Anomaly Detection in Aerodynamic Data

Raphaël Kanapitsas

October 2021

Introduction

The following describes my Internship at Airbus, taking place from July to December 2021. I am in the Aircraft Aerodynamics team, on the look-out for Anomalies in generated aerodynamic data. This report will be roughly chronological, and is composed of three main parts :

1. An exposition of the company, the team and the problem at hand.
2. A theoretical view of how Anomaly Detection is defined, and an in-depth explanation of the Variational Autoencoder.
3. The steps I went through and the results I got.

A fourth and last part will offer an overview of the work that remains to be done, and of what doing this intership has taught me.

In this work, I have also tried to explain various things I've learnt, even when they're not directly linked to the goal of the internship. Some of these sections will be marked with a star ($*$), feel free to skip them.

Contents

Chapter 1

The setting and the problem

Let's start with a brief overview of the company, and progressively zoom-in on the actual subject. On the way, I'll explain some basic principles of aerodynamics, which are necessary to understand the data.

1.1 About Airbus

1.1.1 The Company

Airbus is a major company in the domain of Aircraft and Aerospace, and is the world's largest aircraft manufacturer. It was founded fifty years ago as a cooperation between France, Germany and the UK. Airbus is headquartered in Toulouse, but has many sites in Europe (France, Germany, Spain, UK) and also in the rest of the world (China, United Sates, Canada, India), which employ in total 135k people.

Airbus is comprised of :

- Commercial Aircraft (70% of revenue), which produces airliners like the medium-range A320 family, and the long-range wide-body A330 and A350.
- Defence and Space (20% of revenue), which makes the A330 MRTT (Multi Role Tanker Transport) and the A400M; as well as satellites and others.
- Airbus Helicopters (10% of revenue).

I was in the Commercial Aircraft division, at the Toulouse/Saint-Martin-du-Touch site. This location alone has 30k employees. Most airplanes are assembled here, with parts coming from France, Germany, the UK and Spain (see figure 1.1), delivered by the Beluga.

The team I am working in is IGAAT (for Aircraft Aerodynamics in Toulouse). It is composed of about fifteen people, some specializing in flight models, calibration, clinometry, icing, etc...

You will find a summary of the different teams in the appendix.

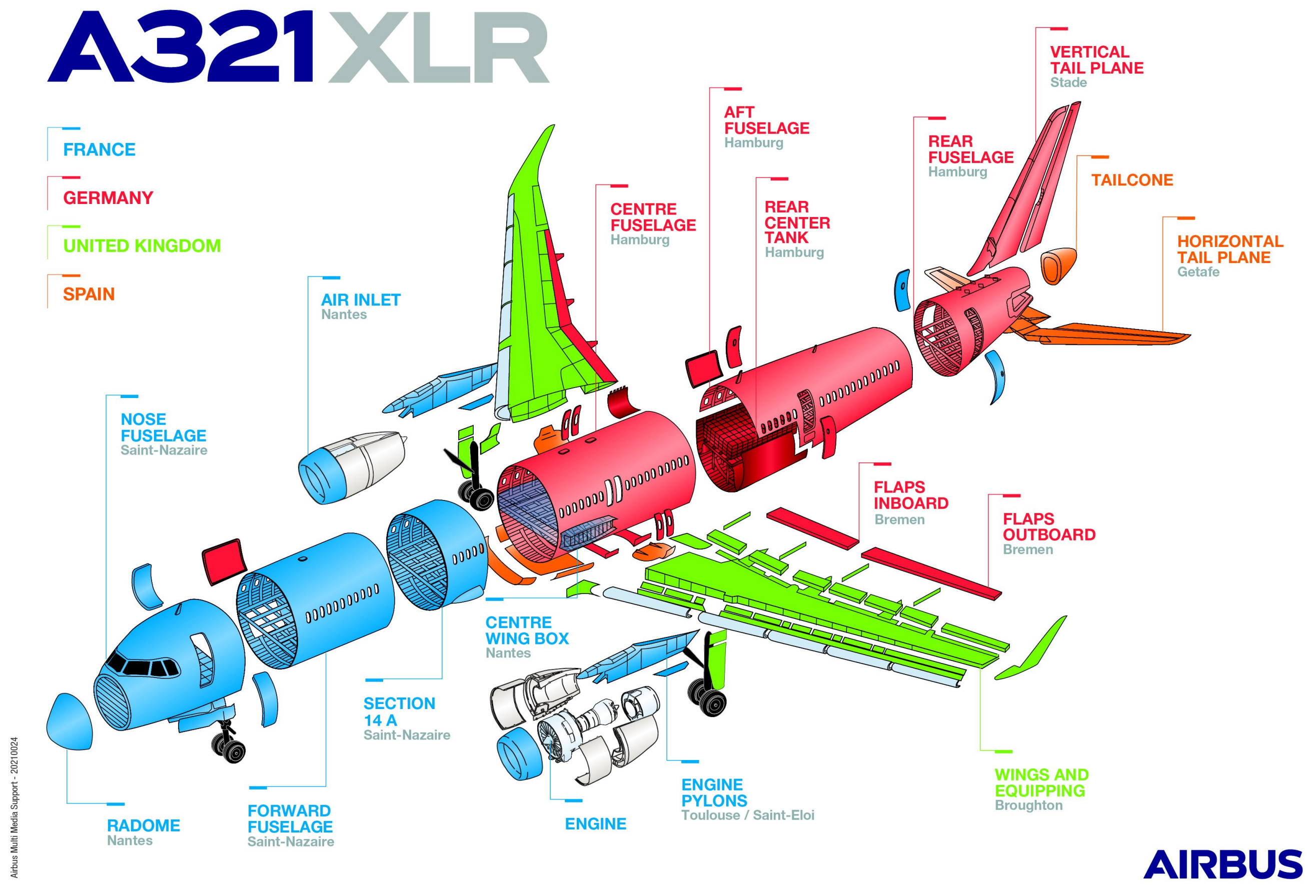


Figure 1.1: Origin of the different parts of an A321XLR

1.1.2 Development cycle

The development of an aircraft is a long and precise process, implicating many different people and know-hows. Everyone needs to be able to collaborate and progress in an organized way. That's why this process is quite strictly managed.

The old

The development cycle starts with the *Feasibility* phase. There, several options are explored (using quick and rather crude tools) until MG3 where the general shape of the aircraft is decided.

The *Concept* phase marks a more in-depth analysis and refinement of the decided shape. How-

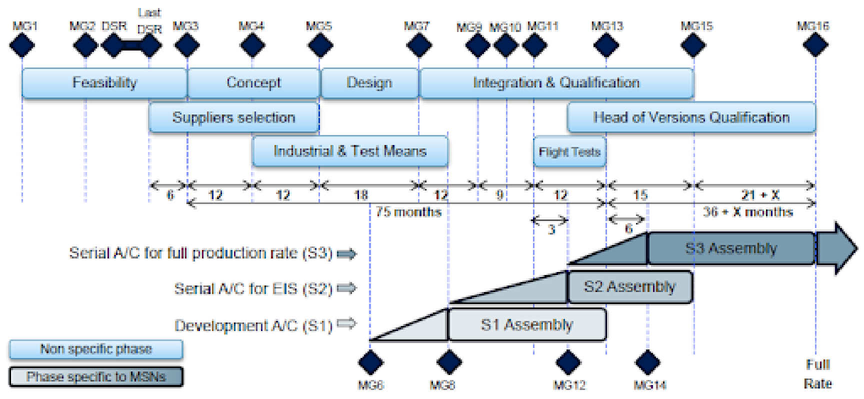


Figure 1.2: The current development cycle

ever, large modifications cannot be made, which sometimes causes issues.

The New

A new process is being created to replace the current one. It aims at two things: reducing the overall development time by doing more problem-solving at the beginning; and improving product choices by iterating more and earlier in the process, and using more powerful tools than in the current *Feasibility* phase.

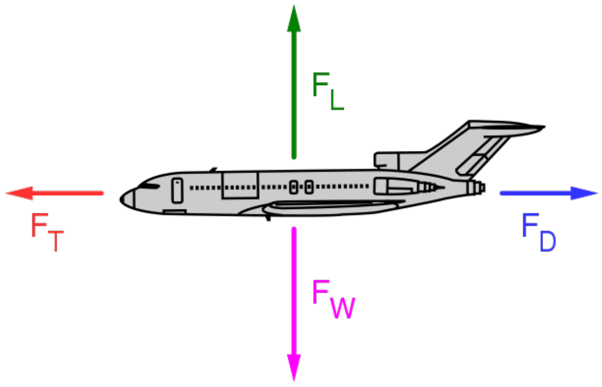
The new process is divided in two: the out-of-cycle (OoC) loop, and the actual, *in-cycle* phase. The OoC loop is where the two previous problems are solved. In contrast to the current cycle, more time is given to generate and assess multiple configurations, until the convergence to a robust candidate.

There are two phases within the OoC loop. In the first, several concepts are proposed to fit the needs. In the second, these concepts are explored using efficient (and mostly automated) tools. This enables rapid iterations. The goal is to iron-out any sub-optimal decision which could negatively affect later development, as well as exploring more of the concept space.

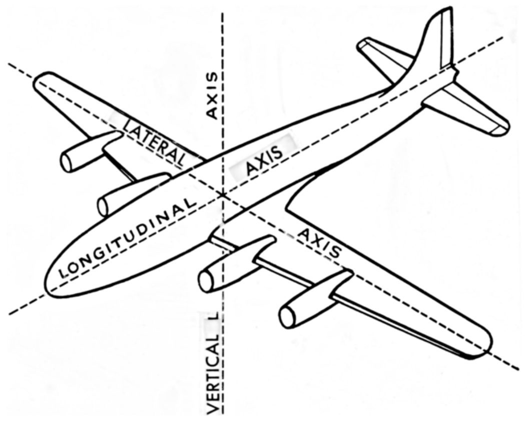
1.1.3 IGAA

The main task of IGAA (Aircraft Aerodynamics) is to build aerodynamic models (more on what this means later). There are roughly two different kinds of work :

- Creating and fitting models. First with quick tools to study the feasibility of a concept, the air-worthiness of a design, and explore which options are the best. Then, more in depth, to predict the behaviour of the airplane as well as possible.
- More specific and down-to-earth: analysis of the data from specific airplane (as in a unique



(a) Main forces acting on a flying airplane



(b) The three axis of an airplane

Figure 1.3: Main forces and axis on an airplane

serial number), with flight tests. There can also be particular problems which require the expertise of IGAA.

There are also different specialties :

- Anemometry: finding the speed and altitude of the aircraft (based pressure measurements).
- Clinometryf: finding the angles of the aircraft, relative to the airflow and the surface.
- Icing : modelling the aerodynamic influence of icing on different parts of the aircraft (especially leading edges of the wings).

1.2 Aerodynamics

1.2.1 What do we need to know?

In order to design an aircraft which satisfies some requirements (range, speed, etc), we need to be able to predict how a given aircraft will perform in the air.

Unfortunately, the physics of fluid dynamics have no known analytical solutions and their approximation are extremely computation-intensive and expensive. This is why we need simplified models, good enough in practice and (relatively) simple to use.

The existence of a solution to the Navier–Stokes equations is one of the Millennium Prize Problems

1.2.2 Basic forces and models

Four forces act on a flying aircraft (barring asymmetrical flight and turns):

- Downwards, vertically, the weight

- Upwards, normal to the wing surface, the lift
- Opposite the direction of travel, the drag
- Aligned with the fuselage, the thrust

The sum of these forces will determine the position of the aircraft at a given time.

More generally, there are three axis to consider, for both the force along them, and the rotation around them. They all pass through the center of gravity. The following table summarizes the six coefficients of interest.

Axis	Force	Force Coeff	Rotation	Moment Coeff
x	Drag	C_x	Roll	C_l
y	Side force	C_y	Pitch	C_m
z	Lift	C_z	Yaw	C_n

Forces along these axis can be expressed in a general form, with specific coefficients depending on the shape of the object.

$$F = \frac{1}{2}\rho v^2 C S_{\text{ref}}$$

Here, ρ is the density of the air, v the speed of the airflow, C the aerodynamic coefficient and S_{ref} the reference area. This expression can be used to find the forces for drag, side force and lift, using the coefficients C_x, C_y and C_z. There are similar expressions for moment coefficients.

Example of a simple model

One very important thing to know about an aircraft is when it is going to stall: at a constant speed, unsurprisingly, increasing the angle of attack α increases the lift coefficient C_z. This goes on, until a point where the airflow on the upper side of the wing is too turbulent, and the lift crumbles when further increasing α. This is the point where the aircraft *stalls.*

It turns out the first part of this phenomenon is linear. Indeed, for small values of α :

$$C_z = C_{z_\alpha}(\alpha - \alpha_0)$$

This expression can be corrected for larger values of α using a non-linearity term ΔC_z^{NL} (function of α). This constitutes a simplified part of an aerodynamic model.

$$C_z = C_{z_\alpha} \cdot (\alpha - \alpha_0) + \Delta C_z^{NL}$$

1.2.3 Performance & Handling Qualities

These two disciplines aim at characterizing aircrafts:

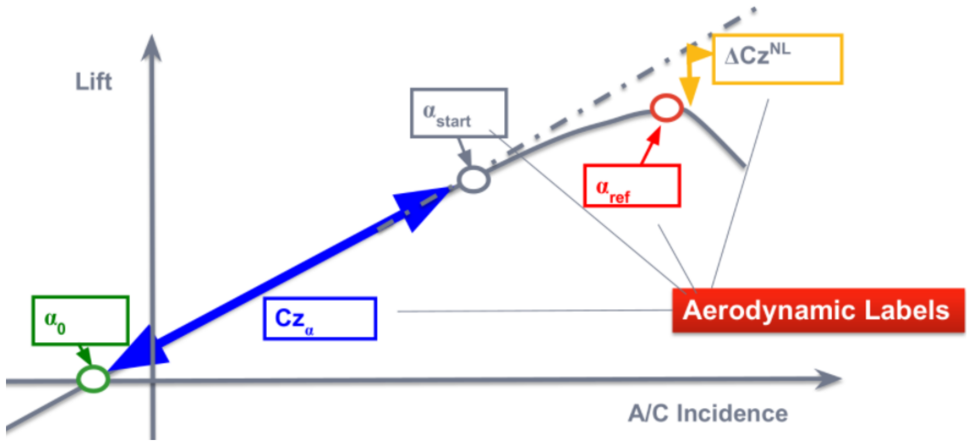


Figure 1.4: Simple model of lift with respect to α

Performance focuses mainly on official certification, for authorities like the European Aviation Safety Agency or the Federal Aviation Administration in the US. Some key parameters are fuel efficiency, climbing speed, range, take-off and landing performance,

Handling Qualities focuses on the aircraft's maneuverability. This is mostly done by the IYC team, which carefully models the effect of controls, so that flight laws can be tuned for the needs.

1.2.4 Reynolds number*

The Reynolds number is a dimensionless quantity which gives an idea of how a fluid will behave around an object, from laminar to turbulent. It is defined as :

$$\mathrm{Re} = \frac{uL}{\nu}$$

Where:

- u is the flow speed (m/s)
- L is the characteristic length (m)
- ν is the kinematic viscosity (m^2/s)

Generally, a small Reynolds number (< 1) will correspond to viscous (maybe laminar) flow, while a higher (40-1000) might lead to a vortex street, and higher still will result in a turbulent flow. For a flying aircraft, this might be around 10^8-10^9, which is definitely turbulent. For a WT test in the same atmosphere, it might be 100 times smaller, which means some phenomena won't be the same.

1.2.5 Boundary layer*

A fluid like air is viscous, and will slow down near a surface. If we consider a flow of air along a wall, and plot the air speed, it would look like something like figure 1.5.

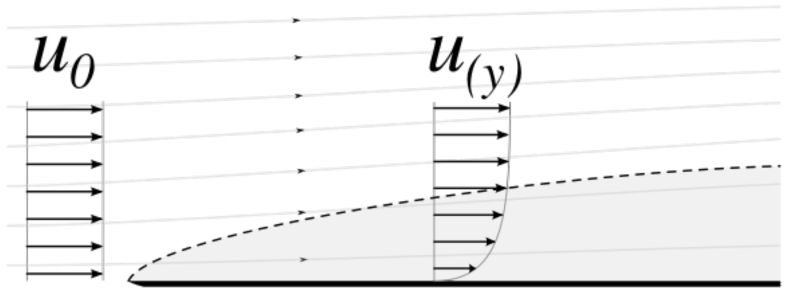


Figure 1.5: Fluid velocity and boundary layer

At a height of $x = 0$, the speed is null, and gradually increases as we go further from the wall. The area where the speed is most slowed down by the wall is called the *boundary layer*. It grows the further along the x axis you go, and might get turbulent after some point.

Even though there is no clear cut-off for this area, the concept is useful as it simplifies the aerodynamic analysis : you only have to take into account viscosity in the boundary layer, and can consider it null outside.

Boundary layer separation

If adverse pressure gradients are too high (for example at high angles of attack), the slowed-down air in the boundary layer might actually *reverse* its direction. This causes the appearance of vortices and boundary layer separation. You get higher drag and lower lift, the airplane might stall.

To remedy this, some airplanes have some small metal sheets on their wings, specifically designed to create vortices on the upper wing side. This adds energy to the boundary layer, and delays separation (the trade-off is slightly increased drag). These devices are useful only at high α, in high-lift configurations (as on figure 1.7).

1.2.6 Other models

Aerodynamic models are essential, but more is needed to have a complete picture of the aircraft. Other teams design models for the engines, loads inside the structure, systems (actuactors, primary flight controls), etc... These are integrated to model the full behaviour of the aircraft.

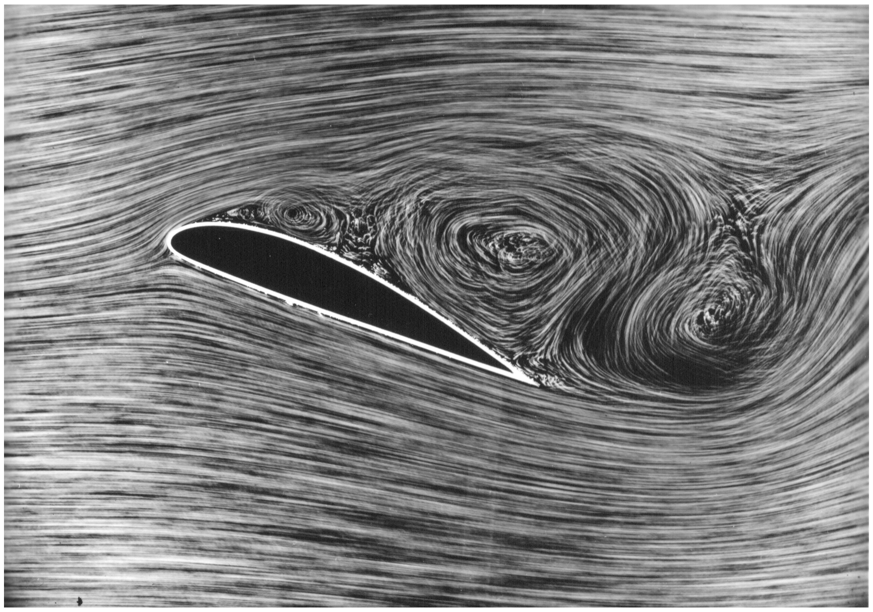


Figure 1.6: An example of flow separation

1.3 Aerodynamic models and data

1.3.1 Models

Idea and simple example

One of the goal of the team is to design and calibrate flight models for existing and future airplanes. These are used to predict how the said aircraft would perform in all of its flight domain (for a given angle of attack α, mach number, etc). The model outputs values like the lift coefficient C_z, drag coefficient C_x, torques, etc. We can see it as a function:

Mach number is speed expressed as a proportion of the speed of sound.

$$\begin{aligned} f : \mathcal{X} \subset \mathbb{R}^n &\longrightarrow \mathbb{R}^6 \\ (\alpha, \beta, M, \dots) &\longmapsto (C_x, C_y, C_z, C_l, C_m, C_n) \end{aligned}$$

At its core, the model is a set of functions and their corresponding *labels* (i.e parameters). Each function models the behaviour of an aerodynamic value given a small set of inputs. The number of inputs varies, it can go up to four for some of the parts with the most complicated interactions, like for the Horizontal Tail Plane (HTP). So f might actually look like this

$$f(x) = \big(f_1(x_a, x_b), f_2(x_b, x_c), f_3(x_a, x_d, x_e), \dots\big)$$

The model is made of smaller parts. For the individual values like C_z, the result is the sum of the basic effect, ground effect, HTP effect, elevator, etc... At a larger level (which is more relevant to the problem at hand), the longitudinal, lateral and rotational effects are considered in isolation.

$$C_z(\alpha = 10, \beta = 5) = C_z^{\text{longi}}(\alpha = 10) + \Delta C_z^{\text{lat}}(\alpha = 10, \beta = 5)$$

Where $\forall \alpha : \Delta C_z^{\text{lat}}(\alpha, \beta = 0) = 0$

Example of a real (part of a) model

The model is concerned with six values : C_x, C_y, C_z, C_l, C_m and C_n. For each of these coefficient, the effect of different parts or configuration of the airplane is considered.

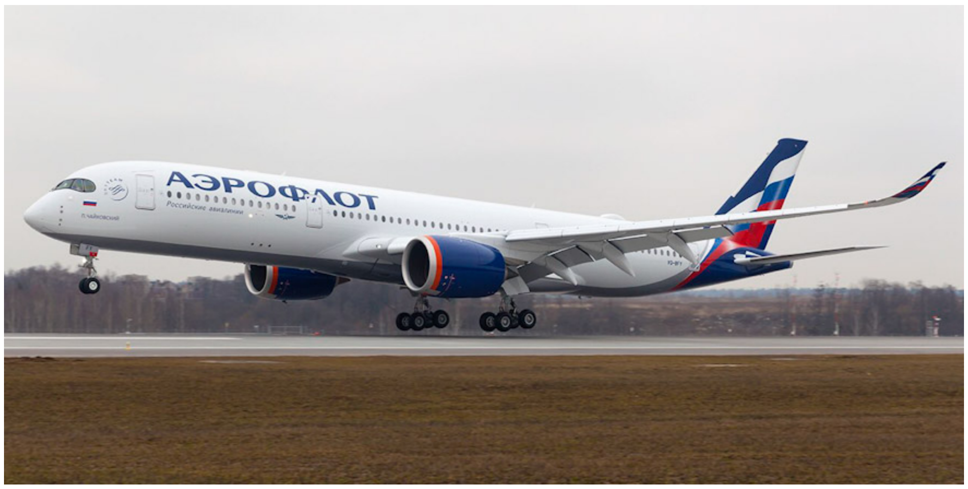
Let's take for example the lift coefficient. There is an expression for the effect of all the following on C_z: without tails, HTP, spoilers, elevators, pitch rate, incidence rate, ailerons, side-slip. This gives a decomposition like so (the details don't matter) :

$$C_z = \Delta C_z^{\text{WFP}} + \Delta C_z^{H} + \Delta C_z^{\delta q} + \Delta C_z^{sp} + \Delta C_z^{Ail} + \Delta C_z^{\dot{\alpha}} + \Delta C_z^{\beta}$$

Now, let's take the first part, the lift coefficient without tails ΔC_z^{WFP} (Wing, Fusalage, Pod). This itself is broken into two parts : before and after the stall. Let's look at the part before the stall.

$$\Delta C_z^{\text{ante-stall}} = \left((1 - K_z^{AS}) C_{z_\alpha} + K_z^{AS} \frac{C_{z_{\max}}}{\alpha_{\max} - \alpha_0} \right) (\min(\alpha, \alpha_{\max}) - \alpha_0)$$

The parameters in this expression are either labels or simple expression. The model for the lift needs more than 50 labels.



The slats (on the leading edge) and flaps (on the trailing edge) are deployed, giving more curvature to the wing. Lift and drag are significantly increased. See fig 3.3.

Figure 1.7: A350 in landing configuration

1.3.2 How a model is made

Generally, when a model is needed, the design starts with a reference model, from a well known aircraft, more or less similar to the target aircraft. Then, it is iteratively improved and tested, using a combination of Computational Fluid Dynamics (CFD), Wind Tunnel testing, and flight test data. Each of these have pros and cons.

CFD is very versatile and produces a lot of data: you can test any shape at any altitude, speed, etc. The results are volumetric and complete: you can access pressures for any point in space. However, it is very computationally intensive (therefore long and expensive), and also lacks predictability for highly turbulent flow (like in a low-speed, high-lift configuration). Indeed, turbulent flows are much harder to predict, and the result depends a lot on the chosen mesh and vortex modelling. There might be numerical instability and inaccuracies.

Wind Tunnel gives real data, and many configurations can be tested (it is easy to change the speed or α for example). However, the air-frame can't be changed easily, and the results do not directly translate to the real airplane (the scale is much smaller, which makes the Reynolds number quite different). Because of physical constraints for the placement of sensors, data points are very sparse compared to CFD. Wind Tunnel testing requires a larger up-front investment, and takes more time than CFD.

Some WT tests are done under a pressure of 4 bars and an atmosphere of 100 Kelvin nitrogen, to get closer to the Reynolds number of a real aircraft

Flight tests give definitely realistic data. It is also extremely expensive, and the data can be noisy. Lastly, the environment can't be easily controlled, and the configurations able to be tested are limited to what the pilots (or autopilot) can do. Even though this kind of the data is the *real deal*, there are many things to take into account. For example, rather counter-intuitively, the slat/flap configuration can influence the static pressure sensors which are at the front of the airplane. This is the kind of subtle interaction you have to keep track of to correctly make use of the data.

In the very early development phase, CFD is more often used, as it is much faster to iterate designs with, and thus enables to quickly explore the best options. It is also quite suited to the study of the high-speed cruising phase, where the airflow is more laminar. For more definitive designs and more turbulent air flows, wind tunnel testing is used together with CFD.

These different approaches are complementary. By having some redundancy and comparing the results between these, we are also able to get an idea of the uncertainty of these measurements and simulations.

1.3.3 Sizebox

The flight models are needed by other teams, namely IGF and IYC (the *clients*):

- Performance evaluation is done by IGF
- IYC integrates the Aero model with others parts, checks handling qualities, flight laws and potential failures.

However, in order to do the simulations using *Sizebox*, we need to have pre-computed data from the model. This data is organized in tables, consisting of a direct and systematic evaluation of every aerodynamic coefficient for every combination of flight parameters in a given discrete flight domain (for example, mach ranging from .1 to .6 with a .05 step, α ranging from -5 to 15 with 1 degree increments, etc).

These tables are on the order of 10 to 20 columns wide, and .1M to 1M rows long. A file containing all the tables for an aircraft might range from 100Mo to 1Go.

Using these tables, evaluating the model is as simple as looking up in the table, and interpolating if needed. This is much easier than evaluating many different equations, and also has the advantage to always be in the same format for every airplane.

1.4 Anomalies in the Data

1.4.1 Why anomalies?

When making an aero model, many things can go wrong

- The data could be noisy, or need correction. For example, CFD could be wrong in high-lift scenarios. Wind Tunnel data often needs correction because of the different Reynolds number.
- The model might be incomplete : the aerodynamic influence of a new component might not be well understood.
- Human errors can happen : a bug in software, a misplaced comma, a wrong sign...

All that to say that the data can, and probably will at some point, contain anomalies. This is especially true because a lot of the process, from CFD to data generation, is completely automatic.

1.4.2 Feedback loop between teams

As said previously, Sizebox data generated by IGAA is used by IGF and IYC to conduct simulations for performance and handling qualities evaluations. From there, two things can happen : either the data is ok, or it is wrong in some way.

In the first case, the development cycle continues nominally. In the other case, the testing teams have to report back to IGAA, which then has to find what the error is and where it is coming from. Once they fix it, they have to re-generate the data, and send it back. This situation slows down the process a lot.

Internally, finding anomalies is also hard. First, the data is high-dimensional and not easy to visualize. It also requires someone who knows aerodynamics well in order to notice when a trend or value isn't right. Last but not least, finding where the data is wrong is just the first step, you then have to correct the labels (which there are thousands of).

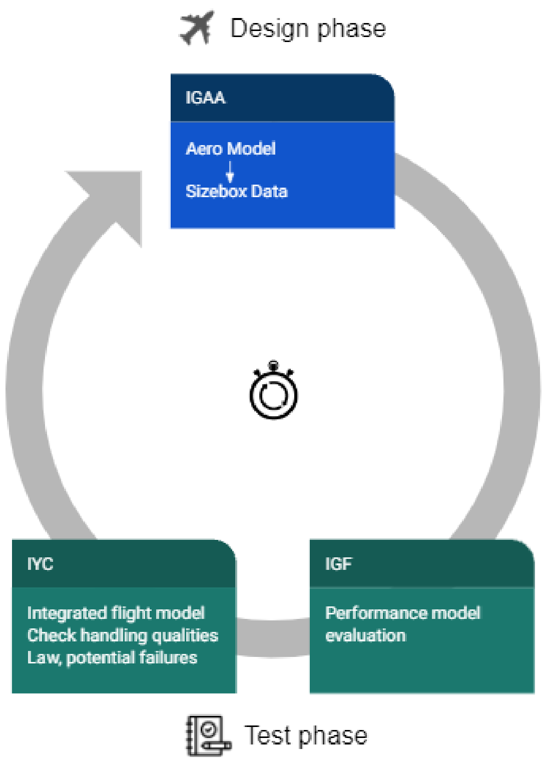


Figure 1.8: Illustration of the data flow between teams

1.4.3 Learn directly from the labels ?

This raises a question : why not detect anomalies in labels themselves ? They are, after all, the most fundamental parameters of the model. But there are two main reasons not to do that: different airplanes do not have the same number of labels, and do not share the same labels. This means the data itself hasn't the same structure or meaning between two airplanes. It would be really hard to train a model with that. In addition, labels are only one part of the model; the equations are the complementary part. Although they are based on physical principles, they might also be the source of anomalies.

On the contrary, the Sizebox data is in the same format for every airplane (some may have more parameters than others), and reflects the actual behaviour of the model; it depends on both the labels and the equations.

1.4.4 In short

The system as-is works. But quick human inspection of the data and simple checks are not enough to catch all anomalies. When some make their way to the delivered data, it slows down the development cycle, and takes precious time for an aerodynamic expert to fix. For these reasons, we need to automate anomaly detection as much as possible.

This is especially important for the new OoC phase, where many different airplane designs are going to have to be tested quickly.

Chapter 2

Theory: Anomaly Detection and the VAE

Now, the problem should be fairly clear. Let's quickly define what we talk about when we talk about anomalies; see some examples of what has been done in the field of AD; and go in depth into the model I've been using: the VAE. This section is formal but not strictly rigorous. I will try to give comprehensive details, but still provide intuition.

2.1 A quick theoretical view of AD

This section is heavily based on [6].

In this section, we are interested in what anomalies are. Having some data, we would like to find a way to detect 'strange' samples, to quantify how 'out-of-place' the data is, compared to a 'normal' distribution.

Let's try to get a crisper view of these terms.

2.1.1 Formal definition

Let's consider the data space $\mathcal{X} \subset \mathbb{R}^d$ (given by our task). We can then define what is normal as the distribution $\mathbb{P}^+$ on $\mathcal{X}$, corresponding to the way data would be distributed in a perfect world. This distribution can be very complex and conditional.

We consider that $\mathbb{P}^+$ has a density p^+ (with respect to the Lebesgue measure). This density p^+ gives us a measure of how normal a given data point is. Then, it feels natural to define anomalous data as $\mathcal{A} = \{x \in \mathcal{X} : p^+(x) \leq \tau\}$, the set of possible data points with a low density. The data distribution may as well be conditional, with a conditional density $p^+(x|t)$. This will be the case for our application.

In our problem, we suppose we have access to some data points $x_1, x_2, \dots x_n \in \mathcal{X}$ distributed following $\mathbb{P}^+$. This is an un-supervised setting, where we don't have access to any label.

We can distinguish two main types of anomalies:

- A simple point anomaly : a single $x \in \mathcal{A}$.
- A group anomaly : a cluster of related anomalies $\{x_j : j \in J\} \subset \mathcal{A}$

There is also a more qualitative distinction between low-level and high-level anomalies :

- Low-level anomalies could be artefacts in image, typos in words. They are closer to the raw data.
- High-level anomalies are semantic. They could be the pictures of abnormal objects, or text which doesn't belong to the same context.

High-level anomalies could be only small differences, as seen from the raw data space. That's why Deep Learning methods are more suited to detecting those, as they build useful and complex representations of the data at a more abstract level.

However, we usually never know p^+ (if we did, the problem would be solved). Then, the goal of AD is to approximate low-density regions of the data, either explicitly or implicitly.

2.1.2 Types of methods

We can distinguish three methods for Anomaly Detection : probabilistic, reconstruction-based and classification-based.

Probabilistic methods might be the most obvious and natural : if knowing the density is enough, just find a way to approach it (example: KDE, Normalizing Flows). Once you have an approximation of it, it is simply a matter of evaluation on the data of interest.

A **reconstruction**-based model learns to reconstruct the data (generally by passing through some kind of bottleneck like in a VAE). The expectation is that when the model sees an anomaly (which doesn't look like the data it was trained on), it will make a larger error in the reconstruction.

Classification models are adapted from the classical supervised setting, but strive to only learn *one* class, corresponding to normal instances. Intuitively, it tries to find the smallest fit in data-space.

We will focus on reconstruction-based AD. For this, we train a model on normal data (which contains no anomalies). This model learns to reconstruct this data sufficiently well. Then, when it is evaluated on something out of the ordinary, it generally makes a large reconstruction error $\|x - \hat{x}\|$. This is the signal we use for AD.

2.2 Previous works

The comparative study [2] shows that VAEs have been successfully applied to anomaly detection and segmentation in brain scans.

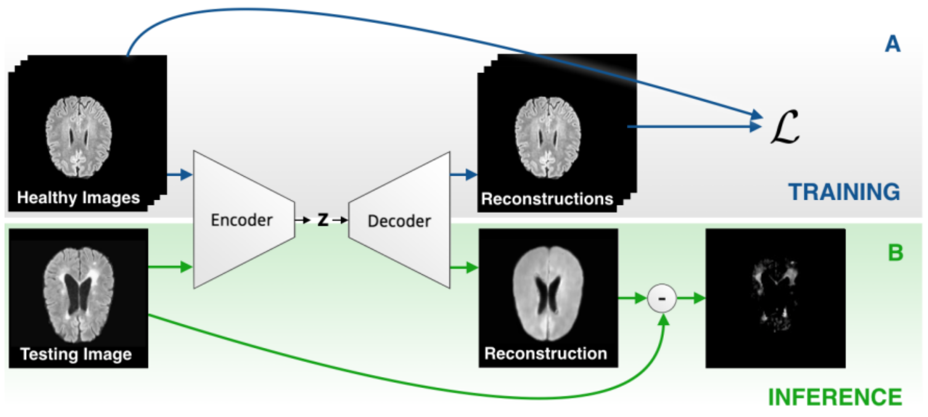


Figure 2.1: How a VAE is used for anomaly segmentation in brain scans [2]

In [5], a conditional VAE is used to detect anomalies in tabular data from the Large Hadron Collider (part of CERN). The authors first compare their method with others on some benchmarks (MNIST, fashion-MNIST), and then share their results on the LHC data. This paper contains an important insight that *many* implementations (including mine, for now) miss. I will talk about it more in section 2.3.6.

They also distinguish two types of anomalies. "Type A" anomalies are big changes on a single feature, while "Type B" anomalies are a small but systematic change on a group of features with the same condition. They argue the former can be detected with the reconstruction error (loosely) $\|\frac{1}{\sigma}(x - \hat{x})^2\|_\infty$, while the latter is more linked to the $\mathbb{KL}$ divergence. In the end, the two are combined with a logical `OR`.

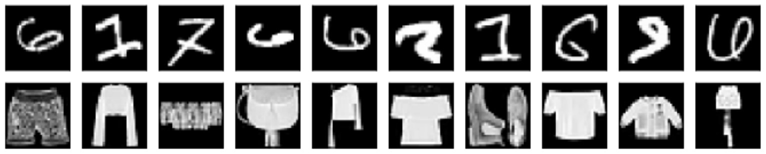
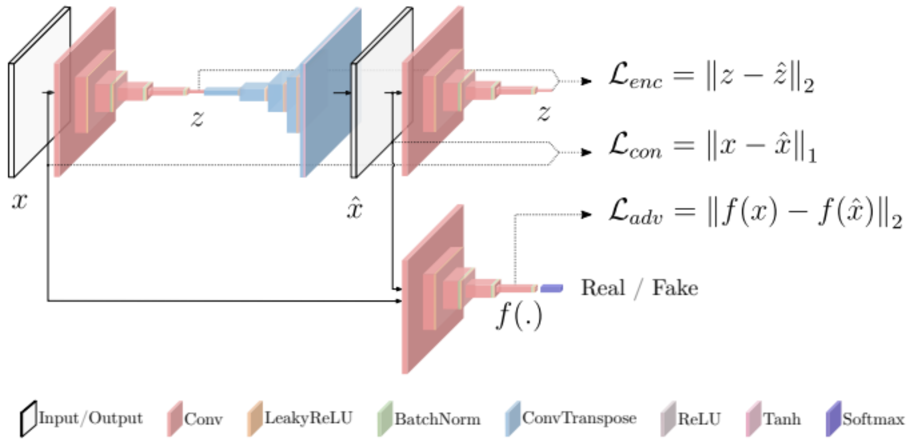


Figure 2.2: Most anomalous samples from MNIST and fashion-MNIST from [5]

The model shows good performance on both the benchmarks (see figure 2.4) and on the specific LHC data. They also find the model is applicable to many domains.

More intricate architectures can be used, like in [1], which is a kind of fusion between a VAE and a GAN, humorously called GANomaly. It shows good performance and computational efficiency compared to previous GAN-based techniques, on several datasets; which shows how adaptable these techniques are. This is a potential model to try in the future.



$\mathcal{L}_{enc} = \|z - \hat{z}\|_2$
$\mathcal{L}_{con} = \|x - \hat{x}\|_1$
$\mathcal{L}_{adv} = \|f(x) - f(\hat{x})\|_2$
x
z
$\hat{x}$
$\hat{z}$
Real / Fake
$f(.)$
Input/Output
Conv
LeakyReLU
BatchNorm
ConvTranspose
ReLU
Tanh
Softmax

Figure 2.3: GANomaly architecture [1]

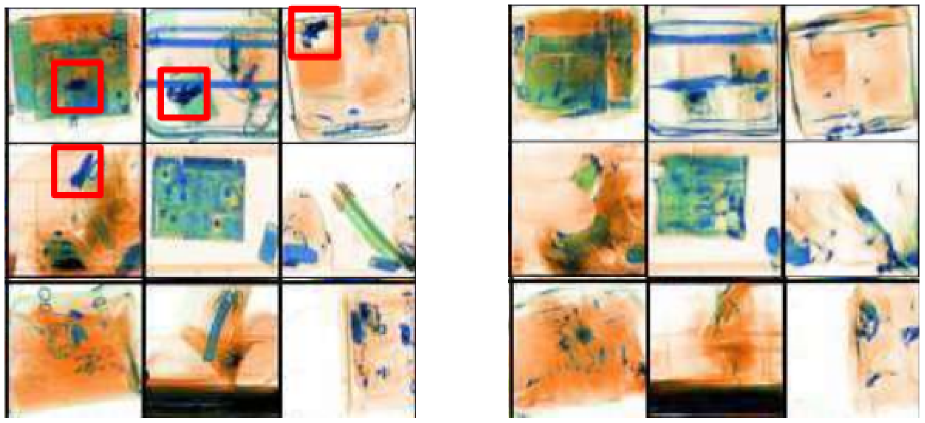


Figure 2.4: Example of weapon detection in scans from the FFOB dataset, with GANomaly [1]

The review [6] gives a good framework to categorize and compare different AD methods. For a general understanding of VAEs, I found [3] to be a good introduction, and [7] to provide a more in-depth understanding, as well as "debunking" common misconceptions about VAEs.

2.2.1 Why a VAE ?

Among all methods, I thought better to have one which gives a continuous measure of anomaly, rather than just a binary label. This excluded all one-class classification methods and similar.

Among those left, reconstruction-based ones seemed both intuitive in principle, and effective in practice.

The most widely used in the papers I read were Variational Autoencoders and Generative Adversarial Network. The former has a nice mathematical interpretation, and is easy enough to implement and train. The latter is more advanced, giving sometimes more performance, but is harder to implement and train.

In particular, the original GAN architecture is *not* suited to reconstruction-based anomaly detection, because there is not direct way to find a latent vector corresponding to the data. Instead, the GAN is expanded to include an encoder (for example). This leads to complicated architectures, which makes training and debugging harder.

That is why I have chosen to use the VAE, at least at first. We might try using GANs later, in the time left until the end of the internship.

2.3 VAEs in depth

We have access to a dataset $\mathcal{D} = \{x_1, x_2, \dots, x_n\} \subset \mathcal{X} \subset \mathbb{R}^d$, where each point is assumed to be identically distributed from a *true* distribution p^*.

2.3.1 The Classical Autoencoder

The autoencoder (AE) is a neural network which goal is to reconstruct the input it is given. The trick is that the data has to pass through a small-dimension layer, thus some information is lost and the problem isn't easy.

It is composed of two parts : an encoder f and a decoder g. The encoder takes the input x (an image for example) and returns a vector $z = f(x)$ in latent space $\mathcal{Z}$. This vector is of a smaller dimension than the input. Then, the job of the decoder is to take this latent vector and reconstruct the input as well as possible.

The goal of the network is to minimize the reconstruction error $\|x - g(f(x))\|$ (generally the L^2 norm). The intuition is that, by being forced to pass through the lower-dimensional latent space, the network will have to learn a useful representation of the data.

The AE is only good at encoding the data it has learnt. A random latent vector, which does not match with any data, will in general result in a gibberish output. This model lacks continuity, and is not a generative model.

2.3.2 The Variational Autoencoder

The Variational Autoencoder (VAE) happens to have the same structure as the Autoencoder, but it originates from a very different and probabilistic intuition. A VAE is first and foremost an unsupervised way to learn (complicated) distributions. Unlike an AE, a VAE is a generative model.

We have a parametrized distribution p_θ, which we would like to get as close as possible to the real distribution p^* by learning the parameters θ. As this distribution can be really complicated, it is useful to impose some structure into the model by introducing *latent variables*, written as $z \in \mathcal{Z} \subset \mathbb{R}^k$, where the dimension k of the latent space is smaller than the dimension d of the data.

The model can be represented as a Bayesian network (graphical model) with the following factorization :

$$p_\theta(x, z) = p_\theta(z)\ p_\theta(x|z)$$

Even when the prior $p_\theta(z)$ and conditional $p_\theta(x|z)$ are simple, the marginal distribution $p_\theta(x)$ (which can be though of as a mixture) can be very intricate. Often, we choose an isotropic Gaussian distribution :

$$p_\theta(x|z) = \mathcal{N}\big(x|f(z, \theta), \sigma^2 I\big)$$

For the model to have captured the distribution, there must exist a $z \in \mathcal{Z}$ in latent space for every $x \in \mathcal{X}$ in the dataset, so that z causes the model to generate something close to x.

We would like to maximize the probability $p_\theta(x)$ of each point $x_1, \ldots, x_n$ in the dataset. Indeed, we hope that if the model is likely to generate the training data, it is also likely to generate data that is similar.

$$p_\theta(x) = \int_{\mathcal{Z}} p_\theta(x, z)\ dz$$

However, this integral is intractable. Note that it follows from Bayes' formula that

$$p_\theta(z|x) = \frac{p_\theta(x, z)}{p_\theta(x)} = \frac{p_\theta(x|z)}{p_\theta(x)} p_\theta(z)$$

So while $p_\theta(z)$ is the prior distribution over latent space, we can see $p_\theta(z|x)$ as the posterior distribution over latent space, given the data. Given a tractable posterior, the previous integral isn't a problem anymore.

2.3.3 Posterior approximation

To solve the intractable posterior, we introduce an *inference model* (the encoder) $q_\phi(z|x)$, which has some parameters ϕ. The goal is to learn ϕ so that $q_\phi(z|x) \approx p_\theta(z|x)$.

The idea is to sample z that are likely to have produced x, and compute $p(x)$ from those. Intuitively, the space of likely z values knowing x should be much smaller than the Gaussian prior, which means we could compute $\mathbb{E}_{z \sim q_\phi}\ p_\theta(x|z)$ easily. But does it give us a good approximation of $p(x|z)$?

To see how these two are related, we need to use the Kullback-Leibler divergence.

2.3.4 $\mathbb{KL}$ Divergence

The KL divergence can intuitively be thought of as the amount of information that is lost by using one probability distribution q instead of a reference distribution p. If those are defined on $\mathcal{X}$ (q strictly positive) :

$$\mathcal{D}_{\mathbb{KL}}(p\|q) = \int_{\mathcal{X}} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

Which can be re-written as an expectation :

$$\mathcal{D}_{\mathbb{KL}}(p\|q) = \mathbb{E}_p \log p(x) - \log q(x)$$

Plugging the distributions we are interested in, and applying Bayes' Rule:

$$\begin{aligned} \mathcal{D}_{\mathbb{KL}}\big(q_\phi(z|x)\|p_\theta(z|x)\big) &= \mathbb{E}_{z\sim q_\phi(z|x)} \log q_\phi(z|x) - \log p_\theta(z|x) \\ &= \log p_\theta(x) + \mathbb{E}_{z\sim q_\phi(z|x)} \log q_\phi(z|x) - \log p_\theta(x|z) - \log p_\theta(z) \end{aligned}$$

And finally

$$\log p_\theta(x) - \mathcal{D}_{\mathbb{KL}}\big(q_\phi(z|x)\|p_\theta(z|x)\big) = \mathbb{E}_{z\sim q_\phi(z|x)} \log p_\theta(x|z) - \mathcal{D}_{\mathbb{KL}}(q_\phi(z|x)\|p_\theta(z))$$

- The left hand side is what we want to maximize, especially $p_\theta(x)$. The divergence can be though of as a regularization term, which tends to make $q_\phi(z|x)$ more like $p_\theta(z|x)$. These quantities are however not tractable.
- The right hand side is what we actually can maximize, using SGD-based algorithms.

The autoencoder structure emerges (figure 2.5): q_ϕ encodes x to z, and p_θ reconstructs it into $\hat{x}$. It is important to keep in mind the difference between the probability distributions, and the functions that implement them. Generally speaking, p_θ and q_ϕ are *parametrized* probability distributions. The actual code will implement functions that will output these parameters.

2.3.5 Latent space

A simple assumption is made here : the latent space is Gaussian. The trick is that we can use a (sufficiently powerful) function approximator (like a neural network) to go from a Gaussian distribution to any other distribution.

$$p(z) \sim \mathcal{N}(0, I)$$

2.3.6 Training objective

Usually, q is also chosen to be an isotropic Gaussian :

$$q_\phi(z|x) \sim \mathcal{N}\big(\mu_\phi(x), \sigma_\phi(x)\big)$$

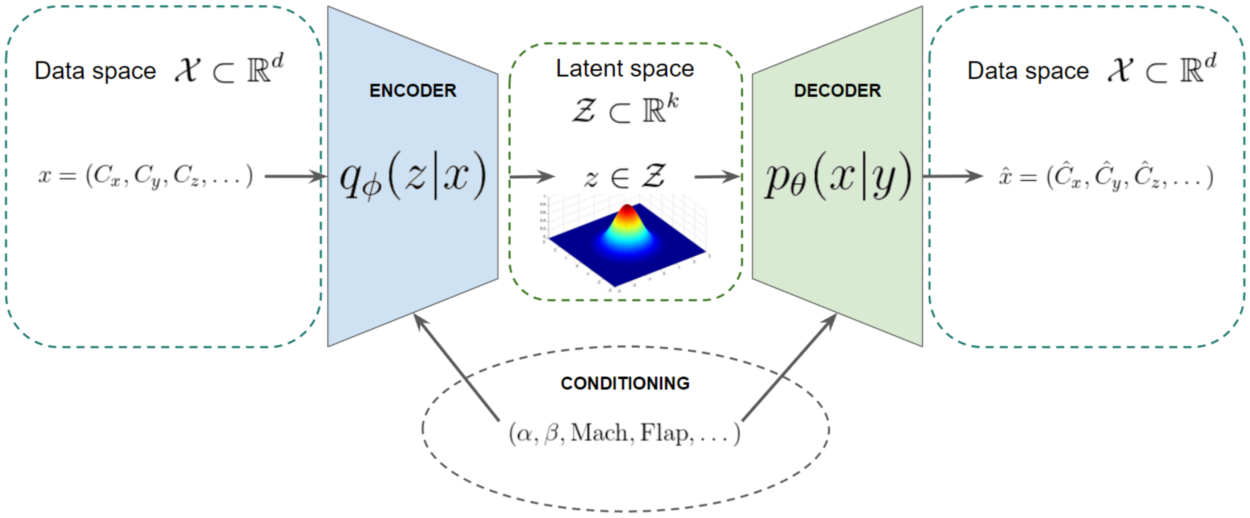

Data space $\mathcal{X} \subset \mathbb{R}^d$
$x = (C_x, C_y, C_z, \dots)$
ENCODER
$q_\phi(z|x)$
Latent space
$\mathcal{Z} \subset \mathbb{R}^k$
$z \in \mathcal{Z}$
DECODER
$p_\theta(x|y)$
Data space $\mathcal{X} \subset \mathbb{R}^d$
$\hat{x} = (\hat{C}_x, \hat{C}_y, \hat{C}_z, \dots)$
CONDITIONING
$(\alpha, \beta, \text{Mach}, \text{Flap}, \dots)$

Figure 2.5: Diagram of the chosen VAE model

Where $\mu_\phi(x) = (\mu_1, \mu_2, \dots, \mu_k)$ and $\sigma_\phi(x) = (\sigma_1, \sigma_2, \dots, \sigma_k)$ are the outputs of the neural network with weights ϕ used for the encoder. In this configuration with a Gaussian prior $p(x)$ and Gaussian posterior $q(z|x)$, the KL divergence has a simple analytical form (quite tedious to derive), which is easy to compute.

$$\begin{aligned}&\mathcal{D}_{\mathbb{KL}}\left(\mathcal{N}(\mu_0, \Sigma_0) \| \mathcal{N}(\mu_1, \Sigma_1)\right)\\ &= \frac{1}{2}\left(\operatorname{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T\ \Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\frac{\det \Sigma_1}{\det \Sigma_0}\right)\end{aligned}$$

Applied to our case (with Σ diagonal with values $\sigma_1, \dots \sigma_k$) :

$$\begin{aligned}&\mathcal{D}_{\mathbb{KL}}\left(\mathcal{N}(\mu, \Sigma) \| \mathcal{N}(0, I)\right)\\ &= \frac{1}{2}\left(\operatorname{tr}(\Sigma) + \mu^T\mu - k + \log\frac{1}{\det \Sigma}\right)\\ &= \frac{1}{2}\sum_{i=1}^{k} \sigma_i + \mu_i^2 - 1 + \log \sigma_i\end{aligned}$$

For estimating $\mathbb{E}_{z\sim q(z|x)} \log p(x|z)$, we simply sample one $z \sim q(z|x)$ and consider $\log p(x|z)$ as an approximation of the previous expectation (this can be though of as Monte-Carlo with a sample size of 1). This is good enough, as we are anyway going to average over mini-batches during optimization.

As we chose an isotropic Gaussian distribution for $p(x|z)$, we have (these are μ_i and σ_i of the decoder, different from the previous ones).

$$\begin{aligned} p_\theta(x|z) &= \prod_{i=1}^{d} \frac{1}{\sigma_i\sqrt{\tau}} e^{-\frac{1}{2}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2} \\ \log p_\theta(x|z) &= \sum_{i=1}^{d} -\log\sigma_i - \frac{1}{2}\log\tau - \frac{1}{2}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2 \\ &= -\frac{d}{2}\log\tau - \sum_{i=1}^{d}\log\sigma_i - \frac{1}{2}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2 \end{aligned}$$

Note that if we choose $\sigma_i = 1$ for all i, the negative log probability is almost the mean squared error :

$$-\log p_\theta(x|z) = \frac{d}{2}\log\tau + \frac{1}{2}\|x-\mu\|_2^2$$

Many implementation make this choice, and have a final loss that is MSE + KL. The assumption of every feature having the same amplitude in patterns might not be realistic at all. This point is raised in [5] and [7]. The confusion might stem from thinking that the decoder directly outputs a reconstruction $\hat{x}$. This is not (should not) be the case. Instead, like the encoder, the decoder is meant to represent a distribution.

So what many people think is the reconstruction $\hat{x}$ is instead the mean of the Gaussian distribution that the decoder is meant to represent. Leaving out σ makes it a more constrained distribution, which might not fit the data as well.

Indeed, considering $\sigma = 1$ for all features makes learning difficult (impossible?) for patterns in the data having a smaller resolution. Most of the time, the data is normalized to have a variance of 1. It is very likely then, that patterns in the data will have a smaller variance than that.

This problem might be mitigated by β-VAE [4], which I will go into later.

Chapter 3

Practice, results, and visualization

My tutor, Xavier Bertrand, has been working on applying Deep Learning techniques on Aero data since 2016. While DL was starting to become popular in the IT world at this time, it wasn't really a thing in the Aeronautical industry. Xavier has had the opportunity to try it, without knowing for sure it would work.

However, it worked really quite well, and there are now things that can be done correctly only with Deep Learning. This has been applied, for example, on pressure coefficients (around the wing profile) in order to fuse sparse real data, and complete but un-calibrated CFD data. It is also being used for *surrogates*, which are simply neural nets taught to reproduce a model, in a more truthful way than interpolating between know data points.

In this setting, VAEs have already been used, and a couple of people in the team are used to Deep Learning techniques. The choice of VAE was also guided by Xavier.

After understanding theoretically how a VAE works, I wanted to get familiar with the code on a simple example. I made the obvious choice and went with MNIST, the *Hello World* of Deep Learning. Plenty of papers and implementations also show their results on this dataset, which meant I could see whether I got similar results.

Please keep in mind that, at the time of the writing, I'm two-thirds into the internship. There are a lot of things I'm still working on, and that I need to figure out in the next month and a half. For example, all the following results use a β-VAE, as I did not realize the problem in [5] until recently, and haven't had the time to implement "σ-learning" in the decoder.

3.1 Tools*

3.1.1 IT environment

The physical computer I used was running Windows. However, all the programming and training was done on Virtual Machines on an internal server named *Giseh*. They were running Red Hat Enterprise, and some were equipped with Tesla GPUs, for Deep Learning.

As few machines were configured for Deep Learning, they were quite busy, and it was sometimes a challenge to get access to it if I hadn't connected early in the morning. They other machines were however perfectly fine if no training had to be done.

3.1.2 Programming environment

Most people in the team who develop in Python use Pycharm, an IDE specifically made for this language. I have tried using it, but I found it rather clunky, a mess to configure properly, and not very efficient because I wasn't used to it at all.

VIM, if the reader isn't familiar with it, is a command-line text editor first released in 1991. It is quite old, basic and un-intuitive. I was already acquainted with it, so I started using at first, thinking I would switch to PyCharm once it would work properly.

The particularity of VIM, compared to modern text editors, is that the mouse is rendered absolutely useless: everything is done using the keyboard. This may sound restrictive, but once you get the hang of it, moving around and doing commands becomes very fast and efficient. It also has the advantage of being *very* modular. The basic commands can be combined in many ways. You can customize everything, create macros and scripts, and use the large number of plugins developed by the community. Little by little, I customized it to suit my needs, and my configuration file grew to almost 200 lines.

For example, I was really used to Jupyter Notebooks and the ability to execute a chunk of code, interact with the variables, etc. To mimic this, I used VIM's internal terminal emulator to open an IPython shell (even better, use the `%autoreload 2` command to automatically reload changed library files, and apply it to already-imported functions and already-created objects!). I found a script somewhere on StackOverFlow which enables the user to select some text, send it to the shell and execute it. This way, I could have both the powerful text editor and the flexibility of Jupyter Notebooks, which is very handy for experimenting.

During this time, I learnt a lot about how VIM works and how to make it mine. I am quite comfortable with it, and it would now be even harder to switch to another editor or IDE. I like the lightness and simplicity of it. Because it is so simple, nothing is hidden as it would be in an IDE. This comes at the price of a lot of tinkering.

3.1.3 Linux & Git

I was already very familiar with Linux, having played a lot with it in my youth. I did learn a few things about it, and how to customize it (like the `.bashrc`). As the project got more complicated, I started using `git` for version control. It is really useful to separate features and fixes, have a history of changes, and to backup code in case something happens to the local copy. I have used it to revert some changes that didn't work out, which saved me a lot of time.

3.2 Familiarization on MNIST

3.2.1 First draft

I obtained the code for a VAE from some previous work by the team. It was written using the TensorFlow library, which is the choice of the team for Deep Learning. Replacing the encoder and decoder by some convolutional and transpose convolutional layers, I simply adapted it to the 28 by 28 pixels images of MNIST.

The model is defined through five different classes :

- The `Encoder` class which handles the sampling and encapsulates an `Encoder_layer` object.
- In the same way, there is a `Decoder` class encapsulating a `Decoder_layer` object.
- The main `VAE` class ties it all together, defines the loss as well as the training and testing steps.

I kept the same structure throughout the whole project.

3.2.2 Conditioning

Once I was sure everything was working properly, I added conditioning.

In the original VAE, x is fed to the encoder f to get a latent representation z, which is the only information the decoder g has access to in order to reconstruct x.

$$f : x \longmapsto z$$
$$g : z \longmapsto \hat{x}$$

A very popular and successful modification is to condition the input by some additional information y. There, both the encoder and decoder have access to y.

$$f : (x, y) \longmapsto z$$
$$g : (z, y) \longmapsto \hat{x}$$

This enables us to decrease the amount of data the latent space has to encode and have better reconstructions

In the case of MNIST, we can condition on the label (the actual digit). For the encoder, this was done by one-hot encoding the label, concatenating it with the output of the convolution, and feeding everything to a dense layer. Similarly, for the decoder, the one-hot-encoded label was simply concatenated with the latent vector, which together were fed to a dense layer, and then to the transpose convolutions.

3.2.3 β-VAE

The implementation of the VAE I started from had the classic loss, with MSE + KL. This means that the variance of the decoder was implicitly assumed to be 1 (see section 2.3.6). I did not really understand this at the time, but as a fix for this rigidity in the model, I added a simple modification known as β-VAE.

Indeed, a major issue with the current setup is that the KL-divergence often term provides too much, or too little regularization. A simple way to remedy this was introduced in [4], where, in the final loss, the divergence term is multiplied by a β factor.

In the paper, this was introduced as a way to disentangle features in the latent space. The hope is that, by doing this, each dimension in the latent space will represent something more simple, and more interpretable.

3.2.4 An alternative interpretation of β

Consider what was derived in 2.3.6:

$$-\log p_\theta(x|z) = \frac{d}{2}\log\tau + \sum_{i=1}^{d}\log\sigma_i + \frac{1}{2}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2$$

Now, if we assume the variance is the same for every dimension, that is $\forall i : \sigma_i = \sigma$, we have :

$$\begin{aligned}-\log p_\theta(x|z) &= \frac{d}{2}\log\tau + d\log\sigma + \frac{1}{2\sigma^2}\sum_{i=1}^{d}(x_i-\mu_i)^2\\ &= \frac{d}{2}\log\tau + d\log\sigma + \frac{\|x-\mu\|_2^2}{2\sigma^2}\end{aligned}$$

We can see that if σ is fixed rather than learnt, the first two terms are constant and do not matter for the optimization process. However, lowering σ leads to an increased importance of the MSE in the final loss, and conversely.

Seeing that, we can view the β-VAE technique as not multiplying the KL divergence by β, but as dividing the MSE by β (because that doesn't change the objective). As such, it seems that using β-VAE is equivalent to making the assumption that $\sigma^2 = \beta/2$. This somewhat fixes the issue (although the variance is still the same for every feature), but now we have yet another hyper-parameter!

3.2.5 Most anomalous samples

MNIST is already quite curated and clean, but it does contain images weirder than others. This is a good opportunity to check whether a VAE could pinpoints samples that would, to the human eye, seem anomalous. This has already been done in [5].

There are three possible sources of information for detecting anomalies.

- First, the reconstruction error (or the normalized reconstruction error $\|\frac{1}{\sigma}(x-\mu)\|$, as in [5]) either as the MSE or the max error (the latter gives, in practice, better results).
- Second, the $\mathcal{D}_{\mathbb{KL}}$ divergence term, which in practice gives poor results, but may be better at detecting group anomalies, or detect different kinds.
- Third, you could look at the latent representation of a sample, and consider it anomalous if it deviates too much from the origin (which is supposed to be a standard centered Gaussian). In practice, it's not a very good indicator.

On its own, the reconstruction error give the best results in almost every situation, and that is what is used in practically every reconstruction-based anomaly detection paper (mainly based on VAEs and GANs).

The images getting the highest score for the MSE and the KL loss are presented in figure 3.1. Many of these digits are indeed badly written, or even illegible. However, some look normal (apart from being thicker). I think it's fair to say the model has captured some of the "anomalousness" of this data.

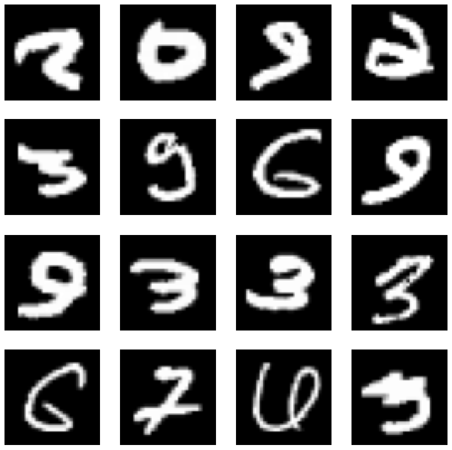
3.3 Goals and Challenges

3.3.1 Goals

The main goal of this internship is to study the *feasibility* of Anomaly Detection in Sizebox data. We would like to know:

- The different kinds of anomaly that are in the data. For this, we have to look more closely at certain trends, and ask people who often work with the data.
- Which kinds can be detected.
- If the technique is precise enough to be useful

The long-term goal (if the technique works well enough) is to provide a tool to find anomalies quickly (even better, automatically) and integrate it into the production pipeline.


(a) MSE

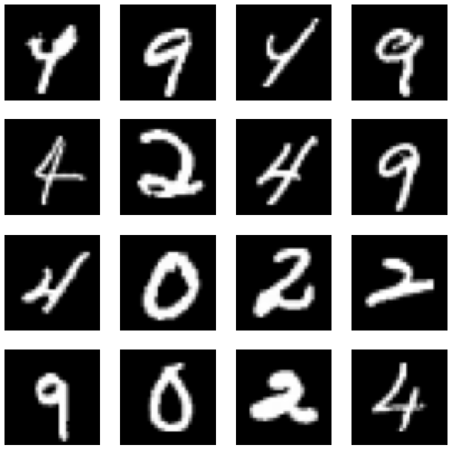

(b) KL

Figure 3.1: Most anomalous samples with respect to the MSE and KL loss

3.3.2 Challenges

Data

The data might (and probably will) be contaminated. This means the model will train on *some* anomalies, which might degrade its performance if it is confronted to the same kind of anomalies during testing. To mitigate this, we may need regularization and more data.

We also have to deal with noisy data. Indeed, the equations and labels in the model are not perfectly coherent. This is especially visible in the high-lift tables, where some trends are not as smooth as they should be. Noise like this will hinder training, and will probably increase the reconstruction error for normal samples, which will make seeing anomalies more difficult.

There are also choices to be made in the preparation of the data. To get a versatile model, we need to train it on many different kinds of airplanes. For each of these planes, there are several sizebox tables which are quite large. There will be some trade-offs between data size and independance of the different effects.

During testing, we need to make sure our model remains generalizable, and does not only memorize the training data. It must both be able to reconstruct normal data from another airplane correctly, and still detect anomalies.

Architecture

Even if we have chosen a model, every architecture remains imaginable. For example, in a VAE, there is complete freedom in the implementation of the encoder and decoder. If they are dense networks, there is the depth and width to choose. It could also be a more deliberate structure, following known physical principles for example.

Beyond these, the huge hyper-parameter space has to be explored, as always. Optimizer, learning rate, regularization parameter, activation function...

Testing

Anomalies are, by definition, rare. There are also not easy to find. Thus, it might not be easy to come up with a good picture of what anomalies look like; and this is especially true for aerodynamic data, which is high-dimensional and not easy to visualize and understand.

This poses a challenge for testing: we need to create a dataset (or several) containing anomalies, so that we can have an idea of the model's performance. We could also run the model on unvalidated data, and see if the high-error points are indeed anomalies. However we might unknowingly miss some.

To check the performance, having access to a labeled (normal/anomalous) dataset, we can plot the ROC curve and error distribution in order to see how much the model is able to detect, and how well normal and anomalous data are separated.

3.4 Preparing the data

3.4.1 Shape of the data

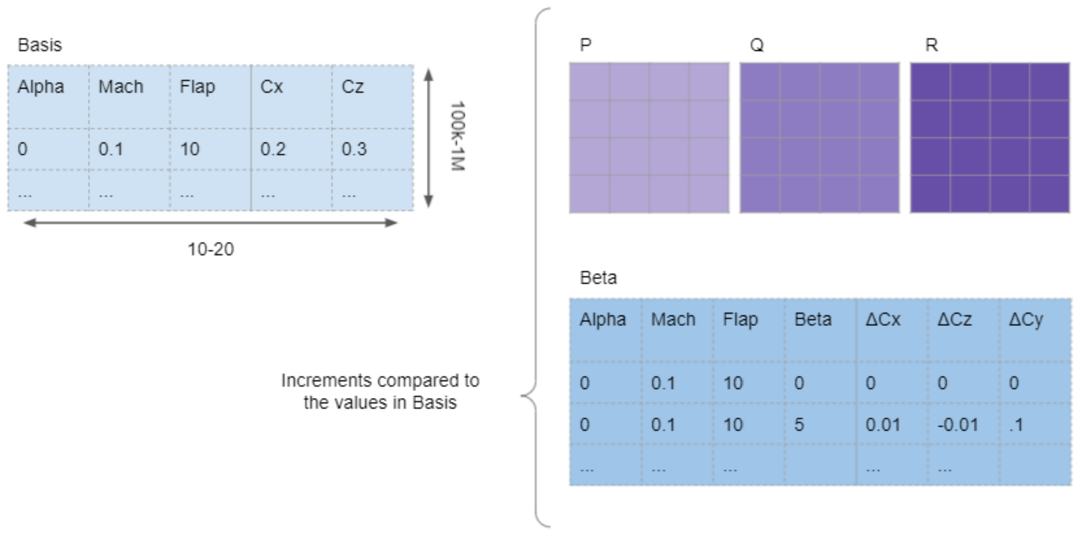
Like I said, the data is comprised of several tables, each of them evaluating some Aerodynamic coefficients (plus some gradients) for many combinations of flight parameters (α, β, Mach, etc). However, it's not just one big table for each aircraft. Instead, some effects are de-coupled and put into different tables.

The most important table is `Basis`, which only includes longitudinal effects. In it, no β in the parameters, no C_y in the computed coefficients (because they are always 0 here). Other effects are going to be expressed as an increment from this table.

For example, the effect of side-slip is in the `Beta` table. To get the complete picture for given flight parameters, you have to also look into the `Basis` table for the corresponding parameters, and add them with what is in `Beta`. The effect of rotations is included in three other table, `P`, `Q` and `R`.

This follows the principle in 1.3.1.

If the tables weren't separated this way, the sheer combinatorics of all the different parameters would mean outputting huge files, a hundred gigabytes or more.



Alpha	Mach	Flap	Cx	Cz
0	0.1	10	0.2	0.3
...	...	...	...	...

Alpha	Mach	Flap	Beta	ΔCx	ΔCz	ΔCy
0	0.1	10	0	0	0	0
0	0.1	10	5	0.01	-0.01	.1
...	...	...		...	...	

Figure 3.2: Schematic view of Sizebox data for one airplane

3.4.2 Implementation

The actual files are in `matlab` format. In it are the tables (Basis, Beta, etc). In a given table, the data is stored as a big tensors, each corresponding to a coefficient or gradient. For example, in `Basis`, there might be coefficients C_x, C_z, C_m and their partial derivatives $\frac{\partial C_x}{\partial \alpha}$, $\frac{\partial C_z}{\partial \alpha}$, $\frac{\partial C_m}{\partial \alpha}$, $\frac{\partial C_x}{\partial \text{Mach}}$, $\frac{\partial C_z}{\partial \text{Mach}}$, $\frac{\partial C_m}{\partial \text{Mach}}$. In each of these tensor, each dimension represents a parameter.

You also have two arrays, `sweep_names` and `sweep_values`, which tell you which parameter is being changed on each dimension, and what values those parameters take.

For example, the first dimension might correspond to α, containing 26 elements with $\alpha = -10, -9, \ldots, 25$.

3.4.3 Re-shaping

In my case, this format is really unpractical to work with: first it's in matlab format, and I'm going to use Python all the way. Second, it's not easy to access the data: you have to look for each coefficient in a different tensor.

To solve this, I made a script which reads this matlab file, and returns any of its table as a Pandas dataframe. Each line correspond to a specific configuration: the first few columns are the flight parameters, and the other are the computed aerodynamic coefficient and gradients.

Now, two things have to be done before we can use this data for training. First, one parameter is unlike the others : the slat/flap configuration.

3.4.4 Slats & Flaps

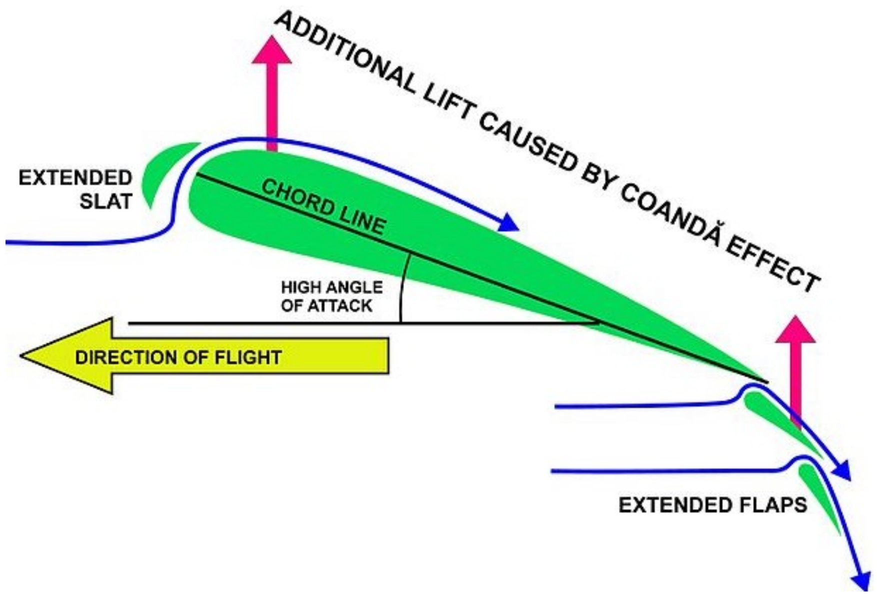


Figure 3.3: Slats and Flaps

Slats and flaps are devices which can extend the leading edge and trailing edge of the wing, in order to significantly increase lift (but they also increase drag). They are used during take-off and landing. These can be deployed at several angles (like 0°, 5°, 10°, 20°). But generally, the angles at which slats and flaps are deployed are summarized in a single integer know as a configuration. For example, for the A321, these are the possibilities :

Conf	Slat	Flap
0	0	0
1	18	3
2	18	10
3	22	14
4	22	23
5	27	26
6	27	34

The configurations are different for every aircraft, which might make learning just from the configuration number a bit delicate. Instead, I chose to replace the conf. by the corresponding values of slat and flap.

3.4.5 Normalization

The second step is simply to normalize the whole dataset, as some columns have wildly different values (coefficient are smaller than 1, while the altitude goes up to 40k feet). I chose the standard $\sigma = 1$, $\mu = 0$ normalization, which is standard practice for neural nets.

3.5 First try on ZEROe data

Once I had everything working properly with MNIST, I was ready to try it on real areo data. My first test were on the ZEROe aicraft, which is a project of small, multi-engine electric aircraft.

Xavier gave me two tables for this aircraft: the normal one, and another with tweaked labels, to simulate anomalous data. The goal was to see how well the model could reconstruct the original data, and whether it could differentiate the anomalous data.

This was also an opportunity to test the influence of various factors, like the number and depth of layers, the size of the latent space, the value of β and other hyper-parameters. However, I first tried to train traditional machine learning algorithms ("shallow learning"), in order to have a benchmark to compare the VAE with.

3.5.1 Data and training

The original data was separated into a train set and test set. I tried several splits (more on that later). The model was only trained on the former, while both the test set and anomalous set were used for testing.

3.5.2 Metrics and benchmarks

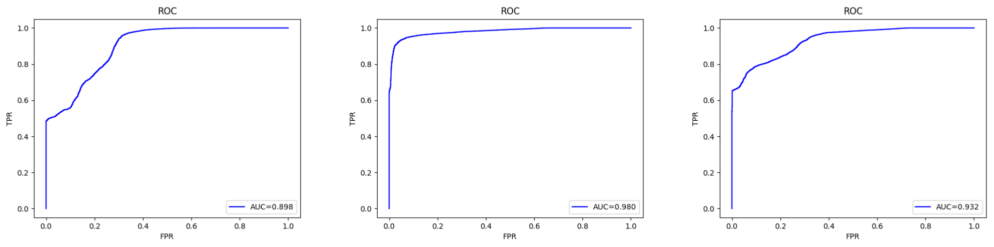
To measure performance, I used two kinds of plot :

- A ROC curve to see what kind of error detection could be achieved.
- An error distribution curve, for both the test set and the anomalous set, to see how well the two were separated.

From all the methods tested (Isolation Forest, SVM, LoF and Elliptic Envelope), the LoF gave the best results (see figure 3.4), with an excellent AUC of 0,994. The hyper-parameters weren't fined-tuned.

3.5.3 Results

With a VAE, getting a better performance than with the LoF method wasn't hard, with an AUC comparable or better. Here are some things I tried and checked.



(a) Normal model (b) Huge model (c) Huge model with overfit

Figure 3.4: ROC curves for VAEs trained with a missing chunk of the flight domain

A first possibility was that the encoder was useless, and that the decoder did all the work, acting like a surrogate (learning the whole model). This is not (completely) the case, as removing the encoder lead to a significant performance decrease (the AUC went from $\sim$ .99 to .96). Removing the conditioning lead to an even worse $\sim$ .90 AUC.

I found that reducing the training size, from 100k points to only 5k decreased performance only slightly. That is not too surprising, considering how close to each other many points are.

Unexpectedly, increasing the latent dimension made the model perform consistently worse. With latent dimensions 1, 2 and 4, the AUC went from .997 to .99 to .97. This might be over-fitting in action.

To understand if the model could generalize and how the number of parameters affected it, I tried to train on non-shuffled data. More precisely, instead of randomly sampling the train and test set, I instead took a chunk in the middle of the data for the test set. This means the model has *not seen at all* a whole part of the flight domain. The "normal" model performed poorly at .90 AUC. The "huge" model (as in over-parametrized), however, did much better with .98. But let it overfit, and it drops down back to .93.

Lastly, I looked at the influence of data contamination. I put 2% then 5% of the anomalous data into the training set. In both cases, the VAE managed an AUC of $\sim$ .995. Some data contamination does not seem to be too much of a problem, then.

3.6 Training on the actual data

3.6.1 Data problems

After these preliminary tests, I had to adapt it to the actual data we are interested in. We would like the model to work on a wide range of aircraft, from the A320 family to the A330 and A350. The data for these aircraft is more comprehensive (than for the ZEROe), with hundreds of thousands of lines in each table, for each aircraft.

The question then, is how to combine this data before training. At first, I naively wanted to combine all the tables (`Basis`, `Beta`, etc) into just one. But I should have realized, before implementing it, that this was going to result in *huge* files (tens or hundreds of gigabytes). This is aggravated by the fact that the model isn't evaluated on the same grid in the different tables. For example, in `Basis`, α might have a 1 degree step, while in `Beta` they're just 2 degrees steps. Then we have to interpolate to the finer level, which makes files even bigger.

I thought about random sampling, or keeping fewer points. However, we would have to follow the same procedure when testing, and we do need to test *every* point, when looking for anomalies. The simpler solution, at this point, was just to train one model for each table. This isn't ideal, but will suffice for a start.

3.6.2 Dataset

The data was prepared as previously stated (re-shaping, slat/flap translation, normalization). In addition, in order to have a balanced dataset and not "favour" any aircraft, I sampled the same number of point from every file (as many as the smallest of them contained). This should not be a problem for learning the trends, because points close to each other are very similar, and are expected to not contain anomalies.

In the following sections, the dataset is comprised of eight `Basis` tables, from four different planes: A321neo, A330-800, A350, and NSA (a prototype, similar to the others). Each of them has a High-Speed (HS) file (no slat or flap) and a High-Lift file (lower speed, with every configuration of flat/slat). This results in a 150 Mo dataframe, with 1M rows and 15 columns.

3.6.3 Training

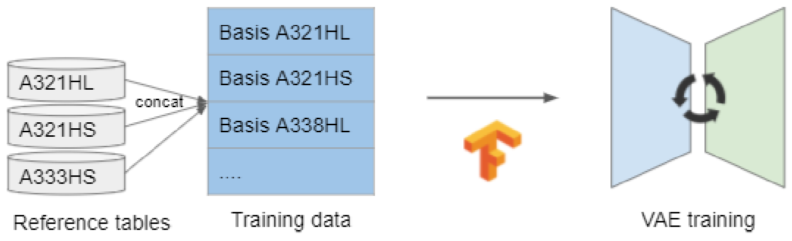
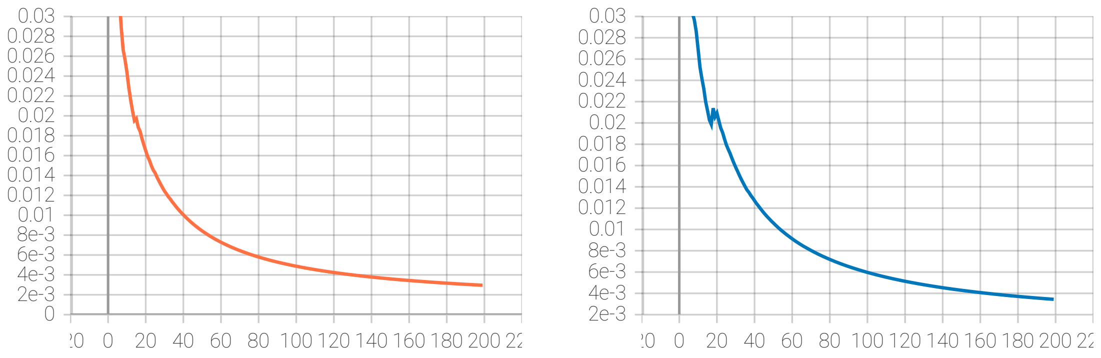


Figure 3.5: Training with Sizebox data

The following parameters were chosen:

- 200 epochs. To check for overfit, another file was used as a testing set at each epoch. The reconstruction error kept going down for both the training and testing set (see figure 3.6)
- `Adam` optimizer with a starting learning rate of 0.003 (Higher, the optimization process would sometimes become unstable). After epoch 15, the learning rate was progressively decreased.



(a) Train MSE (b) Test MSE

Figure 3.6: Reconstruction error while training on the full data

- Three dense layers (256, 512, 256) for both the encoder and decoder. This was chosen by intuition alone, and what I deemed to be enough learning capacity. As we saw earlier, a bigger model seems better.
- A latent dimension of two. Both because it seems to be enough to attain a low error, and because it is easy to visualize. One was enough for the ZEROe data, but here we have several airplanes. This information is not in the conditioning, which means it has to be encoded *somehow* in the latent vector, hence the need for more capacity.

These parameters were chosen with a mix of intuition, tinkering, and the result of the previous experiment. The slow training ($\sim$ 1h), sometimes difficult access to GPUs, and tricky testing, made experimenting with hyper-parameters troublesome, and not a priority for now.

After 200 epochs, an MSE of .003 was reached. This means, very roughly, a typical error of .05 on the normalized data, which seems reasonable.

3.7 Visualization and Validation

Once the model is trained, we can pass the table we need to test though the VAE, and compute the reconstruction error for each point and each value (figure 3.7). This gives an idea of how anomalous a point is, and which value is the most anomalous. However, this output is of the same dimension and size as the input table, which makes exploring difficult.

In order to see whether the model learned something useful, we need a way to visualize the output. After trying `matplotlib`, `Bokeh` and `Holoviews`, I settled on `Dash/Plotly` to make an interactive dashboard. It is composed of several graphs which help locate the data regions with the highest error, and see what the data looks like there. Details about the dashboard are in appendix B.

Testing and validation are the challenges here. There is no labeled test dataset, no history of anomalies. This makes understanding how well the model performs not so straightforward.

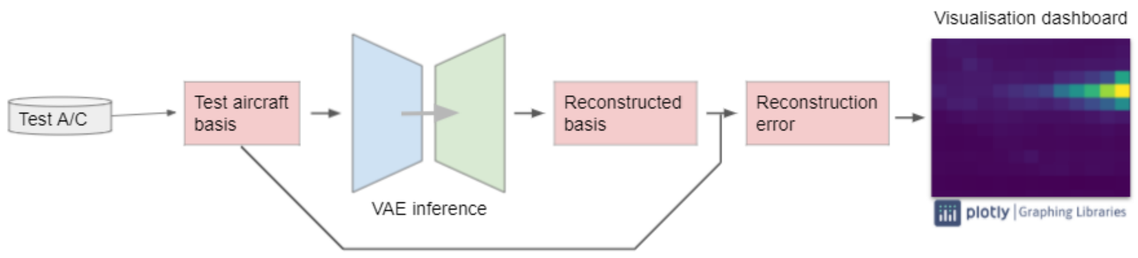


Figure 3.7: Testing the trained model

3.7.1 Latent Space

Plotting the latent space (figure 3.8) shows two things.

First, the training went well, as it looks close enough to a standard Gaussian distribution. On the smaller ZEROe dataset, I sometimes ended up with many clusters, which isn't supposed to happen.

Second, we can distinguish more or less defined clusters corresponding to each airplane. This means that, like we hoped, the VAE encoded *in some way* the type of aircraft in the latent space, without itself having direct access to this information.

3.7.2 Artificial Anomalies

To see whether the model could detect anomalies, Xavier changed a couple of labels and generated a new table. These changes only affected part of the flight domain, and I had to see whether I could see what was going wrong, and where. To do this, I evaluated the table on the trained model, and looked at the dashboard. Here is an example:

The heatmap (figure 3.9) shows a high reconstruction error around mach 0.6, and at high alpha. Looking closer, this error is mostly on C_x. Plotting C_x against Mach shows *ludicrous* values (C_x should never be larger than 1, but here it peaks at 30).

Of course, these kind of anomalies are not very subtle, but it shows the model work. On a couple of these tests with Xavier, I was able to successfully see where the anomalies were. Data loading and inference take only dozen seconds or so, making the whole process rather quick.

The important thing to notice is how easy to see the errors are on the dashboard. By changing the parameters on the heatmap, it's possible to get a good view of what the model has detected. It also doesn't require knowledge of aerodynamics, except for confirming that high-error points are indeed anomalies.

These are promising results, that I hope to build upon in the time left.

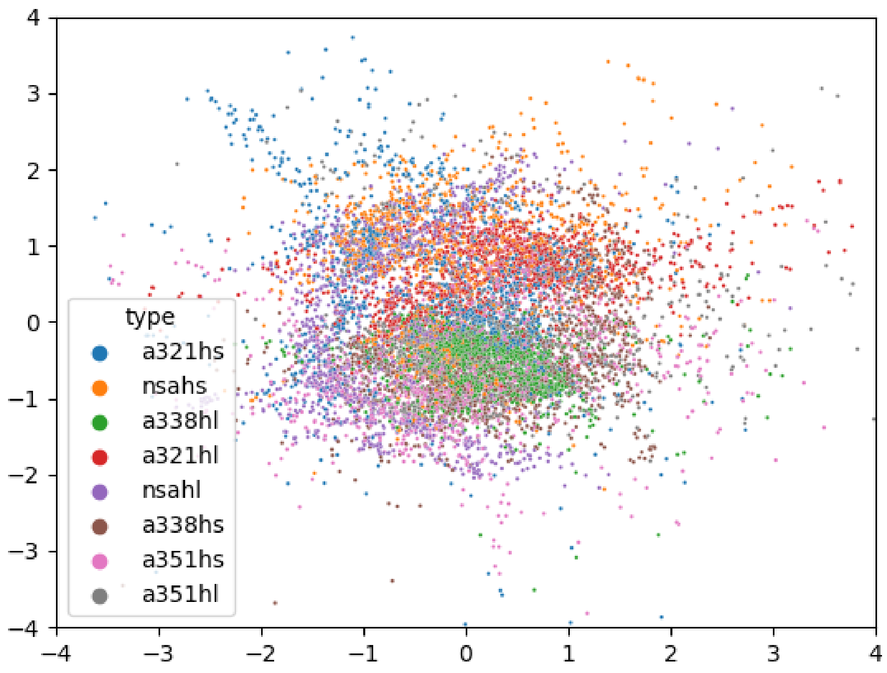
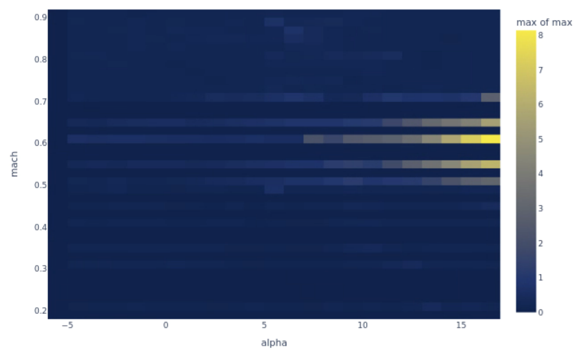
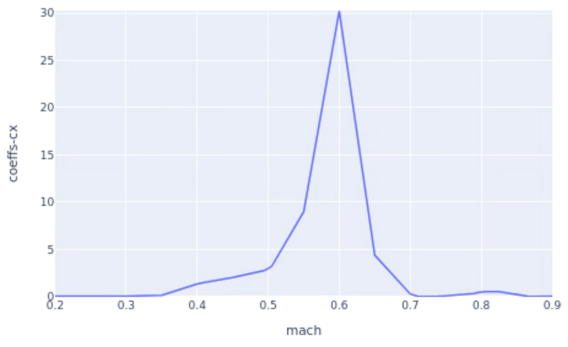


Figure 3.8: Latent space representation of the trained model, colored by type



(a) Error heatmap



(b) C_x vs Mach at high α

Figure 3.9: Heatmap and corresponding data for a fake anomaly

Chapter 4

Next steps and closing words

4.1 Future Work and Production

As I said, there is still time left and things to figure out.

For now, the priority is a comprehensive and representative testing dataset. It should contain normal and anomalous points from a variety of airplanes, all labeled. This would enable a qualitative assessment of performance, and may help improve the model a lot; as well as testing and comparing other models (GAN-based for instance).

I would also like to improve the dashboard, in terms of both design and usability. It could be better organized, faster, and show more information.

Even if the dashboard I created is useful and helps pinpoint anomalies, it is not automatic at all. The space still has to be explored by a human, even though it doesn't require much knowledge. We need the model to return a list of points or areas which are most likely anomalous. Just taking the points with an error above a certain threshold isn't ideal, as it might return too many to be useful, and might also miss the bigger picture. Something better would be some kind of clustering algorithm, which would take into account the error for each point.

Once the model reaches a satisfactory accuracy, it must be integrated into the large process of data generation and delivery, which takes the form of a pipeline. This also would require testing, as well as providing an explanation of how it works, and logs in case of errors.

This is outside the scope of this internship !

We could also imagine a user interface where feedback about the anomalies could be given (false positive, etc...) which could be used to train the model further (in a semi-supervised setting). This way, it would keep improving and learning from its mistakes.

4.2 What I've learnt

4.2.1 Working

This actually was my first internship, and first time in an office. The thing that struck me in the first weeks, was how I was not being told what to do. There was a general direction and a defined goal, but I had to figure out the path myself. This contrasted very much with school and university, where I knew exactly what I had to do, most of the time. This was both refreshing and a bit daunting; I sometimes felt I was not skilled enough for the job.

Not knowing where to go allowed me to explore, learn new things (about DL, aerodynamics), read papers, and experiment. Having a lot more time than for the projects I was used to, I could look at the details.

Another challenge was dealing with the *Planning Fallacy*, and getting organized alone. Virtually free from direct orders, lengthy meetings, or deadlines, I had to decide how to use my time. Historically speaking, I'm not great at that. I did plan my days, but consistently underestimated the sheer time things take: getting familiar with a technique or library, debugging obscure error messages, creating a simple interface, getting data ready, making slides for a presentation... All these seemed simple enough, but took me longer than I thought.

On such a long time period (relative to my past projects), I sometimes found it hard to stay motivated and focused. I sometimes felt stuck, trying to make trivial things work. Although I was in a great team and could ask any question I wanted, I still was the only one working on that subject. This made some days monotonous.

4.2.2 Aero and Deep Learning

Deep Learning is a relatively new and booming field. When I read about Anomaly Detection, it was clear that no consensus existed, there was nothing definitive about practice or theory (which is generally true for DL). This made choosing the method and having a clear view of the subject quite difficult.

I found impressive how a rather simple model like the VAE can be adapted to many different use cases, from images to tabular data and even time series. Of course, for each of these, the encoder and decoder are not the same, but I still think it's remarkable how data-agnostic the concept is.

Of course, I have also seen and learnt about aerodynamics, and airplanes in general. I had the chance to witness a flight test from the telemetry room, to visit the simulator and iron-bird room, and get an up-close view of the Final Assembly Line.

Bibliography

[1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P. Breckon. “GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training”. In: (2018). arXiv: 1805.06725 [cs.CV].

[2] Christoph Baur et al. “Autoencoders for Unsupervised Anomaly Segmentation in Brain MR Images: A Comparative Study”. In: (2020). arXiv: 2004.03271 [eess.IV].

[3] Carl Doersch. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606.05908 [stat.ML].

[4] Irina Higgins et al. “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework”. In: (2016). URL: https://openreview.net/forum?id=Sy2fzU9gl.

[5] Adrian Alan Pol et al. “Anomaly Detection With Conditional Variational Autoencoders”. In: (2020). arXiv: 2010.05531 [cs.LG].

[6] Lukas Ruff et al. “A Unifying Review of Deep and Shallow Anomaly Detection”. In: *Proceedings of the IEEE* 109.5 (May 2021), pp. 756–795. ISSN: 1558-2256. DOI: 10.1109/jproc.2021.3052449. URL: http://dx.doi.org/10.1109/JPROC.2021.3052449.

[7] Ronald Yu. “A Tutorial on VAEs: From Bayes’ Rule to Lossless Compression”. In: (2020). arXiv: 2006.10273 [cs.LG].

List of Figures

Appendices

Appendix A

Organization and hierarchy

The whole company is organized in teams in a tree-like structure. Each level in the tree is represented by a letter. The top levels are for example I for Engineering, H for Human Resources, F for Finance, and so on.

Acronym	Function
I	Engineering
IG	Flight Physics
IGA	Aerodynamics
IGAA	Aircraft Aerodynamics
IGF	Flight Dynamics and Performance
IGL	Loads and Aeroelasticity
IGW	Mass Properties
IY	Systems
IYC	Aircraft Control

Table A.1: Reference of teams cited in the report

Appendix B

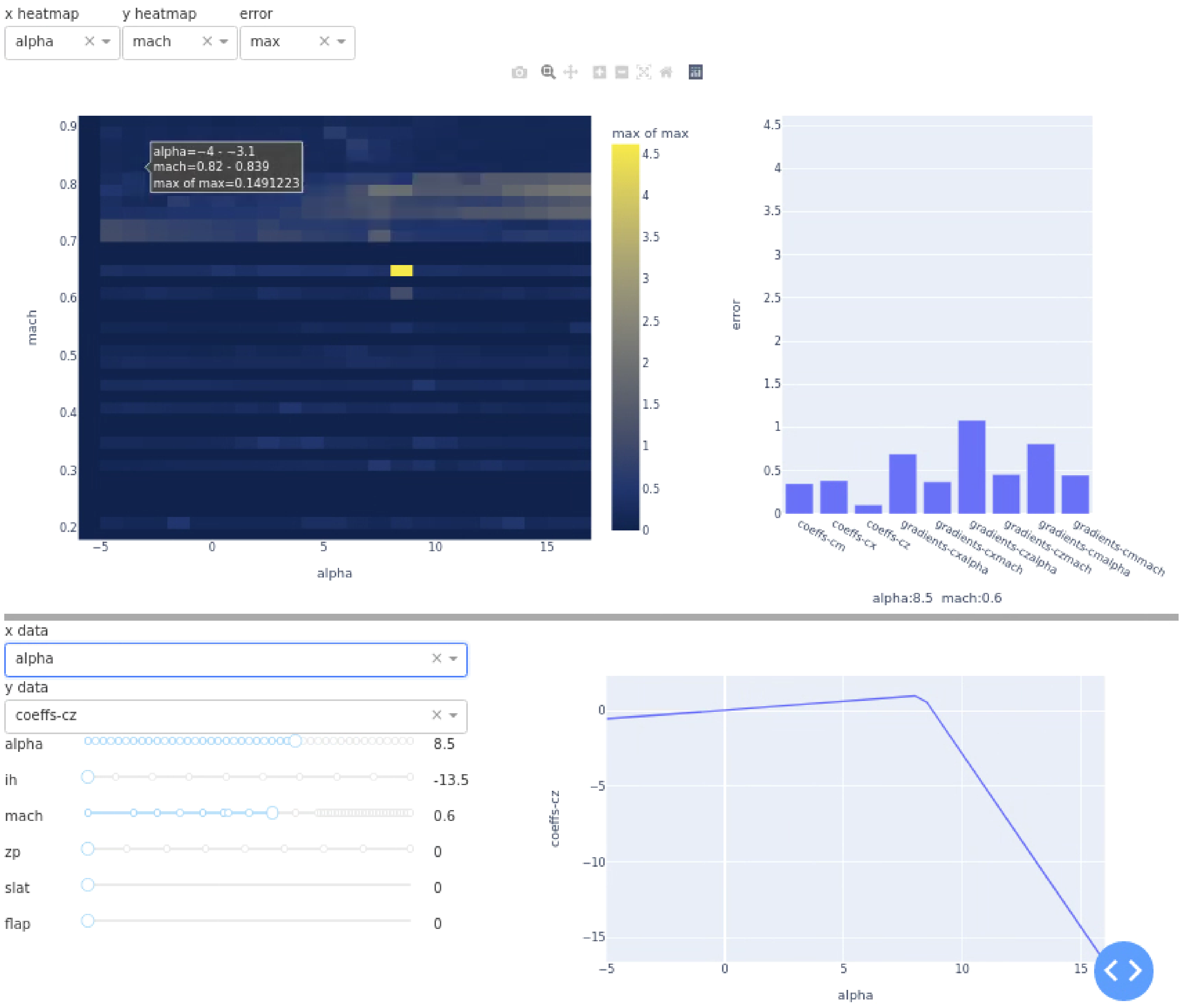
Dashboard

Below (figure B.1) is an example of what the dashboard shows once some data has been evaluated on the model. It was coded with Dash, which is both very powerful, with a lot of options, types of graph and inputs; and kind of a mess to code. The styling requires using CSS, which I'm not an expert in at all, that's why it doesn't look great. It is divided between top part showing two graphs with the error made by the model, and the bottom part, made to simply plot the (original) data.

At the top, a heatmap shows the maximum reconstruction error for the whole dataset, with respect to two parameters, by default α and Mach (which the user can choose). A small visual issue with this visualisation, is that parts of the flight domain with no data look the same as with no error. This isn't too important though. By clicking on a part of this heatmap, the corresponding parameter values are selected for the right and bottom part.

The right part shows the max error on the different aerodynamic values, for the region selected on the left. This is a way of seeing which parameters is most likely off.

On the bottom, the graph is simply here to plot the data, and see whether it looks good. The sliders are automatically updated by clicking on the heatmap, and the rest can be chosen. Every flight parameter / aerodynamic value combo can be chosen by the user.



x heatmap: alpha
y heatmap: mach
error: max

alpha=−4 - −3.1
mach=0.82 - 0.839
max of max=0.1491223

alpha:8.5 mach:0.6

x data: alpha
y data: coeffs-cz

alpha 8.5
ih -13.5
mach 0.6
zp 0
slat 0
flap 0

Figure B.1: Dashboard interface