdot \lambda_{class} + \mathcal{L}_{mb} \cdot \lambda_{mb} + \mathcal{L}_{I\mathrm{d}} \cdot \lambda_{\mathrm{i}}\mathrm{d} \tag{2.4}$$

Cyc : Stands for Cyclic, $A\mathrm{d}v$: Stands for adversarial, $Class$: Stands for classification, mb : Stands for mini-batch and $I\mathrm{d}$: Stands for identity.

Discriminator Loss:

$$\mathcal{L}_D = \mathcal{L}^R_{\mathrm{adv}} \cdot \lambda^R_{\mathrm{adv}} + \mathcal{L}^F_{\mathrm{adv}} \cdot \lambda^F_{\mathrm{adv}} + \mathcal{L}_{class} \cdot \lambda_{class} \tag{2.5}$$

$A\mathrm{d}v^F$: Stands for adversarial evaluated over fake images, $A\mathrm{d}v :^R$ Stands for adversarial evaluated over real images, $Class$: Stands for classification.
Changes in the optimiser are evaluated: RMSprop, ADAM and Gradient Descent are tested. a Different architecture has been proposed. All though it is heavily based on Stargan, due to the dataset size being considerably smaller, the amount of Resnet Blocks is reduced. Proposed changes of activation functions and normalisation layer from ReLU to Leaky ReLU and from Instance Normalisation to Batch Normalisation.
Learning Rate is set to 10^{-6} with a linear decay to 0, from half of the total iterations to the end.

2.6 CNN Network

The convolutional neural network is based on the one introduced in [8]. architecture:

- convolutional layer
  - ReLU
- Max Pooling
  - Stride=2
- ReLU
- Dropout Layer
- ArgMax Layer

The network is trained during 20.000 iterations with a learning rate of 10^{-6}. It reads images of 256x256 pixels in grey scale in batches of 8.

2.7 Evaluation Criteria

To measure the results with the objectives in mind, a confusion matrix and the indexes for precision, sensitivity, specificity and F-1 score were used.
all though 3 classes exist other than accuracy and confusion matrix, the metrics will evaluate the amount of positives and negatives, being baseline case positive and fault cases negatives. Accuracy is the measure of correctly classified images. It will be measured both for class accuracy and for fault state accuracy: Class accuracy then considers each class individually, while fault state class only considers baseline vs faulty, ergo an inner race fault considered as an outer race fault, is still counted as correct. Class Accuracy:

$$Class_{Accuracy} = \frac{\text{Correctly Classified Images}}{\text{Total Images}}$$

Fault State Accuracy:

$$FS_{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{Total}$$

Precision is a measure of how frequently positive predictions correspond to true positives. Precision:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

formula Sensitivity measures how well real positives are classified, hence:

$$Sensitivity = \frac{\text{True Positives}}{TruePositives + FalseNegatives}$$

Specificity is used as a measure of false positives, it corresponds to the ratio of true negatives to total negatives:

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives}$$

F-measure is used as a measurement of the balance between precision and sensitivity

$$F_1 = \frac{2x\text{True Positive}}{2x\text{True Positives} + \text{False Positives} + FalseNegatives}$$

in Addition to this final evaluation criteria, some methods were used to evaluate intermediate steps, in an effort to isolate the results from the different GAN networks.

- CNN:
  To evaluate the effectiveness of the CNN model, the network is trained on the original datasets, a section of the original datasets are used to evaluate the results
- GAN:
  To evaluate the effectiveness of the GAN model, the network is trained on the original images, then the translated images are classified by the previously mentioned CNN.

It is important to note that this type of evaluation is merely for guidance purposes, a network must be tailored to the real images it is meant to work with, in any case they provide helpful insights of the location of the weaknesses in the complete build.

Chapter 3

Results

The model was run on both datasets as per the methodology presented on previous sections and codes presented in the annex section.
Using Matlab and parameters later described, scalograms and spectrograms were created with the data from aforementioned datasets.
With spectrograms and scalograms (in parallel) A base model with all ReLUs as Leaky ReLU, a Gradient Descent Optimiser, no minibatch, no identity loss, resnet blocks as per StarGAN, and Instance Normalisation is proposed, parameters are evaluated on the table

To obtain results, signals from MFPT and CWR were processed in Matlab in order to create both spectrograms and scalograms.
For both a window length of 3 revolutions was used -in accordance with each signals frequency- and down-sampling as to make signals with different sampling rates comparable, an overlap of half a window length was used to better capture the signal. Matlab's native CWT and STFT functions were used for the process.
This images were run on parallel through the methodology described in the previous section. The code included as an annex details the application of such methodology.
Different parameters were tested to optimiser the final model, for this the activation function used in the model, the optimiser, the normalisation method and the weight of the different losses, as well as some particular options - mini-bath and identity loss inclusions- were adjusted.
This variations are part of the final tuning stages of the model. Major decisions were made in the development stages within the strive for convergence. Such changes did not provide quantifiable and therefore are not illustrated in this section.
Testing of parameters was carried out by running the model on a low iteration basis for each variation and thus, obtaining the results portrayed on table 3.1

In light of this results, the Adam Optimiser was used, activation functions were set as ReLU functions, instance normalisation was applied, losses were weighted as suggested in [1] and it was decided not to use identity loss nor mini batch discrimination.

Table 3.1: Parameter evaluation

	Class Accuracy	Fault State Accuracy	Precision	Sensitivity	Specificity	F-1 Score
Base	0.28	0.66	1.0	0.03	1.0	0.06
Resnet Block	0.35	0.64	Null	0.0	1.0	0.00
Adam	0.32	0.66	0.75	0.07	0.97	0.13
Mini Batch	0.31	0.67	Null	0.0	1.0	0.0
Identity Loss	0.34	0.65	Null	0.0	1.0	0.0
Relu's	0.37	0.69	0.41	0.08	0.91	0.13
Batch Normalization	0.29	0.30	0.30	1.0	0.0	0.45
Equal Losses	0.349	0.7	0.0	0.0	0.98	0.0
Adam+ReLU	0.30	0.30	0.30	1	0.0	0.46
No GAN	0.57	0.78	0.58	1.0	0.57	0.73

3.1 CWR to MFPT

At first Data was sorted as exemplified in image 2.1. Here the GAN network was trained using images from the MFPT, and so, transforming CWR images into the MFPT fault domain.

3.1.1 Scalograms

Greyscaled scalograms were used as one of the alternative images. Results were evaluated using metrics as per section 2.7
The baseline model was run for comparison with the same inputs for training and evaluating. Results are shown on table 3.2.

Iterations	Class Accuracy	Fault State Accuracy	Precision	Sensitivity	Specificity	F-1 Score
30000 iters	0.29	0.49	0.36	0.94	0.008	0.52
No GAN	0.41	0.58	0.4	1.0	0.21	0.58

Table 3.2: Proposed Model vs Simple Model

Tables 3.3 and 3.4 show confusion matrices for classification as to clarify the nature of the results, on the generative approach and the baseline approach.

Table 3.3: Confusion Matrix for classification based off of fake images

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	131	0	8
	Inner Race	152	2	14
	Outer Race	77	85	0

The evolution of the cost of the loss function for both the generator and the discriminator is illustrated in figures 3.1 and 3.2.

Table 3.4: Confusion Matrix for classification without GAN network

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	137	0	0
	Inner Race	102	30	32
	Outer Race	97	47	24

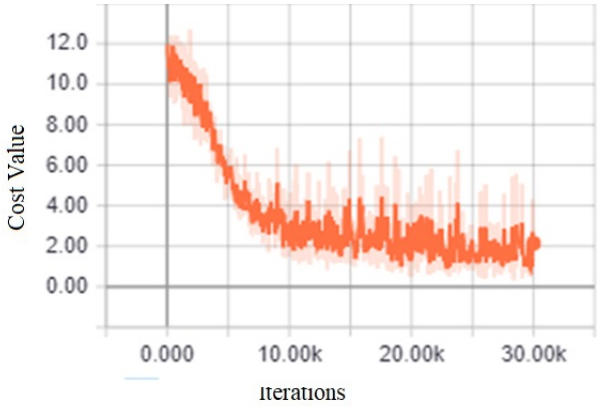

Figure 3.1: Scalogram Discriminators loss with the MFPT scalograms as training images.

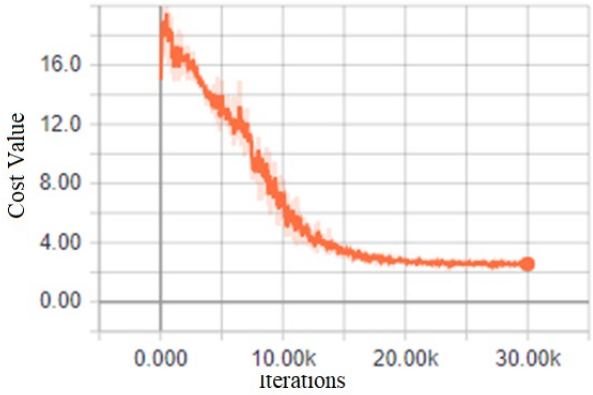

Figure 3.2: Scalogram Generator loss with the MFPT scalograms as training images.

3.1.2 Spectrograms

Grey-scaled scalograms were used as one of the alternative images. Results were evaluated using metrics as per section 2.7
The baseline model was run for comparison with the same inputs for training and evaluating. Results are shown on table 3.1.2.

Table 3.5: Proposed model vs Simple model

Iterations	Class Accuracy	Fault State Accuracy	Precision	Sensitivity	Specificity	F-1 Score
30000 iters	0.35	0.69	Null	0.0	1.0	0.0
No GAN	0.35	0.69	Null	0.0	1.0	0.0

Tables 3.6 and shows confusion matrix for classification as to clarify the nature of the results.

Table 3.6: Confusion Matrix for classification with GAN network

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	0	0	147
	Inner Race	0	0	153
	Outer Race	0	0	134

The evolution of the cost of the loss function for both the generator and the discriminator is illustrated in figures 3.3 and 3.4.

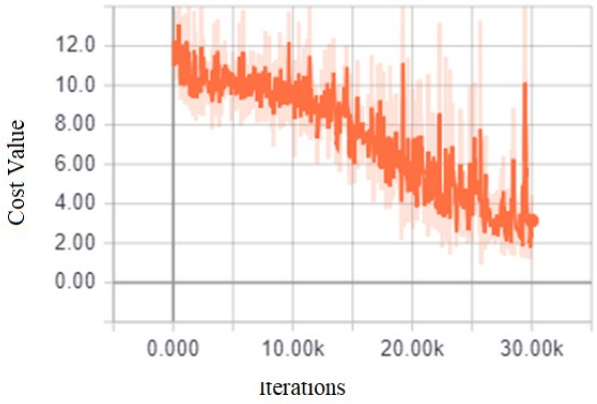

Figure 3.3: Spectrogram Discriminators loss with the MFPT spectrograms as training images.

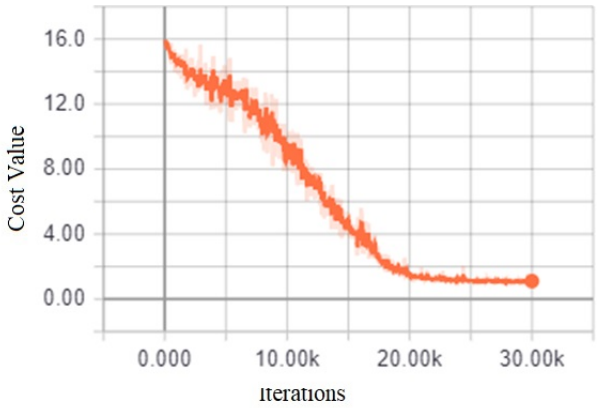

Figure 3.4: Spectrogram Generator loss with the MFPT spectrograms as training images.

3.2 MFPT to CWR

Afterwards the data was inversely sorted, training the generative adversarial network on the Case Western Reserve University dataset and using baseline Machine Failure Prevention Technology Society dataset translated on the GAN to train the convolutional neural network.

3.2.1 Scalograms

Grey-scaled scalograms were used as one of the alternative images. Results were evaluated using metrics as per section 2.7
The baseline model was run for comparison with the same inputs for training and evaluating. Results are shown on table 3.7.

Table 3.7: Proposed Model vs Simple Model

Iterations	Class Accuracy	Fault State Accuracy	Precision	Sensitivity	Specificity	F-1 Score
30000 iters	0.32	0.62	0.0	0.0	75	0.0
No GAN	0.47	0.94	0.94	0.83	0.95	0.88

Tables 3.8 and 3.9 show confusion matrices for classification as to clarify the nature of the results, on the generative approach and the baseline approach.

Table 3.8: Confusion Matrix for classification with GAN network

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	0	209	8
	Inner Race	75	250	0
	Outer Race	9	240	0

Table 3.9: Confusion Matrix for classification without GAN network

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	170	34	0
	Inner Race	11	162	156
	Outer Race	0	215	43

The evolution of the cost of the loss function for both the generator and the discriminator is illustrated in figures 3.5 and 3.2.

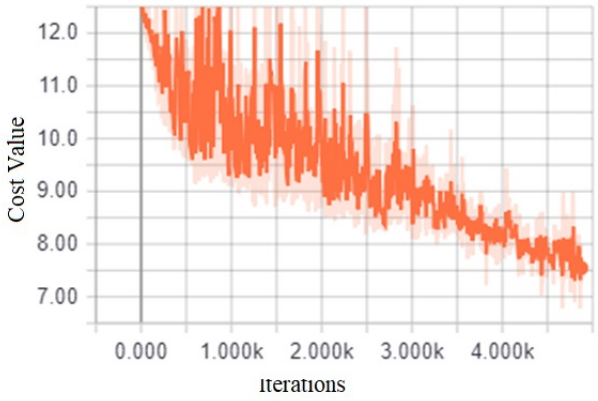

Figure 3.5: Scalogram Discriminators loss with the CWR scalograms as training images.

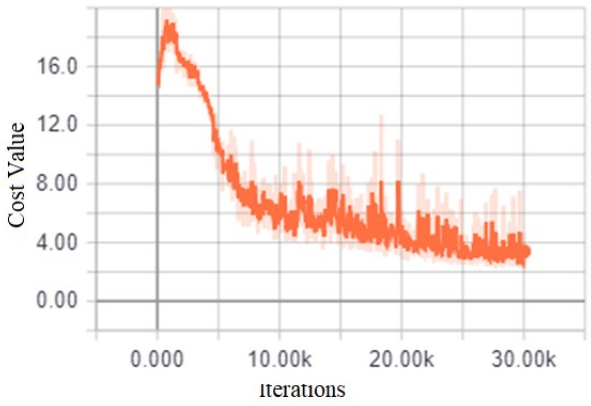

Figure 3.6: Scalogram Generator loss with the CWR scalograms as training images.

3.2.2 Spectrograms

Grey-scaled spectrograms were used as one of the alternative images. Results were evaluated using metrics as per section 2.7
The baseline model was run for comparison with the same inputs for training and evaluating. Results are shown on table 3.10.

Table 3.10: Proposed Model vs Simple Model

Iterations	Class Accuracy	Fault State Accuracy	Precision	Sensitivity	Specificity	F-1 Score
30000 iters	0.32	0.32	0.32	1.0	0.0	0.48
No GAN	0.40	0.59	0.44	0.69	0.35	0.54

Tables 3.11 and 3.12 show confusion matrices for classification as to clarify the nature of the results, on the generative approach and the baseline approach.

Table 3.11: Confusion Matrix for classification with GAN network

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	82	0	0
	Inner Race	85	0	0
	Outer Race	92	0	0

Table 3.12: Confusion Matrix for classification without GAN network

		Predicted		
		Baseline	Inner Race	Outer Race
Real	Baseline	61	27	0
	Inner Race	40	43	0
	Outer Race	39	49	0

The evolution of the cost of the loss function for both the generator and the discriminator is illustrated in figures 3.1 and 3.2.

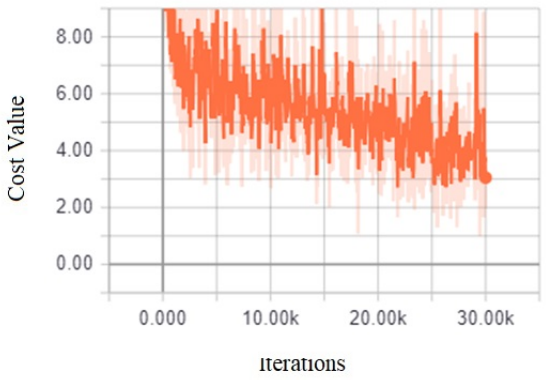

Figure 3.7: Spectrogram Discriminators loss with the CWR spectrograms as training images.

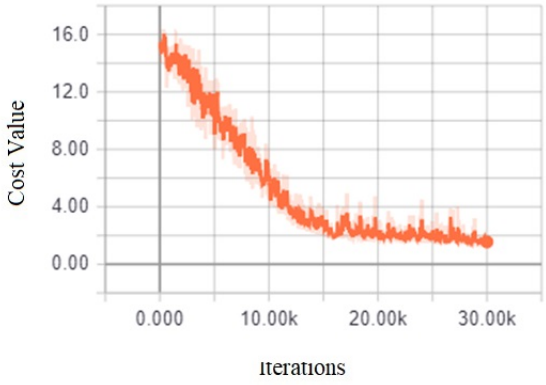

Figure 3.8: Spectrograms Generator loss with the CWR spectrograms as training images.

Chapter 4

Result Analysis

Firstly, RMSprop is not present in the result section because it had an intrinsic tendency to drive the model to divergence within the first 100 iterations. This meant it was dropped early on in the tuning stages.With this in mind it's important to note that the network had a strong tendency towards divergence, even when a total mode collapsed -generating completely grey images- was averted.
As shown in tables 3.1, 3.1.2, 3.7, 3.2, 3.10. the non generative approach was consistently better than the one proposed, this suggests issues with the generative network.
Confusion Matrices and -in less obvious manner- metrics, show a strong tendency to mode collapse, even when apparently different images were outputted for each class. In some cases, the metrics would show considerable differences between one experiment and the next, *ceteris paribus*. In this cases the confusion matrix showed that the results from classification had switched from diagnosing the whole test set as a faulty state to classifying the test set as baseline. This two can be attributed to mode collapse, were the seed or the random initialisation of the variables switched the mode into which the networked collapsed i.e. the network learned to generate all images from one domain more favourable for metrics once and one less favourable on the next depending on randomly initialised values.
The amount of Iterations was always limited by the tendency to divergence, in a much higher degree than equipment availability. yet, some loss results suggest that in some cases the network may have benefited from a longer training session.
The rather simple design of the Convolutional Neural Network proved not to be a limiting factor in the combined experiments. But had been the GAN results better, the CNN's simplicity might have been an issue.
Mini Batch Discrimination, identity loss and the use of Batch normalisation over Instance normalisation, all though proven effective for [19] and [14]. Resulted in no significant improvements.
As with many other GAN networks there was no clear co-relation between loss values and image quality, even less so between loss value and classification accuracy down the line. The use of grey scale images instead of RGB might have introduced additional complications to the network. all though in theory it reduces the complexity of the model, it is possible that it favoured completely grey images. Also it is possible that reducing the ratio of layers to labels, made the filters to simple for the information.

The proportion Amongst the different losses that compose both the generator's and the discriminator's complete loss were key factors in convergence. On the other hand this role implied little to no space to use them to improve classification accuracy. Fine tuning might improve this limitation but it implies unreasonable amounts of runs, considering the amount of loss components.
The selection of datasets and their categorisation implied the generative network had no abstraction on higher levels. The use of a single dataset gives no parameter to avoid over-fitting. The inclusion of baseline images from the allegedly unlabelled dataset proved no advantage in this respect. The use of a 3rd dataset would most likely reduce over-fitting on high levels of abstraction, and might improve results.
As [8] mentions attempting to diagnose without extracted features is an extension of the scope of the model. A less ambitious approach could include relevant features, diminishing the autonomy of the model but improving results.
Non Generative results might be explained by the detection of higher frequencies from the faults, without differentiation in between them, only to the baseline ones. this is congruent with the results shown.

Chapter 5

Discussion & Future Works

From the analysis is clear that the proposed methods is not an effective way of accomplishing the objective of an effective diagnostic of vibration signals in limited data situation through the use of deep learning generative models.
The reasons might be partly explained on result analysis training and decisions made on parameters. It is this works conclusion though, that the main reason for the model not working goes beyond parameters and training methods.
First of all, vibration analysis for bearing is based on the fundamental frequencies introduced in section 1.1.2. This frequencies are dependant on bearing characteristics and , all though this was a known challenge of the model, it played a role in it incapacity to perform due to negative interactions with the other main complications. Moreover StarGAN's proposed method for image to image translation might be more *disguise* than *translation*.
As illustrated in 5.1 the generator does not consider the input label of the image which is

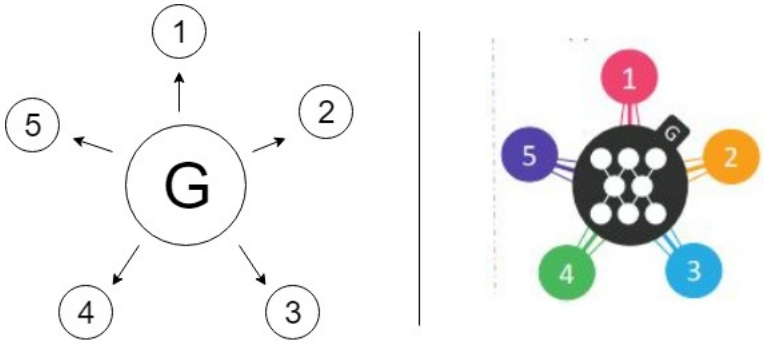

Figure 5.1: On the left this works comprehension of StarGAN, on the right the author conceptualisation of StarGAN

translating, it merely looks for the features to translate and *pastes* some features from the target domain on top of the existing features. In StarGan this can be seen in some pictures where a change in the expression domain (angry,sad,happy, etc.) changes eye colour.
StarGAN translation feature is heavily based off of cycleGAN, and in [19] it is mentioned that all though it has good results in colours and textures, it performs badly when shaped must be changed.

Faces are great candidates for image translation, lots of small distinctive features. this is an advantage for cyclic image translation. On the other hand Scalograms and spectrograms contain information across the whole image, and diagnostics often rely on the relationship between different sections of the images.
Cyclic loss -fundamental for both Stargan and Cyclegan- aims to achieve a linear resemblance between the generated images and, as specified above, for an effective diagnose, requires a relation between certain parameters underlying in each image. For the method to work as it is expected to, the relationship drawn between domains must be of a proportional nature whilst with cycle loss is one of locations.
The proposed model required of the network to discover patterns *within* the image and replicate such internal patterns on images from the new set.

Training methods can have a significant impact on results, but not event the best training method can make up what is a fundamental concept error.
For future works the author does not discard the possible application of image-to-image translation networks to bearing diagnosis, but this works conclusion is that cycle loss based image-to-image translation is not an effective method.

Conclusion

In this work, a novel method for improving fault diagnosis in scenarios with little data available was studied.
The proposed method consisted in the use of an image-translation GAN in order to obtain theoretical fault signals and use them to subsequently train a CNN.
Generative Networks such as StarGAN from [1], CycleGAN from [19] and others such as [6], where studied to design a generative network.For the elaboration of a classificator, CNN networks such as those presented by [9] and [18] where studied.
Bearing datasets from the Society for Machine Failure Prevention technology and from the Case western Reserve University were used. The data was categorised and filtered, leaving as domain the fault state present in both sets. The wavelet and Short Time Fourier Transform were applied to the vibration signals to make images, scalograms and spectrograms respectively.
Networks were trained using both types of images and in both directions (MFPT to CWR and vice-versa). Performance was measured using Confusion Matrix, accuracy, precision, sensitivity, specificity and F-1 Score indexes.

The general objective is achieved only partially, the proposed method is developed and evaluated, yet the results show effective classification is not achieved by it. on the specific objectives, both a Generative adversarial network and a Convolutional neural network are designed and implemented, different parameters are tested for effectiveness and the novel method is compared to a simpler one. Yet again the GAN network falls far behind it's expected performance.

The Convolutional neural network proves to have an acceptable performance despite it's simple model, and scalograms prove to be much more effective in classification than spectrograms, all though neither proved to work with the generative method.
The best performance achieved by the network was 0.35 for class accuracy and 0.69 for Fault State accuracy, which corresponds to a total model collapse on a favourable class for statistics.

The parameters which can help improve the network are studied thoroughly in chapter four, the irreconcilable tendency for fast divergence of the RMSProp optimiser, the diagnosis of the GAN as the main issue with the method, iterations are proposed as an improvable aspect, as well as the colour pallet for the images. Mini batch discrimination, identity loss and batch normalisation are all shown to be ineffective in improving results. Fine tuning

of loss weights is pointed out as an improvable aspects yet most-likely it will prove to be inefficient in regards of the $\frac{TimeNeeded}{marginal\mathrm{di}ference}$ ratio.

The authors conclusions on the application of image-to-image translation are exposed in chapter five, where the proposal to not pursue efforts to use Cycle loss translation but rather other image transformation methods or similar networks with losses capable of representing proportion over location.

Chapter 6

Bibliography

[1] Y. Choi, M. Choi, M. Kim, J. Ha, S. Kim, and J. Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image- translation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8789–8797, 2018.

[2] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *NIPS 2016)*, 2016.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Wade-Farley, A. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *ArXiv 2014)*, 2014.

[4] C. Gueder Soares and F López Peña. *Developments in Maritime Transportation and Exploitation of Sea Resources*. CRC press, Francis Group, London, 2014.

[5] A. Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA95472, United States, 2017.

[6] P. Isola, J.-Y. Zhu, and A. Efros. Image-to-image translation with. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] S. Khan and T. Yairi. A review on the application of deep learning in system health management. *Mechanical Systems and Signal Processing*, 107:241–265, 2018.

[8] D. Lee, V. Siu, R. Cruz, and C. Yetman. Convolutional neural net and bearing fault analysis. *International Conference on Data Mining*, 2016.

[9] D. Lee, V. Siu, R. Cruz, and C.0 Yetman. Convolutional neural net and bearing fault analysis. *International Conference on Data Mining*, 2016.

[10] H. Li, F. Xu, H. Liu, and x. Zhang. Incipient fault information determination for rolling element bearing based on synchronous averaging reassigned wavelet scalogram. *Measurement, Elsevier*, 2014.

[11] R. B. Randall. *Developments in Maritime Transportation and Exploitation of Sea Resources*. John Wiley & Sons, ltd, The Atrium, Southern Gate, Chichester, West Sussex,

PO19 8SQ, United Kingdom, 2011.

[12] Z. Ren, A. Zhou, E. Chunhui, M. Gong, B. Li, and B. Wen. Crack fault diagnosis of rotor systems using wavelet transforms. *Computers and Electrical Engineering*, 45:33–41, 2015.

[13] D. Rumelhart, G. Hinton, and R. William. Learning representations by back-propagating error. *nature*, 323:533,536, 1986.

[14] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A Radford, and X. Chen. Improved techniques for training gans. *arXiv:1606.03498 [cs.LG]*, 2016.

[15] W. J. Staszewski. Wavelet based compression and feature selection for vibration analysis. *Journal of Sound and Vibrations*, 211:735–760, 1998.

[16] Society For Machinery Failure Prevention Technology. Society for machinery failure prevention technology fault datasets.

[17] Case Western Reserve University. Case western reserve university bearing data center website.

[18] S. Zhang, S. Zhang, Wang B., and T. bB. Habetler. Machine learning algorithms for bearing fault diagnostics, a comprehensive review. *ArXiv 2019)*, 2019.

[19] J.-Y. Zhu, T. Park, i. Philip, and A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *International Conference for Computer Vision*, 2017.

Chapter 7

Annex

7.1 GAN

```
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  2 20:06:30 2018

@author: jp
"""
import tensorflow as tf
from data_loaderC import *
from set import set_
from modelD import *
import glob
import logging
from scipy.misc import imsave
import time
import os
import imageio

for handler in logging.root.handlers[:]:
    logging.root.removeHandler(handler)
logging.basicConfig(filename='SGlog.txt',format='%(levelname)s:%(message)s',
level=logging.ERROR)

class storgon:
    def __init__(self,set_):
        self.set=set_
        self.DATA={}
        self.glr=self.set['g_lr']
        self.dlr=self.set['d_lr']
        self.z=[]
        self.mode=self.set['mode']

    def testline(self):
        lt=self.dataloader(self.set['test_im'],'test_c')
        self.inputs=input_fn(lt[0],lt[1],self.set,self.DATA,lt[2],lt[3])
        print(len(lt[0]))
        self.size=len(lt[0])

    def pipeline(self):
        l1=self.dataloader(self.set['DAT1_image_dir'],'c_a1')
        #l2=self.dataloader(self.set['DAT2_image_dir'],'c2_a1')
        #self.size=len(l2[0]+l1[0])

#self.inputs=input_fn(l1[0]+l2[0],l1[1]+l2[1],self.set,self.DATA,l1[2]+l2[2],l1[3]+l
2[3])
        self.size=len(l1[0])
        print(len(l1[0]),len(l1[1]))
        self.inputs=input_fn(l1[0],l1[1],self.set,self.DATA,l1[2],l1[3])
```

```
    def dataloader(self,path,dims):#(DAT=dictionarios para el
dataset,set_=configuraciones, dims= string con las categorias del dataset)
        imgs=glob.glob(path)
        lbls=[]
        onehots=[]
        lbl_list=self.set[dims]
        Z=[]
        for a in imgs:
            for b in range(len(lbl_list)):
                if lbl_list[b] in a:
                    lbls.append(b)
        for a in lbls:
            OH=[]
            for b in range(len(lbl_list)):
                OH.append(0)
            OH[a]=1
            onehots.append(OH)
            Z.append(0)

        return [imgs,lbls,onehots,Z]
        #return input_fn(imgs,lbls,self.set,DAT,onehots,Z)



    def build_model(self):
        #onehot to trg

        self.global_step= tf.Variable(0, name="global_step", trainable=False)
        with tf.variable_scope('alpha',reuse=tf.AUTO_REUSE):

self.alpha=tf.Variable(tf.random_uniform([self.set['batch_size'],1,1,1],maxval=0.99)
)
        with tf.variable_scope('model',reuse=tf.AUTO_REUSE):
            self.CTRG=target_label(self.inputs['DS'],self.inputs['labels'])
            if self.mode=='test':
                self.CTRG=tf.reshape(self.target,[8,-1])
            self.pre_G=gen_pre_input(self.inputs['images'],self.CTRG)
            self.G=gen_setup(self.pre_G)
            if np.random.randint(1,4)==3:
                self.G=gaussian(self.G)


#self.Gc=gen_setup(gen_pre_input(self.G,tf.reshape(self.set['target3'],[8,-1])))
            if self.mode != 'test':
                self.Gc=gen_setup(gen_pre_input(self.G,self.inputs['onehot']))
            self.Fsrc,self.Fcls=dis_setup(self.G)
            self.Rsrc,self.Rcls=dis_setup(self.inputs['images'])
```

```
            self.gp_hat=self.alpha*self.inputs['images']+(1-self.alpha)*self.G
            self.Gp_src,_=dis_setup(self.gp_hat)
            self.clasif=tf.reduce_mean(self.Fcls)
    def losscalc(self):
        #disc real

LD_real=tf.squared_difference(tf.reshape(tf.reduce_mean(tf.reduce_mean(self.Rsrc,2),
1),[8]),[0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9])
        #disc Fake

LD_fake=tf.squared_difference(tf.reshape(tf.reduce_mean(tf.reduce_mean(self.Fsrc,2),
1),[8]),[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])


LD_cls=tf.nn.softmax_cross_entropy_with_logits_v2(labels=self.inputs['onehot'],logit
s=self.Rcls)
        #generator1

LG_src=tf.reduce_mean(tf.squared_difference(tf.reshape(tf.reduce_mean(tf.reduce_mean
(self.Fsrc,2),1),[8]),[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]))


LG_cls=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=self.CTRG,lo
gits=self.Fcls))

        LG_cyc=tf.reduce_mean(tf.abs(self.inputs['images']-self.Gc))

        LG_id=tf.reduce_mean(tf.abs(self.inputs['images']-self.G))
        LG_mb=minibatch(self.G)


        gloss=LG_src+LG_cls*10+LG_cyc*5
        dloss=LD_real*2+LD_fake+10*LD_cls

self.ldtuple=(tf.reduce_mean(LD_fake),tf.reduce_mean(LD_real),tf.reduce_mean(LD_cls)
)
        self.lgtuple=(LG_cls,LG_src)

        self.goptimizer=tf.train.GradientDescentOptimizer(self.glr,name='gop')
        self.doptimizer=tf.train.GradientDescentOptimizer(self.dlr,name='dop')
       #
self.goptimizer=tf.train.AdamOptimizer(self.glr,beta1=0.5,beta2=0.99,name='gop')
       #
self.doptimizer=tf.train.AdamOptimizer(self.dlr,beta1=0.5,beta2=0.99,name='dop')
        self.model_vars = tf.trainable_variables()
```

```
    d_vars = [var for var in self.model_vars if 'dis' in var.name]

    g_vars = [var for var in self.model_vars if 'gen' in var.name]

   # a_vars = [var for var in self.model_vars if 'alpha' in var.name]


    with tf.variable_scope('op',reuse=tf.AUTO_REUSE):
        self.dtrainer = self.doptimizer.minimize(dloss, var_list=d_vars)

        self.gtrainer =self.goptimizer.minimize(gloss, var_list=g_vars)
        #self.atrainer = self.aopti.minimize(self.Gp_src, var_list=a_vars)

    self.gloss_sum=tf.summary.scalar("gloss", gloss)
    self.LGcyc=tf.summary.scalar("LGcyc", LG_cyc)
    self.LGsrc=tf.summary.scalar("LGsrc", LG_src)
    self.LGcls=tf.summary.scalar("LGclass", LG_cls)
    self.lgmb=tf.summary.scalar("LGmb", 0)

    self.dloss_sum=tf.summary.scalar("dloss", tf.reduce_mean(dloss))
    self.LDreal=tf.summary.scalar("LDreal", tf.reduce_mean(LD_real))
    self.LDfake=tf.summary.scalar("LDfake", tf.reduce_mean(LD_fake))
    self.LDcls=tf.summary.scalar("LDclass", tf.reduce_mean(LD_cls))

def DEBUG(self):

    self.build_model()
    self.losscalc()
    self.saver=tf.train.Saver()
    init = (tf.global_variables_initializer(),tf.local_variables_initializer())
    if not os.path.exists(self.set['sample_dir']):
        os.makedirs(self.set['sample_dir'])
    if not os.path.exists(self.set['model_save_dir']):
        os.makedirs(self.set['model_save_dir'])
    if not os.path.exists(self.set['result_dir']):
        os.makedirs(self.set['result_dir'])
    if not os.path.exists(self.set['log_dir']):
        os.makedirs(self.set['log_dir'])



    with tf.Session() as sess:
        sess.run(init)
        sess.run(self.inputs['iterator_init_op'])
        if self.set['mode']=='restore':
            print('shit')
        writer=tf.summary.FileWriter(self.set['log_dir'])
        timer=time.time()
```

```
            timex=timer-time.time()
            batchcount=0
            adams=tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, scope='op')
            adams_init=(tf.variables_initializer(adams))
            sess.graph.finalize()
            for epoch in range(0,self.set['num_iters']):
                if self.size-batchcount<20:
                        print(batchcount)
                        sess.run(self.inputs['iterator_init_op'])

                        batchcount=0

                sess.run(adams_init)
                timex=timer-time.time()
                #logging.info(timex)
                #logging.info(str(epoch))
                if epoch%1000==0:

disc,gen,r,s=sess.run([self.ldtuple,self.lgtuple,self.inputs['onehot'],self.CTRG])
                    logging.error('LD:'+str(disc[0])+str(disc[1])+str(disc[2])+'
   LG:'+str(gen[0])+str(gen[1]))
                    batchcount+=8
                if epoch%500==0:


                    timex=timer-time.time()

                    #logging.info(timex)

                    print('epoch:',epoch)

output,ctrg,onehots,Gc,image=sess.run([self.G,self.CTRG,self.inputs['onehot'],self.G
c,self.inputs['images']])
                    batchcount+=8
                    i=0
                    if not os.path.exists(self.set['sample_dir']+'/'+str(epoch)):
                        os.makedirs(self.set['sample_dir']+'/'+str(epoch))
                    for out in output:

imageio.imwrite(self.set['sample_dir']+'/'+str(epoch)+'/G_'+str(i)+'.jpg',out)

imageio.imwrite(self.set['sample_dir']+'/'+str(epoch)+'/Gc_'+str(i)+'.jpg',Gc[i])

imageio.imwrite(self.set['sample_dir']+'/'+str(epoch)+'/IMAGE_'+str(i)+'.jpg',image[
i])


                        i+=1
```

```
                if epoch>self.set['lr_update_step']:

self.glr=self.set['g_lr']*(1-(epoch-self.set['lr_update_step'])/(self.set['lr_update
_step']))

self.dlr=self.set['d_lr']*(1-(epoch-self.set['lr_update_step'])/(self.set['lr_update
_step']))



                if epoch%self.set['n_critic']==0:
                    #optimize G

_,summary,lgsrc,lgcyc,lgcls,lgmb=sess.run([self.gtrainer,self.gloss_sum,self.LGsrc,s
elf.LGcyc,self.LGcls,self.lgmb])
                    #_=sess.run(self.gtrainer)

                    writer.add_summary(summary,epoch)
                    writer.add_summary(lgsrc,epoch)
                    writer.add_summary(lgcyc,epoch)
                    writer.add_summary(lgcls,epoch)
                    writer.add_summary(lgmb,epoch)
                    batchcount+=8

                if self.size-batchcount<20:
                    sess.run(self.inputs['iterator_init_op'])
                    batchcount=0
                if epoch+1%100==0:

self.saver.save(sess,self.set['model_save_dir'],global_step=epoch)

                #optimize D

_,summary,ldreal,ldfake,ldcls=sess.run([self.dtrainer,self.dloss_sum,self.LDreal,sel
f.LDfake,self.LDcls])
                #_=sess.run(self.dtrainer)
                writer.add_summary(summary,epoch)
                writer.add_summary(ldreal,epoch)
                writer.add_summary(ldfake,epoch)
                writer.add_summary(ldcls,epoch)
                batchcount+=8
                self.epoch=epoch


            writer.add_graph(sess.graph)
```

```
            output=sess.run(self.G)
            i=0
            for out in output:
                imageio.imwrite(self.set['sample_dir']+'/LG_'+str(i)+'.jpg',out)
                i+=1
            self.saver.save(sess,self.set['model_save_dir'])

    def test(self,num):
        if num==1:
              self.target=self.set['target1'] #IR
        elif num==2:
              self.target=self.set['target2'] #OR_
        elif num==3:
              self.target=self.set['target3'] #N
        self.build_model()
        self.saver=tf.train.Saver()
        self.class_d=0
        classcounter=0
        batchcount=0

        init = (tf.global_variables_initializer(),tf.local_variables_initializer())

        with tf.Session() as sess:
            sess.run(init)
            sess.run(self.inputs['iterator_init_op'])
            i=0
            Iter=0
            if not os.path.exists(self.set['result_dir']+'/'+str(num)):
                  os.makedirs(self.set['result_dir']+'/'+str(num))
            self.saver.restore(sess,self.set['model_save_dir'])
            while 100>Iter:
                while (self.size-batchcount)>8:
                      output,clasif=sess.run([self.G,self.clasif])
                      self.class_d+=clasif
                      classcounter+=1
                      batchcount+=8
                      for out in output:

imageio.imwrite(self.set['result_dir']+'/'+str(num)+'/'+str(i)+'.jpg',out)
                           i+=1

                Iter+=1
            self.class__d=self.class_d/classcounter
            #batchcount=0tf.reset_default_graph()

            #for epoch in range(self.set['out_imgs']):
             #   output=sess.run(self.G)
              #  i=0
```

```
                # for out in output:
                 #
imsave(self.set['result_dir']+'/'+str(epoch)+'_'+str(i)+'.jpg',out)
                  #   i+=1
                 #batchcount+=8
                 #if self.size-batchcount<8:
                 #    sess.run(k.inputs['iterator_init_op'])
                 #    batchcount=0



"""
    def train(self):

        self.build_model()
        self.losscalc()
        self.saver=tf.train.Saver()
        init = (tf.global_variables_initializer(),tf.local_variables_initializer())
        if not os.path.exists(self.set['sample_dir']):
            os.makedirs(self.set['sample_dir'])
        if not os.path.exists(self.set['model_save_dir']):
            os.makedirs(self.set['model_save_dir'])
        if not os.path.exists(self.set['result_dir']):
            os.makedirs(self.set['result_dir'])
        if not os.path.exists(self.set['log_dir']):
            os.makedirs(self.set['log_dir'])



        with tf.Session() as sess:
            sess.run(init)
            sess.run(self.inputs['iterator_init_op'])

            if self.set['mode']=='restore':
                print('shit')
            writer=tf.summary.FileWriter(self.set['log_dir'])
            timer=time.time()
            timex=timer-time.time()
            logging.info(timex)
            batchcount=0
            for epoch in range(sess.run(self.global_step),self.set['num_iters']):
                timex=timer-time.time()
                #logging.info(timex)
                #logging.info(str(epoch))
                if epoch%100==0:
                    timex=timer-time.time()
                    logging.info(str(epoch))
                    #logging.info(timex)
```

```
                    print('epoch:',epoch)

output,ctrg,onehots,Gc,image=sess.run([self.G,self.CTRG,self.inputs['onehot'],self.G
c,self.input['images']])
                    batchcount+=8
                    i=0
                    if not os.path.exists(self.set['result_dir']+'/'+str(epoch)):
                        os.makedirs(self.set['result_dir']+'/'+str(epoch))
                    for out in output:

imageio.imwrite(self.set['result_dir']+'/'+str(epoch)+'/G_'+str(i)+'.jpg',out)

imageio.imwrite(self.set['result_dir']+'/'+str(epoch)+'/GC_'+str(i)+'.jpg',Gc[i])


                        logging.info('ctrg,hots')
                        logging.info(ctrg[i])
                        logging.info(onehots[i])
                        i+=1

self.saver.save(sess,self.set['model_save_dir'],global_step=epoch)

                if epoch>self.set['lr_update_step']:

self.glr=self.set['g_lr']*(1-(epoch-self.set['lr_update_step'])/(self.set['lr_update
_step']))

self.dlr=self.set['d_lr']*(1-(epoch-self.set['lr_update_step'])/(self.set['lr_update
_step']))

                if self.size-batchcount<8:
                    sess.run(k.inputs['iterator_init_op'])
                    batchcount=0
                #optimize G
                _,summary=sess.run([self.gtrainer,self.gloss_sum])
                writer.add_summary(summary,epoch)
                batchcount+=8
                if self.size-batchcount<8:
                    sess.run(k.inputs['iterator_init_op'])
                    batchcount=0
                #optimize D
                _,summary=sess.run([self.dtrainer,self.dloss_sum])
                writer.add_summary(summary,epoch)
                batchcount+=8
                self.epoch=epoch
            writer.add_graph(sess.graph)
            output=sess.run(self.G)
```

```
            i=0
            for out in output:
                imageio.imwrite(self.set['result_dir']+'/'+str(i)+'.jpg',out)
                i+=1
"""




tf.reset_default_graph()
logging.error('storgon has began')
print('a')
timer=time.time()
timex=timer-time.time()
logging.info(timex)
k=storgon(set_)
k.pipeline()


timex=timer-time.time()
#logging.info(timex)

k.DEBUG()

tf.reset_default_graph()
IR_t=storgon(set_)
IR_t.mode="test"
IR_t.testline()
IR_test=IR_t.test(1)

tf.reset_default_graph()
OR_t=storgon(set_)
OR_t.mode="test"
OR_t.testline()
OR_test=OR_t.test(2)

tf.reset_default_graph()
N_t=storgon(set_)
N_t.mode="test"
N_t.testline()
N_test=N_t.test(3)


timex=timer-time.time()
logging.info('END')
logging.info(timex)
```

```
"""
k.build_model()
init = (tf.global_variables_initializer(),tf.local_variables_initializer())
N=[]
R=[]
with tf.Session() as sess:
    sess.run(init)
    sess.run(k.inputs['iterator_init_op'])

#output,ctrg,onehots,Gc,image=sess.run([k.G,k.CTRG,k.inputs['onehot'],k.Gc,k.inputs[
'images']])
    output=sess.run(k.Gc)
    print('done')
"""
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 12 18:10:28 2018

@author: jp
"""

import tensorflow as tf
import numpy as np
from set import set_
import logging

def Resnet(inputing, dim, name="res"):
    with tf.variable_scope(name):
        outres=tf.contrib.layers.conv2d(inputing,dim,3 , 1, padding='SAME',
activation_fn=None)
        outres=tf.contrib.layers.instance_norm(outres)
        outres=tf.nn.relu(outres)
        outres=tf.contrib.layers.conv2d(outres,dim,3, 1, padding='SAME',
activation_fn=None)
        outres=tf.contrib.layers.instance_norm(outres)
        outres=tf.nn.relu(outres+inputing)

        return (outres)


def gen_setup(input_A, name="gen"):
    with tf.variable_scope(name,reuse=tf.AUTO_REUSE):
        #k=[]
        #k.append(tf.shape(input_A))
        out=tf.contrib.layers.conv2d(input_A,set_['g_conv_dim'] ,7 , 1,
padding='SAME', activation_fn=None)
        #k.append(tf.shape(out))
        out=tf.contrib.layers.instance_norm(out)
        out=tf.nn.relu(out)
        curr_dim=set_['g_conv_dim']
        for i in range(2):
            out=tf.contrib.layers.conv2d(out,curr_dim*2,4 , 2, padding='SAME',
activation_fn=None)
            #k.append(tf.shape(out))
            out=tf.contrib.layers.instance_norm(out)
            out=tf.nn.relu(out)
            curr_dim=curr_dim*2

        for i in range(set_['g_repeat_num']):
            out=Resnet(out,curr_dim,name='res'+str(i))
            #k.append(tf.shape(out))
        #out=tf.cast(out,dtype='float32')
```

```
        for i in range(2):
            out=tf.contrib.layers.conv2d_transpose(out,int((curr_dim) / 2),4 , 2,
padding='SAME', activation_fn=None)
            #k.append(tf.shape(out))
            out=tf.contrib.layers.instance_norm(out)
            out=tf.nn.relu(out)
            curr_dim=curr_dim/2
        out=tf.contrib.layers.conv2d(out,1,7 , 1, padding='SAME',
activation_fn=tf.tanh)
        #k.append(tf.shape(out))
        return out

def dis_setup(input_B, name='dis',reuse=tf.AUTO_REUSE):
    k=[]
    with tf.variable_scope(name,reuse=tf.AUTO_REUSE):
        out=tf.contrib.layers.conv2d(input_B,set_['d_conv_dim'] ,4, 2,
padding='SAME', activation_fn=None)
        k.append(tf.shape(out))
        out=tf.nn.leaky_relu(out,alpha=0.01)
        curr_dim=set_['d_conv_dim']
        for i in range(1,set_['d_repeat_num']):
            out=tf.contrib.layers.conv2d(out,curr_dim*2 , 4, 2, padding='SAME',
activation_fn=None)
            k.append(tf.shape(out))
            out=tf.nn.leaky_relu(out,alpha=0.01)
            curr_dim=curr_dim*2
        kernel_size=int(set_['image_size']/np.power(2,set_['d_repeat_num']))
        out1=tf.contrib.layers.conv2d(out,1 ,3, 1, padding='SAME',
activation_fn=None)
        out2=tf.contrib.layers.conv2d(out,3 ,kernel_size, 1, padding='VALID',
activation_fn=None)
        out2=tf.reshape(out2,[8,3])
        k.append(tf.shape(out1))
        k.append(tf.shape(out2))
        return out1,out2

def n256n(a):
    multi=tf.constant([1,65536])
    theta1=tf.tile(a,multi)
    jj=tf.reshape(theta1,[256,256,tf.size(a)])
    return jj

def gen_pre_input(inp,OHL,name='preinput'):
    with tf.variable_scope(name):
        squares=tf.map_fn(lambda x:n256n([x]), OHL)
        squares=tf.cast(squares,dtype='float32')
        r=tf.concat([inp,squares],3)
        return r
```

```
def nrond(a,tope,n1,n0):
    b=np.random.randint(0,tope)
    while b==a:
        b=np.random.randint(0,tope)
    n=n0+n1[b]+n1[1]
    return tf.constant(n)
def nnorond(a,n1,n0):
    n=n0+n1[a]+n1[1]

    return tf.constant(n)

def lrond(a,tope,l1):
    b=np.random.randint(0,tope)
    while b==a:
        b=np.random.randint(0,tope)
    zero=[0,0]
    f=[1,0]
    n=l1[b]+zero+f
    return tf.constant(n)
def lnorond(a,l1):
    zero=[0,0]
    b=[1,0]
    n=l1[a]+zero+b
    return tf.constant(n)



def ctrgn(oh):
    n1=[[1,0],[0,1]]
    n0=[0,0,0]
    tope=2
    n3=np.random.randint(0,tope)
    n3_1=tf.constant(n3)

n=tf.cond(tf.equal(oh,n3_1),lambda:nrond(n3,tope,n1,n0),lambda:nnorond(n3,n1,n0))
#if new one hot= original retry, if not, continue
    return n


def ctrgl(oh):
    l1=[[1,0,0],[0,1,0],[0,0,1]]
    tope=3
    l2=np.random.randint(0,tope)
    l2_1=tf.constant(l2)
    n=tf.cond(tf.equal(oh,l2_1),lambda:lrond(l2,tope,l1),lambda:lnorond(l2,l1))
```

```
    return n


def target_label_null(ds,oh):
    labels=ds
    labelsun=tf.unstack(labels,num=8)
    ohsun=tf.unstack(oh,num=8)
    out=[]
    zero=tf.constant(2)
    i=0
    for t in labelsun:
        out.append(tf.cond(tf.equal(t,zero),lambda: ctrgn(ohsun[i]),lambda:
ctrgl(ohsun[i])))
        i+=1

    out=tf.concat(out,0)

    out=tf.reshape(out,[8,-1])

    return out

def PP1(t,l):
    non=t
    p=np.random.randint(0,3)
    while non==p:
        p=np.random.randint(0,3)
    return l[p]
def PP2(t,l):
    return (l[t])

def target_label(ds,oh):
    labels=ds
    labelsun=tf.unstack(labels,num=8)
    ohsun=tf.unstack(oh,num=8)
    out=[]
    zero=tf.constant(2)
    i=0
    for t in labelsun:
        eq=np.random.randint(0,3)
        eq2=tf.constant(eq)
        l1=[[1,0,0],[0,1,0],[0,0,1]]
        out.append(tf.cond(tf.equal(ohsun[i],eq2),lambda:
PP1(eq2,l1),lambda:PP2(eq,l1)))
        i+=1

    out=tf.concat(out,0)

    out=tf.reshape(out,[8,-1])
```

```
    return out

def gaussian(imgs):
      a=np.random.normal(0.0,0.1,[8,256,256,1])
      tf.convert_to_tensor(a)
      return (imgs+a)

def minibatch(a):
    a=a*1
    images=tf.unstack(a,num=8)
    i=0
    cost=0
    while i<8:
        i2=0
        while i2<8:
            if i2!=i:
                cost+=tf.abs(images[i]-images[i2])
            i2+=1
        i+=1
    return tf.reduce_mean(cost)

#def Eqtarget(oh,ctrg):
```

7.2 CNN

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 16 11:37:16 2019

@author: jp
"""
import tensorflow as tf
from sett import sett
from input_loader import *
import glob
import os
import sklearn.metrics
import numpy as np

def Model(input_1, name='model'):
    with tf.variable_scope(name,reuse=tf.AUTO_REUSE):
          out = tf.reshape(input_1, [-1, 256, 256, 1])
          out = tf.layers.conv2d(inputs=out, name='convolution_1',
                                 filters=32, kernel_size=3,
                                 padding='same', activation=tf.nn.relu)

          out = tf.layers.max_pooling2d(inputs=out, pool_size=2, strides=2)
          out = tf.layers.conv2d(inputs=out, name='convolution_2',
                                 filters=64, kernel_size=3,
                                 padding='same', activation=tf.nn.relu)
          out = tf.layers.max_pooling2d(inputs=out, pool_size=2, strides=2)
          out = tf.layers.conv2d(inputs=out, name='convolution_3',
                                 filters=64, kernel_size=3,
                                 padding='same', activation=tf.nn.relu)
          out = tf.layers.max_pooling2d(inputs=out, pool_size=2, strides=2)
          out = tf.layers.flatten(out)
          out = tf.layers.dense(inputs=out, name='layer_fc1',units=128,
activation=tf.nn.relu)
          out = tf.layers.dropout(out, rate=0.5, noise_shape=None,seed=None,
training=(True))
          out = tf.layers.dense(inputs=out, name='layer_fc_2',units=3)
          out_cls = tf.argmax(out, axis=1)

          shape=tf.shape(out)
          return shape,out,out_cls



class MLP:
    def __init__(self,sett):
        self.set=sett
        self.DATA={}
        self.mode=self.set['mode']
```

```
    def testline(self):
        lt=self.dataloader(self.set['test_im'],'test_c')

        self.inputs=input_fn(lt[0],lt[1],self.set,self.DATA)


        self.size=len(lt[0])

    def pipeline(self):
        l1=self.dataloader(self.set['image_dir'],'c_a1')
        self.size=len(l1[0])
        self.inputs=input_fn(l1[0],l1[1],self.set,self.DATA)

    def dataloader(self,path,dims):#(DAT=dictionarios para el
dataset,set_=configuraciones, dims= string con las categorias del dataset)
        imgs=glob.glob(path)
        lbls=[]
        lbl_list=self.set[dims]
        for a in imgs:
            for b in range(len(lbl_list)):
                if lbl_list[b] in a:
                    lbls.append(b)
                #else:
                #    print (a)


        return [imgs,lbls]
        #return input_fn(imgs,lbls,self.set,DAT,onehots,Z)
    def build_Model(self):
          self.shape,self.mod,self.out_cls=Model(self.inputs['images'])
          self.accuracy=tf.metrics.accuracy(self.inputs['labels'], self.out_cls)
    def loss_calc(self):
        cross =
tf.nn.sparse_softmax_cross_entropy_with_logits(labels=self.inputs['labels'],logits=s
elf.mod)
        self.loss = tf.reduce_mean(cross)
        with tf.variable_scope('op',reuse=tf.AUTO_REUSE):
              self.optimizer = tf.train.AdamOptimizer(learning_rate=self.set["lr"])

              self.model_vars = tf.trainable_variables()
              self.trainer=self.optimizer.minimize(self.loss,
var_list=self.model_vars)

    def train(self):
          self.build_Model()
          self.loss_calc()
          self.saver=tf.train.Saver()
```

```
          init =
(tf.global_variables_initializer(),tf.local_variables_initializer())
          with tf.Session() as sess:
                sess.run(self.inputs['iterator_init_op'])
                sess.run(init)
                batchcount=0
                for epoch in range(0,self.set['num_iters']):
                      sess.run(self.trainer)
                      batchcount+=8
                      if batchcount+8>self.size:
                            sess.run(self.inputs['iterator_init_op'])
                self.metric=sess.run(self.accuracy)
                if not os.path.exists(self.set['model_save_dir']):
                      os.makedirs(self.set['model_save_dir'])
                self.saver.save(sess,self.set['model_save_dir'])
    def Predict(self):
          self.out_labels=[]
          self.out_pred=[]
          self.build_Model()
          self.loss_calc()
          self.saver=tf.train.Saver()
          init =
(tf.global_variables_initializer(),tf.local_variables_initializer())
          with tf.Session() as sess:
                sess.run(init)
                self.saver.restore(sess,self.set['model_save_dir'])
                sess.run(self.inputs['iterator_init_op'])
                batchcount=0
                while self.size>batchcount+8:

label,prediction=sess.run([self.inputs['labels'],self.out_cls])
                      for a in range(0,7):
                            self.out_labels.append(label[a])
                            self.out_pred.append(prediction[a])
                      batchcount+=8
                self.metric=sess.run(self.accuracy)


print("friendly coding")
k=MLP(sett)
k.pipeline()
k.train()
print("training accuracy:")
print(k.metric)
tf.reset_default_graph()
mp=MLP(sett)
mp.testline()
mp.Predict()
```

```
print (mp.metric)
print(sklearn.metrics.confusion_matrix(mp.out_labels,mp.out_pred))
i=0
l0_0=0
l0_1=0
l0_2=0
l1_0=0
l1_1=0
l1_2=0
l2_0=0
l2_1=0
l2_2=0


for a in mp.out_labels:
      if a==0:
            if mp.out_pred[i]==0:
                  l0_0+=1
            elif mp.out_pred[i]==1:
                  l0_1+=1
            elif mp.out_pred[i]==2:
                  l0_2+=1
      elif a==1:
            if mp.out_pred[i]==0:
                  l1_0+=1
            elif mp.out_pred[i]==1:
                  l1_1+=1
            elif mp.out_pred[i]==2:
                  l1_2+=1
      elif a==2:
            if mp.out_pred[i]==0:
                  l2_0+=1
            elif mp.out_pred[i]==1:
                  l2_1+=1
            elif mp.out_pred[i]==2:
                  l2_2+=1
      i+=1



tots=l0_0+l0_1+l0_2+l1_0+l1_1+l1_2+l2_0+l2_1+l2_2
c_accuracy=(l0_0+l1_1+l2_2)/(tots)
FS_accuracy=(l0_0+l1_1+l2_2+l1_0+l0_1)/tots

sensitivity=(l2_2)/(l2_0+l2_1+l2_2)
specificity=(l1_1+l0_0)/(l1_1+l0_0+l0_2+l1_2)
f_1=(2*l2_2)/(2*l2_2+l0_2+l1_2+l2_1+l2_0)
precision=(l2_2)/(l2_2+l1_2+l0_2)
```

```
print('class:');print(str(c_accuracy));print('Fault:');print(str(FS_accuracy));print
('precision:');print(str(precision));print('sensitivity');print(str(sensitivity));pr
int('specificity');print(str(specificity));print('F1');print(str(f_1))


"""
n=k.build_Model()
init = (tf.global_variables_initializer(),tf.local_variables_initializer())+'/n'+
with tf.Session() as sess:
      sess.run(init)
      sess.run(k.inputs['iterator_init_op'])
      b=sess.run()
print (b)
"""
```