University of Groningen

Master Thesis in Astronomy

Mitigation of electrical standing waves using machine learning

Author:
Roy O.Y. Bos
s3162605

Supervisor:
dr. Russell F. Shipman

Collaborator:
dr Youngmin Seo

July 2021

Abstract

Heterodyne spectrometers are instruments that are commonly used to observe emission lines in the far-infrared, or THz regime. They can provide a high frequency resolution, which allows them to resolve the shape of an emission line and in turn derive the physical properties of the observed object. Unfortunately, spectrometers that use HEB mixers have strong instrument noise, dominated by electrical standing waves (ESWs). ESWs distort the baseline of a spectrum and are hard to remove using analytical methods. The currently used method by Kester et al. [2014] is sufficient in most cases, but of limited use when ESWs change shape rapidly.

As an alternative, we present a pattern recognition approach to remove ESWs from a spectrum. The purpose of this approach is to allow for more flexibility in estimating the shape of a baseline, such that it can be used on data with rapidly changing ESWs. We use a principal component analysis in addition to two neural networks for small corrections.

We apply our method to observations from the HIFI spectrometer, an instrument aboard the Herschel Space Observatory. We show that the algorithm is able to effectively remove nearly all ESWs in the data used for training. On observations of the [CII] line by Langer et al. [2010], it shows varying results; in some of the [CII] observations, the ESWs are up to three times smaller; in other observations, the ESWs are reduced but our algorithm produces erratic features in the baseline.

Our method is able to significantly improve the baseline prediction in cases which are poorly described by the previous method. However, improvements are necessary as the algorithm is somewhat unpredictable, and it will need to be tested on data sets other than those from HIFI.

Contents

Chapter 1

Introduction

Spectral observations play an important role in astronomy. Among other things, they can be used to determine the composition and phases of the interstellar medium (ISM). In the Earth's atmosphere, ro-vibrational lines from water vapour absorb a considerable part of the infrared radiation, so to observe infrared radiation in the ISM it is often necessary to do observations in space. Alternatively, observations can be conducted using high-altitude planes, or using high altitude balloons. Examples of such instruments are GREAT(1) aboard SOFIA(2), STO2(3), and the upcoming GUSTO(4) mission.

The HIFI spectrometer aboard the Herschel Space Telescope was a far-infrared spectrometer with a very high frequency resolution. It could resolve different objects along the line of sight, both spatially and by their velocity. The GOT C+ survey by Pineda et al. [2013] made use of the HIFI instrument to perform high resolution observations of the 158 μm [CII] emission line, throughout the Galactic disk. The [CII] line is the strongest line in the far-infrared and can be found in many environments in the ISM.

Unfortunately, spectroscopic instruments produce many artifacts in the data which can be hard to distinguish from real astronomical sources. One type of artifacts, called electrical standing waves (ESWs), can be found in heterodyne instruments using hot electron bolometer mixers (HEBs). ESWs are large and relatively broad waves extending over the entire spectrum, and are usually the main component of the baseline. It is crucial to have a good baseline removal method that removes these standing waves. Several methods have been developed to deal with ESWs, for example by Kester et al. [2014], and Higgins and Kooi [2009]. However, these were mostly either specific to the instrument, or required ESWs to be relatively stable. Although ESWs were indeed stable for HIFI, this is not necessarily the case for other heterodyne spectrometers.

In this thesis we aim to provide an additional, general algorithm for reducing ESWs, which could be applied to any instrument with rapidly changing baselines. We apply a principal component analysis to describe the ESWs, in combination with two neural networks.

Project outline

The first chapters give an introduction on the relevant topics. For this project we make use of data from HIFI, a far-infrared spectrometer on board of the Herschel space observatory. Chapter 2 goes into detail about some of the applications of far-infrared spectroscopy, and HIFI observations in particular. Next, Chapter 3 talks about the basics of heterodyne spectroscopy and the components of the HIFI instrument. Chapter 4 describes how the HIFI instrument produces ESWs, and explores the previous methods that have been developed to deal with ESWs. The next two chapters give a summary on neural networks. Chapter 5 explains the basics and how they work, while Chapter 6 details the training process and necessary data processing steps.

In Chapter 7 we outline the methods that we applied, and how we preprocess the data. The results of the method are shown in Chapter 8. In Chapter 9 we compare the results on [CII] observations with the results from the pipeline method, and we quantify the performance of our algorithm using mock data. Lastly, the appendix in Chapter 12 includes the script used to preprocess the data in the HIFI pipeline software HIPE [Ott et al., 2010]. The code used to develop the algorithm, including the processed data, is available on GitHub(5)

(1) German REceiver for Astronomy at Terahertz Frequencies
(2) Stratospheric Observatory for Infrared Astronomy
(3) Stratospheric Terahertz Observatory
(4) Galactic/extragalactic Ultra-long duration balloon Spectroscopic Terahertz Observatory
(5) https://github.com/royb98/eswremove.git

Chapter 2

Astronomical applications

The far-infrared regime is home to a large number of spectral lines. These lines are not confined to a single frequency, but have various widths and shapes. The HIFI instrument was a far-infrared spectrometer with a high enough frequency resolution to resolve the shape of these far-infrared lines. As is detailed in Section 2.4, this high frequency resolution provides a wealth of information about the observed astronomical object, as the line shape is determined by the physical properties of that object.

The GOT C+ survey by Pineda et al. [2013], which made use of Herschel's HIFI instrument, performed high frequency resolution observations of [CII] in the galactic disk. The 158 μm [CII] line is primarily used to determine the distribution of molecular hydrogen in the ISM, which is the subject of Section 2.2. While molecular hydrogen is the main source of star formation, it is not easily observable directly. It can, however, by inferred from [CII] observations.

2.1 The interstellar medium

The interstellar medium (ISM) comprises all regions in between the stars, and is a dynamic environment where matter and radiation continually interact with themselves and each other. Figure 2.1 by Groppi et al. [2009] shows the different components that can be found in the ISM, and how they transform to one another. Starting from a neutral, molecular cloud, gravitational instabilities will lead to the formation of stars. These stars lead to ionization of nearby gas clouds, but most importantly enrich the contents of the ISM at the end of their life. Metals, which are created in the high-pressure environments of the stellar cores, become increasingly abundant in the ISM over time. Thus the composition of the ISM largely determines the evolution of the stellar population.

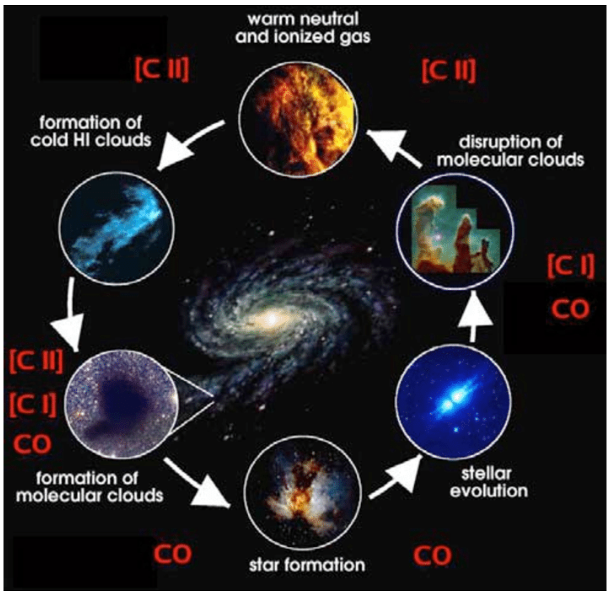

warm neutral and ionized gas
[C II]
[C II]
formation of cold HI clouds
disruption of molecular clouds
[C I]
CO
[C II]
[C I]
CO
formation of molecular clouds
stellar evolution
CO
star formation
CO

Figure 2.1: Illustration of the various components in the ISM, and how they evolve over time [Groppi et al., 2009]. Carbon emission, in different forms, is found in many of these environments.

Most of the matter in the ISM can be found in the form of neutral atomic hydrogen and helium, and neutral molecular hydrogen [Groppi et al., 2009]. Unfortunately, molecular hydrogen and atomic helium are difficult to detect in most environments, even though molecular hydrogen is responsible for the majority of the star formation. Atomic hydrogen can be relatively easily observed in cold clouds using the 21 cm line, but this line is not affected by the density of the gas [Pineda et al., 2013]. Hence, it is hard to distinguish cold atomic clouds from the warm neutral medium using this 21 cm line. Some hot, atomic hydrogen gas can be observed mainly through 656 nm Hα emission, but it requires high temperatures that are not found in any of the cold gas clouds, but rather in hot, ionized HII regions [Anderson et al., 2009]. Molecular hydrogen also does not typically have observable emission lines in cold gas clouds; at typical cold gas cloud temperatures of 10 to 30 K, carbon monoxide (CO) emission is the dominant coolant, while molecular hydrogen emission is dominant only at temperatures above 1000 K [Burion, 1992]. Therefore, in order to derive the properties of the ISM, it is necessary to use other elements as tracers of gas clouds.

To trace molecular hydrogen, often carbon monoxide is used (CO), but a large fraction of molecular hydrogen resides in so-called CO-dark clouds with no detectable CO emission. For such CO-dark clouds, [CII] emission provides an alternative for detecting molecular hydrogen.

2.2 Carbon emission

Carbon, which is a relatively abundant element, can be observed in several forms. As shown in Figure 2.1, it can be found in its neutral state in the form of carbon monoxide (CO), or in its ionized form through [CI] and [CII] emission. Carbon monoxide is mostly observed as ^{12}CO, but it has some other observable isotopologues as well. Regions dominated by far ultraviolet radiation can dissociate CO into atomic carbon and oxygen, and photoionize the carbon. Such regions are often found at the surfaces of clouds, which are exposed to high energy radiation from nearby stars. The [CII] line at 158 μm can be used to trace the warm ionized medium, warm and cold diffuse atomic medium, and warm and dense molecular gas. [CII] emission is especially useful in clouds which contain molecular hydrogen but cannot be observed in CO emission [Langer et al., 2010]. Such regions are called CO-dark H_2 clouds.

The [CII] carbon fine structure line is the strongest line found across the universe, and accounts for about 1% of all the energy emitted in the far-infrared [Rodriguez-Fernandez et al., 2006]. It is therefore an important coolant in the ISM through line emission, especially in the cold neutral medium [Wolfire et al., 2003], at molecular cloud surfaces, and in dense photodissociation regions [Hollenbach and Tielens, 1999]. The widespread influence of [CII] can be explained by the relatively high abundance of carbon and its low ionization potential of 11.26 eV, which is lower than that of hydrogen.

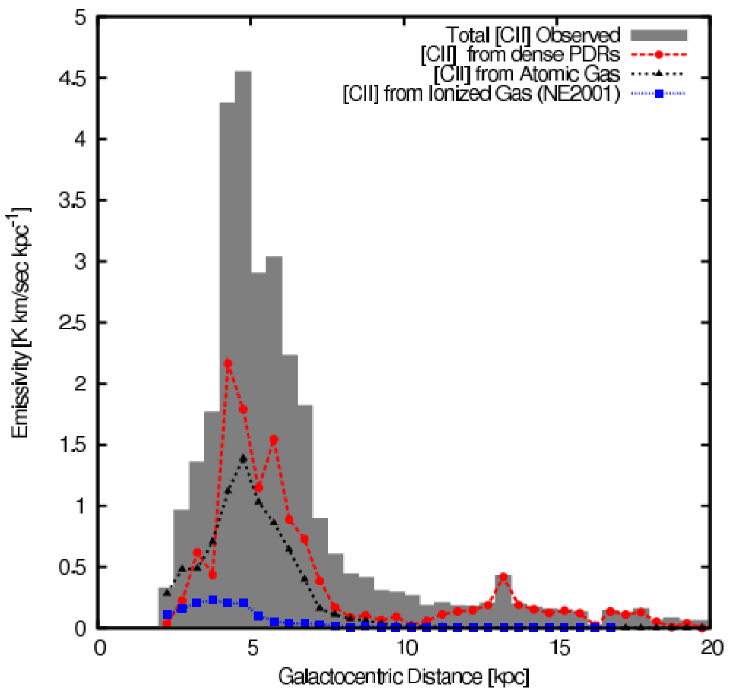


Figure 2.2: Azimuthally averaged distribution of [CII] in the Milky Way [Pineda et al., 2013].

Figure 2.2 shows the distribution of [CII] in the Milky Way as a function of distance from the center. In the Milky Way, [CII] is mostly found in the spiral arms. Pineda et al. [2013] estimates that 47% of the emission originates from photon dominated regions, 28% from CO dark H_2 gas, 21% from cold atomic gas and 4% from ionized gas. Because it traces a large part of the hydrogen mass within the ISM, [CII] can be used as a rough indicator for star formation. For instance, [Smith et al., 2016] uses the ratio of [CII] to far-infrared continuum to estimate the star formation rate in other galaxies.

2.3 Other line emission

Another important element observed in the infrared is nitrogen, which is seen by its fine structure lines at 122 and 205 μm. Because [CII] can be found in many different phases of the ISM, observations of [CII] alone cannot distinguish ionized hydrogen regions from neutral hydrogen regions. The ionization potential of nitrogen is 14.5 eV, which is higher than that of hydrogen and carbon. That implies that [NII] emission cannot be found in neutral hydrogen regions, and can thus be used to differentiate between fully and weakly ionized gas [Goldsmith et al., 2015]. Other important lines in the far-infrared include 63 and 148 nm [OI], [CI], and higher J [CO] emission.

2.4 Line shape

The line shape and width is affected by many physical effects. These effects include pressure broadening, natural broadening due to the uncertainty principle, and Doppler broadening due to thermal and other kinetic motion. These line broadening effects are all dependent on the environment from which the line was emitted. On top of line broadening effects, the observed line of sight may contain multiple astronomical objects with the same line emission. Depending on the velocity of each object along the line of sight, each line is slightly shifted due to the Doppler effect. This results in multiple emission lines close together, or possibly even overlapping. High frequency resolution spectrometers such as HIFI can be used to infer all this information.

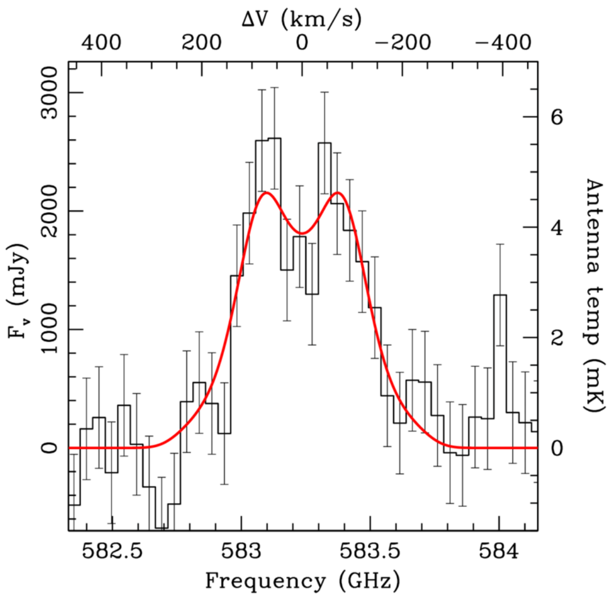


Figure 2.3: A velocity resolved [CII] line, observed by Rhoads et al. [2014] using the HIFI instrument. This line is emitted by S0901, a rotating galaxy at redshift $z = 2$. The bottom x-axis shows the observed frequency, and the top x-axis shows the corresponding rotational velocity.

Unfortunately, spectral observations generally cannot provide much spatial information, which limits their application for observations of other galaxies. However, the distribution of line emission in the Milky Way can tell us about the global properties of other galaxies. As previously stated, the [CII] line is the strongest line in the far-infrared, and is therefore relatively easily visible in faint, high redshift galaxies. Rhoads et al. [2014] demonstrates that velocity resolved [CII] emission can be effectively used to study the kinematics of those galaxies. An example of this is given in Figure 2.3. Galactic [CII] emission is mainly seen in the disk. From the perspective of the observer, the outer ends of this disk rotate in opposite directions, causing a Doppler shift in the frequency of the emission line. As the middle part of a galaxy is optically thick, the [CII] emission at the far side of the galaxy is slightly obscured. This results in two distinct peaks. By fitting a disk model to these observations, one can infer different parameters such as the peak rotation speed v_d and mass surface density Σ.

2.5 GUSTO

The Galactic/Extragalactic ULDB Spectroscopic Terahertz Observatory, or GUSTO, is an upcoming mission that is planned to launch this year (2021). The observatory will be suspended from an Ultra-Long Duration Balloon (ULDB). The ULDB will float at an altitude of about 40 km above Antarctica, for a duration of 100 to 170 days. The observatory itself features a telescope of one meter in diameter, and three instruments which will measure terahertz radiation. SRON and the technical university of Delft will supply the HEB camera's, a local oscillator, and a phase diffraction grating.

GUSTO aims to provide simultaneous observations of [CII], [NII] and [OI], at 158, 205, and 63 μm, respectively. The observations will provide a fully sampled map of the Milky Way disk and nearby Large Magellanic Cloud, and will thus provide a more complete view of the different phases of the ISM. Although the HIFI GOT C+ survey by Langer et al. [2010] also mapped the milky way disk in CII, it had a sparse spatial mapping and did not provide observations which were spatially connected.

Contrary to HIFI, which operated in a stable environment at the L_2 Lagrange point in space, GUSTO will operate in the Earth's atmosphere. Thus the instrument is more prone to move around. Movements of GUSTO's electrical components will cause the ESWs to be less stable over time, underlining the importance of a correction method that functions on rapidly changing ESWs.

Chapter 3

The HIFI instrument

While developing and testing the algorithm, we made use of data from HIFI. The HIFI instrument provided high frequency resolution observations and had relatively stable ESWs, which makes it ideal for developing and testing our algorithm. Furthermore, the HIFI pipeline already featured a method to get rid of ESWs, which we can use for comparison. This chapter will give an overview of the basic observing principles of HIFI (Section 3.2), its most important instrument components (Sections 3.3 to 3.5), and the instrument pipeline used to process the data (Section 3.6).

3.1 Herschel Space Observatory

The Herschel Space Observatory was an ESA cornerstone mission to observe the Universe at far-infrared wavelengths. The satellite was launched in 2009 and put at the L_2 Lagrange point beyond Earth's orbit, where it remained active until 2013. At that time, all of the liquid helium that was used to cool the instruments in the cryostat had been depleted, so the instruments on board could not function at their operating temperatures below 4 K. The telescope was a Cassegrain type with a primary mirror measuring 3.5 meters in diameter. There were three instruments aboard the satellite; the Photodetector Array Camera and Spectrometer (PACS), Spectral and Photometric Imaging REceiver (SPIRE), and the Heterodyne Instrument for the Far-Infrared (HIFI).

The HIFI instrument allowed for spectroscopy in 14 frequency bands and in two orthogonal polarizations simultaneously. The instrument made spectroscopic observations between 0.5 and 1.9 THz, or 0.61 mm and 0.16 mm. Unlike previous far-infrared instruments, it featured a very high frequency resolution. This allowed it to measure the shape of emission lines, and to resolve the velocity.

3.2 Heterodyne spectroscopy

Continuum radiation in the sub-millimeter range can be detected using bolometers. These measure the power of incoming radiation through its heating effects on a material with a temperature dependent resistance. However, line emission in the sub-millimeter range is harder to measure. For sub-millimeter spectroscopy, the weak sky signal has to be amplified before it can be measured with a spectrometer [David Teyssier, ESA, 2016]. Unfortunately, amplifiers are not available in the sub-millimeter range, but they are available at lower frequencies.

One way to allow amplification of the sky signal is by lowering its frequency. This can be done by making use of the heterodyne principle; by mixing the sky signal with the signal of a local oscillator (LO), we get a combined signal. The frequency of the combined signal is known as the intermediate frequency (IF). It is the difference between the two signals, as given by Equation (3.1). By setting the frequency of the LO to be very close to the frequency of interest, the resulting IF will be of a much lower frequency because the difference between the signals is small. Since the frequency of the LO is known, in principle the IF retains all initial information from the sky signal even though it has a much lower frequency. However, this is not the case for HIFI, as it uses dual-sideband mixers (DSB).

$$\nu_{\mathrm{IF}} = \|\nu_{\mathrm{sky}} - \nu_{\mathrm{LO}}\| \tag{3.1}$$

In dual-sideband mixers, there is an ambiguity in the frequency. A single value of the IF can correspond to two source frequencies; an IF value of 1 GHz can correspond to a source frequency both 1 GHz above or below the LO frequency. Source frequencies that are higher than the LO frequency are part of the upper sideband (USB) and will appear in ascending order. Source frequencies below the LO frequency are part of the lower sideband (LSB) and will appear reversed, in descending order. HIFI's dual-sideband mixers measure the combination of both sidebands. In order to determine which spectral lines are above or below the LO frequency, we would need to change the LO frequency. Spectral lines in the two sidebands would then shift in opposite directions as is illustrated in Figure 3.1.

There are also single-sideband mixers (SSB), which suppress either the LSB or the USB in order to get rid of the ambiguity in frequency. Thus no information is lost, with the added benefit of requiring half the bandwidth compared to DSB and not requiring the demodulator to have the exact same frequency and amplitude. Still, DSB might be preferred over SSB as long as there is no confusion about the line frequencies, as it requires less complicated hardware.

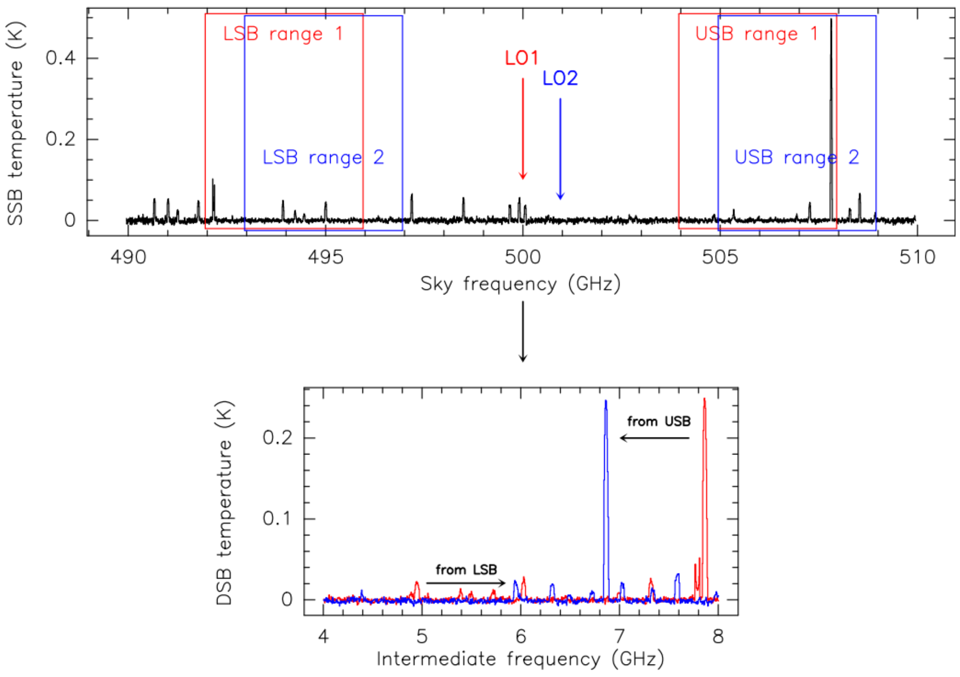


Figure 3.1: Illustration showing how spectral lines in the upper and lower sidebands (USB and LSB) would shift depending on the LO frequency [David Teyssier, ESA, 2016]. The source signal is shown in the top figure and the IF signal is shown in the bottom figure. By increasing the LO frequency from LO1 (red) to LO2 (blue), the spectral lines in the USB will have a lower IF. On the other hand, spectral lines in the LSB range will have a higher IF.

By changing the LO frequency, we change the frequency range of the LSB and USB. This allows us to sample a large part of a source's spectrum. Measuring the IF allows us to infer what the LSB and USB would be. However, the IF is not completely clean; it also contains some instrument noise. To calibrate the on-source signal (ON), it is usually compared with an off-source signal (OFF). Since the OFF signal should not have any notable sources, the OFF spectrum mostly consists of noise from the instrument such as heat noise. Thus, taking the difference between the ON and OFF observations removes most of this internal noise while keeping the source measured in the ON observation. This differencing technique was also applied in the HIFI instrument, where the focal plane chopper was used to quickly switch between ON and OFF (see Section 3.5).

3.3 Mixers

The HIFI instrument allowed for high frequency resolution heterodyne spectroscopy. Its frequency range covered 480-1272 and 1430-1906 GHz, both in the far-infrared. This range is covered by 7 mixer bands (see Table 3.1). Each band in HIFI had either a superconductor-insulator-superconductor (SIS) mixer, or a hot electron bolometer (HEB) mixer. Both mixer types operated at very low temperatures around 4 K to allow superconduction. These mixers not only mixed the sky signal and LO signal, but also amplified it. Our algorithm tries to correct the electrical standing waves (ESWs) that appeared in the higher bands between 1.4 and 1.9 THz, where HIFI used a hot electron bolometer (HEB). In these bands, ESWs were particularly strong.

Table 3.1: Properties of all 14 frequency bands of HIFI [David Teyssier, ESA, 2016]. Here, the HPBW is the Half-Power BeamWidth. We only use the bands that use a Hot Electron Bolometer (HEB) mixer.

Band	LO frequency range (GHz)	Mixer type	IF bandwidth (GHz)	Typical HPBW at band center (arcsec)
1a	487.5–553.5	SIS	4.0 GHz	40.4
1b	562.5–628.5	SIS	4.0 GHz	35.3
2a	634.0–718.0	SIS	4.0 GHz	31.1
2b	722.0–794.0	SIS	4.0 GHz	27.7
3a	807.0–852.0	SIS	4.0 GHz	25.3
3b	866.0–953.0	SIS	4.0 GHz	23.1
4a	957.0–1,053.0	SIS	4.0 GHz	20.9
4b	1,054.5–1,114.0	SIS	4.0 GHz	19.4
5a	1,116.2–1,236.0	SIS	4.0 GHz	17.9
5b	1,235.0–1,272.0	SIS	4.0 GHz	16.8
6a	1,430.0–1,558.0	HEB	2.4 GHz	14.1
6b	1,578.0–1,698.0	HEB	2.4 GHz	12.8
7a	1,701.0–1,794.0	HEB	2.4 GHz	12.0
7b	1,793.0–1,902.0	HEB	2.4 GHz	11.4

3.4 Spectrometers

As shown in Table 3.1, HIFI had an IF bandwidth of 4 GHz for bands 1 through 5, and 2.4 GHz for bands 6 and 7. To sample these bandwidths, HIFI had two types of spectrometers; the wide band spectrometer (WBS) and the high resolution spectrometer (HRS). The WBS had a resolution of 1.1 MHz, or 0.4 $\mathrm{km\,s^{-1}}$, while the HRS could provide a frequency resolution of up to 0.125 MHz [De Graauw et al., 2010]. We apply our algorithm to observations from the WBS. The WBS covered the complete IF bandwidth (either 4 or 2.4 GHz), while the HRS could sample any part of the IF bandwidth with high resolution. Both spectrometers divided the IF signal into four subbands, although not all of these were available for all bands. To obtain a single observation over the sampled frequencies, the separate observations from each subband have to be stitched together.

To increase redundancy, two spectrometers were available of both types, amounting to a total of four spectrometers. All spectrometers were fully independent, but they could do measurements at the same time. As the mixers were sensitive to polarization, the spectrometers measured different polarizations; a horizontal and a vertical polarization.

3.5 Optics

The focal plane chopper was a mirror within the HIFI instrument which could tilt in one direction. Depending on its orientation, the direction from which light was received would change. Thus, the chopper could change the beam direction while the telescope itself would remain stationary. The chopper allowed to switch between five configurations [Higgins, 2011]; one on-source configuration (ON), two off-source configurations (OFF) where the telescope beam is slightly off-center from the source, and two configurations to look at the HIFI internal hot/cold loads for calibration. The on-sky separation between the ON and OFF configurations was roughly 3 arc minutes, and switching between them could be done at a rate of up to 4 Hz [David Teyssier, ESA, 2016].

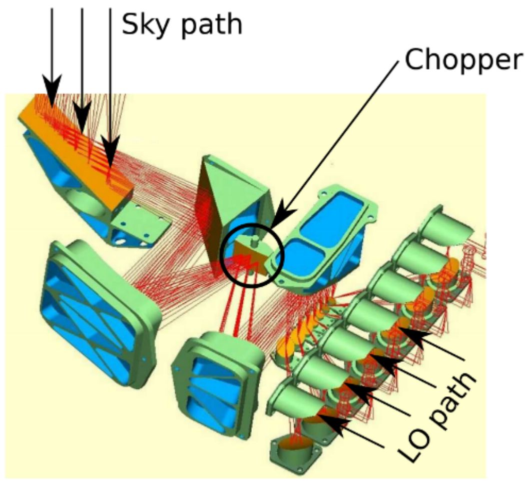


Figure 3.2: Schematic showing the relay optics of HIFI [Higgins, 2011]. Light from the Herschel telescope enters from the top left. Depending on the orientation of the chopper, the spectrometers receive light from different on-sky positions or on-board calibration sources.

Even though Herschel was built to observe sources in the range of 50 to 100 K, the primary mirror had a temperature of around 80 K. Thus, the mirror added a significant amount of background heat radiation to the sky signal. To mitigate this contribution, the signal has to be calibrated using the differencing technique discussed in Section 3.2. The chopper is used to quickly switch between ON and OFF. Taking the difference between the ON and OFF mitigates most of the internal heat noise from the instruments.

3.6 Data pipeline

In order to obtain science-ready spectra, the readings from the spectrometers first have to pass through a pipeline to calibrate and correct instrument artifacts like electrical standing waves (ESWs), discussed in Chapter 4. For this thesis, we use spectral observations from the HIFI instrument. The HIFI pipeline consists of three levels, which each serve a different purpose. The data processing was done using the HIPE software [Ott et al., 2010], and a full review on the pipeline steps is given by Shipman et al. [2017]. The correction of ESWs is done at the end of level 1 of the pipeline, so for our algorithm we use level 1 data (excluding the ESW correction of course). This data uses the intermediate frequency (IF) instead of the sky frequency. The Jython script that was used to preprocess the data is given in the appendix, Chapter 12.

At level 0 and 0.5 of the pipeline, the raw spectrometer output is converted to frequency and flux. Since the two spectrometer types (WBS and HRS) were completely different, they each use different pipelines.

The main objective of level 1 of the pipeline is to calibrate the flux and frequencies. In this step, the measurements of the hot and cold loads as well as their efficiencies are used to adjust the antenna temperature scale, and to find the system temperature. A radiometric weight is applied to each channel, depending on this system temperature. Furthermore, the load measurements as well as the OFF measurements are subtracted to calibrate the flux. Before this subtraction, the OFF measurements are smoothed so that they do not introduce too much noise. Next, the frequencies are adjusted for the Doppler shift due to the motion of the satellite. Lastly, the electrical standing waves are corrected using the method developed by Kester et al. [2014].

In the last level of the pipeline, level 2, the data is simplified for scientific use. Most importantly, it converts the dual-sideband (DSB) IF spectra into two single sideband (SSB) spectra; one for the upper sideband (USB) and one for the lower sideband (LSB). After the frequency conversion, the frequencies are converted to have a uniform grid spacing, while conserving the total power.

Chapter 4

Electrical standing waves

Electrical standing waves (ESWs) are broad, wave-like features that modify the baseline over the entire spectrum. Because ESWs are not perfectly sinusoidal and their period, amplitude and offset are all variable, they are hard to model analytically. This chapter describes the physical origin of ESWs within the instruments, and what methods have been proposed to get rid of them.

4.1 Origin and appearance in stability spectra

The four highest bands of HIFI (6a, 6b, 7a, 7b) make use of hot electron bolometer (HEB) mixers. In these bands, significant electrical standing waves (ESWs) are present in the signal. They are mainly produced by reflections inside the transmission cable between the HEB mixer and the first low noise amplifier (LNA). The lack of an insulator between the mixer and the LNA caused a mismatch in the impedance of both instruments [Higgins and Kooi, 2009]. As a results of this, not all current coming into the mixers is absorbed. Part of the electrical signal is instead reflected back. A part of this backward propagating signal is reflected back again and does get absorbed by the mixer. Thus the mixer measures the superposition of the sky signal and its multiple reflections.

The signals interfere with each other either constructively or destructively, producing standing waves. These waves are nearly sinusoidal, but not exactly. The peak amplitudes of the ESWs oscillate rapidly between each observation. Consequently, taking the differences between ON and OFF observations does not remove the ESWs, as the ESWs have different amplitudes in both observations. After subtracting the OFF measurements, we are still left with the differences between the ESW signals.

Figure 4.1 shows multiple integrations from level 1 HIFI stability data. Stability data consists of observations of source-free regions. As such, stability spectra do not have any strong astronomical signals, and are thus perfect for examining the ESWs. For HIFI, the stability measurements were used to inspect the temporal stability of ESWs. Over time, ESWs Each integration in Figure 4.1 shows clear sinusoidal-like patterns and offsets caused by electrical standing waves. The amplitudes and offsets are different between scans and also change as a function of intermediate frequency.

ESWs are produced after the sky signal is mixed with the signal from the LO, so their shape does not depend on the frequency of the LO signal. However, ESW do change as a function of the strength of the LO signal, as this influences the impedance of the bolometer. Moreover, the ESWs change as a function of the IF, as can be seen in Figure 4.1. The ESWs tend to have larger amplitudes and offsets at the edges of the spectrum.

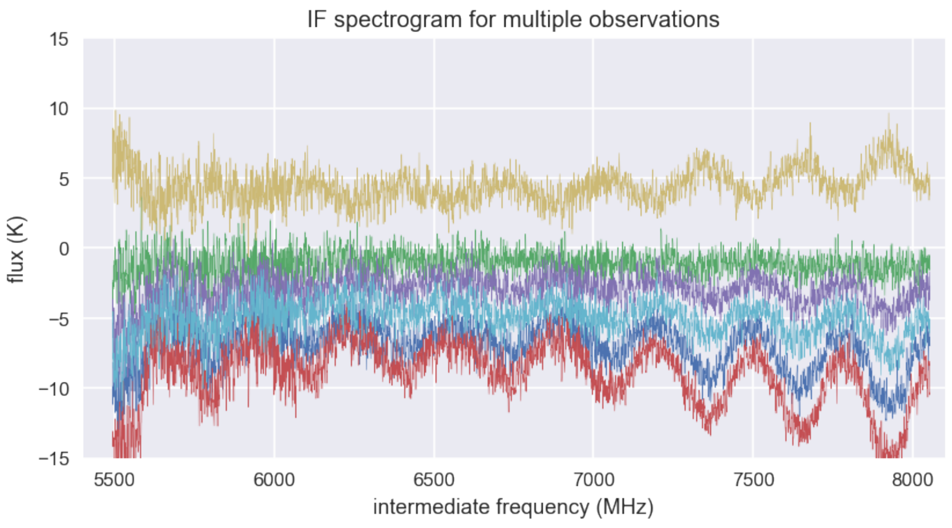


Figure 4.1: Intermediate frequency spectrum for multiple integrations of level 1 HIFI stability spectra. The fluxes in the spectra are allowed to be negative, as we are looking the differences between two signals.

4.2 Approaches to correct ESWs

4.2.1 Physical modeling

Higgins and Kooi [2009] proposes a method that physically models how the mixer current produces ESWs. It makes use of the observation that the difference in mixer current ΔI_{mix} between two calibration phases is correlated to the amplitudes of the ESWs. Also, they conclude that the ESW profile is stable for a given mixer current I_{mix} and current difference ΔI_{mix}. Although this was the case for HIFI, this correlation may not be applicable to other instruments with rapidly fluctuating standing waves, such as the GREAT spectrometer on board of SOFIA [Richter et al., 2015]. Besides that, this method was also not applicable for HIFI, as the mixer current could not be monitored fast enough for this method to work.

In the same paper, Higgins and Kooi [2009] also propose an algorithm that physically models the HEB mixer, the first low noise amplifier, and the coaxial cable that connects them. This method showed excellent results, but would be too computationally extensive for use within a pipeline. To add to this, a physics based method would also be very dependent on the instrument.

4.2.2 Pattern recognition

The method that is currently part of the HIFI pipeline can be considered to use a form of pattern recognition, and is detailed by Kester et al. [2014]. They find that ESWs in HIFI data appear to consist of a limited number of multiplicative families. Thus, to some approximation, any ESW can be described as a scaled version of a representative ESW template. In order to find these templates, they use HIFI stability spectra. As is explained in Section 4.1, stability spectra can be used to represent ESWs. After finding the set of templates, they use Bayesian evidence to match a template with a new ESW. Subsequently, they find the scaling factor to match the size of the template to the ESW. In addition to subtracting a baseline template, they also subtract a 4 node spline to get rid of any broad continuum radiation. The ESW correction method by Kester et al. [2014] requires that ESWs in observations do not have a unique profile, as their profile has to correspond to one of the templates in their data set. Although Higgins and Kooi [2009] supports that the ESW profile was indeed very stable for HIFI, this may not necessarily be true for other instruments.

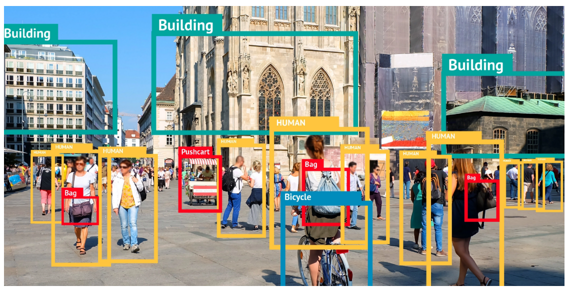
Our algorithm applies a somewhat similar approach to Kester et al. [2014]. Similar to their method, we use HIFI stability spectra to represent ESWs. We train a PCA to decompose these spectra into a set of eigenvectors. In order to remove ESWs from a new spectrum, we do a PCA fit using the same eigenvectors. Thus, we use a linear combination of multiple eigenvectors to describe an ESW, whereas the pipeline method only uses a single ESW template. This allows for more flexibility in the shape of the predicted ESWs, as the shape can be completely different from any of the ESWs in the stability data. To improve the results of the PCA, we apply two neural networks; a convolutional neural network that removes spectral features based on their shape, and a neural network that does small corrections on the PCA.

Chapter 5

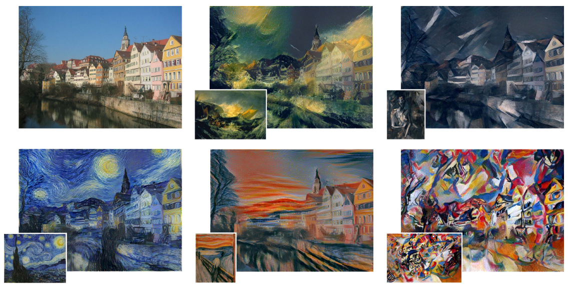
Neural networks

An artificial neural network is a computing system that, without being given explicit instructions, learns to solve a problem. Neural networks have widely varying applications; its uses range from image and speech recognition, to text prediction, or image and audio style transfer. Two such examples are given in Figure 5.1. Both examples use a convolutional neural network in order to recognize patterns in a 2-dimensional image. In the first example [Hafellner, M., 2019], objects in the image are recognized and labeled, while in the second example the style of an image is changed to match the style of another image [Gatys et al., 2016].

Contrary to other types of programming, a neural network is not explicitly told how to obtain the solution. Instead, after making a guess for the answer, it is taught what the actual solution should be, and the network changes its parameters such that its result gets closer to the desired result. This training process is explained in Chapter 6. By repeating this process many times, the network learns. After a neural network has been trained on a large sample of data, it should give the correct results to new but similar data. This chapter will give a basic summary on some of the relevant concepts related to neural networks.



(a) Image recognition identifies and labels objects or people from images. From Hafellner, M. [2019]



(b) Image style transfer combines the style of one image with the content of another image. From Gatys et al. [2016].

Figure 5.1: Two applications of neural networks.

5.1 Structure of a neural network

A basic schematic of an artificial neural network is shown in Figure 5.2 [Malik, F., 2019]. A neural network consists of neurons, which receive input values and produce output values. The neurons are connected by synapses which allow the neurons to pass on their values to other neurons. The value that a neuron holds is better known as its activation.

Neurons are arranged in layers, and neurons in each layer are connected to neurons in the neighbouring layers. A layer in which all neurons are connected to all neurons in the neighbouring layers is known as a dense or fully-connected layer. The first layer is the input layer, the last layer is the output layer and the layers in between are the hidden layers. When the neural network receives input values, they become the activations of the input neurons. Because the size of the neural network is fixed, the input data must always have the same size corresponding to the size of the input layer.

The activations are passed on to the neurons in the next layer until it reaches the output layer. The activations in the output layer are the output values we are interested in. How the activations are passed on from neuron to neuron is determined by the weights and biases, which are discussed in Section 5.2.

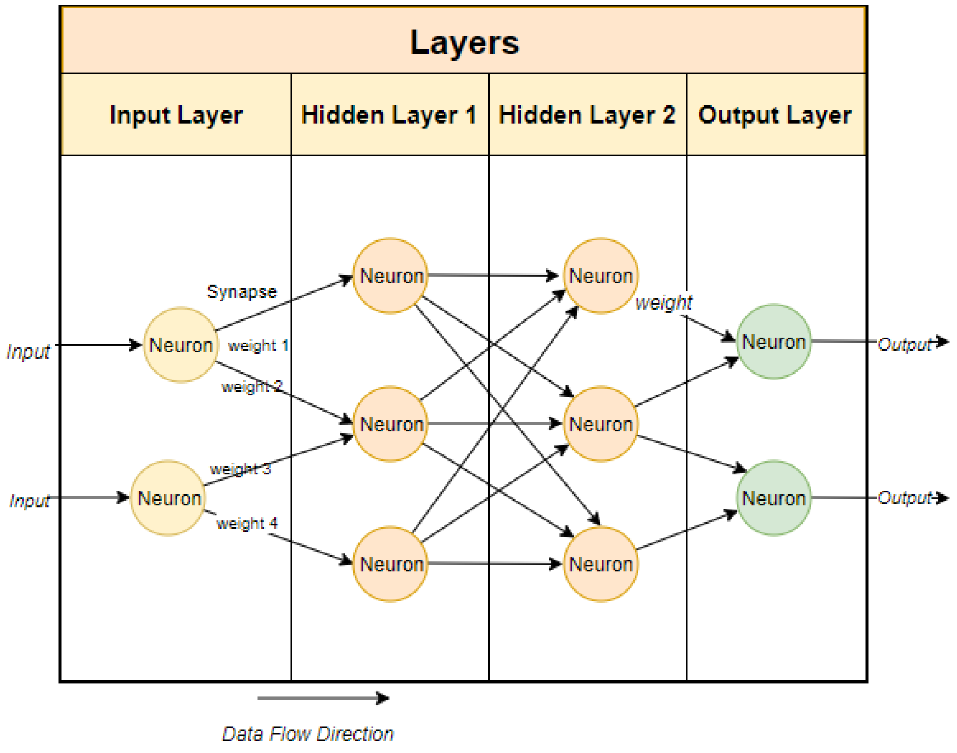


Figure 5.2: Basic structure of an artificial neural network. Neurons are positioned in layers and connected by synapses. Input data enters from the input layer and flows to the right. Depending on the weights of the synapses and the biases of the neurons, the activations are changed. The end result is the activations of the output neurons. Figure from Malik, F. [2019].

The number of neurons in each layer has an effect on how the neural network learns. If a hidden layer is smaller, information is lost as there are fewer neurons to describe the data. On the other hand, a larger layer can hold more information. Large layers could be seen as a system of equations with more equations than free variables. Thus they allow for more possible solutions, but introduce a risk of overfitting.

5.2 Model parameters

Figure 5.3 elaborates on how the activations are passed on from neuron to neuron. Each neuron in the hidden and output layers has a bias parameter b, which can be seen as the baseline activation that the neuron has. If the neuron would not receive any input, its output would be equal to b. Each synapse has a weight parameter w which signifies how well it connects two neurons. If the weight of the connecting synapse is large, the activation of a neuron will have a strong influence on the activation of a neuron in the next layer. Thus, these weights and biases determine how the input activations flow through the network, and ultimately what the output of the neural network will be. Together, the weights and biases are known as the model parameters. The correct values for the model parameters are determined during the training process. The output of each neuron is modified by the activation function, which is discussed in Section 5.3.

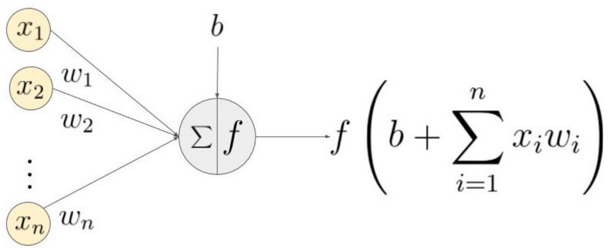


Figure 5.3: Schematic showing how neuron inputs are passed on to a neuron in the next layer, by Sharma, A. [2017].

From now on, the superscript will denote in which layer a parameter is located. The activations of the neurons in the first layer, $x^{(0)}$, are just our input values, so these are given. To find the activation of neuron k in the next layer, $x_k^{(1)}$, we use Equation (5.1). First, we take a sum of all input activations that the neuron receives from the previous layer, $\sum_{i=1}^{n} x_i^{(0)}$. This sum is weighted by the weights of each synapse which connects to the neuron, $w_{k,i}^{(1)}$. To this sum, we add the bias term $b_k^{(1)}$ of the neuron. The result of this is put inside an activation function $f()$.

$$x_k^{(1)} = f\left(b_k^{(1)} + \sum_{i=1}^{n} x_i^{(0)} w_{k,i}^{(1)}\right) \tag{5.1}$$

5.3 Activation function

The activation function modifies the linear output of a neuron, which is simply a sum of inputs, in a nonlinear way. This allows the neural network to approximate arbitrarily complex functions. For any layer in the neural network, it usually doesn't matter that much which activation function is used. However, some activation functions such as the Rectified Linear Unit (ReLU) are quicker to compute than others, and are therefore recommended for large or fully connected layers. In addition to that, one important constraint is that the range of the activation function of the output layer spans all possible output values. For example, if the output layer uses an activation function that is always positive, then the network cannot produce any negative output values.

Four common activation functions are shown in Figure 5.4. The simplest activation function, the Rectified Linear Unit (ReLU), is given in Equation (5.2). This function sets any negative input to zero, and does not modify positive inputs. Unfortunately, this could lead to what is known as the "dying ReLU problem", where many of the neurons produce zero outputs for any input. In such a scenario, the neural network stops learning. To address this, several alternative linear functions could be used, such as the Leaky ReLU in Equation (5.3).

$$\mathrm{ReLU}(x) = \max(0, x) \tag{5.2}$$
$$\mathrm{LReLU}(x) = \max(0.1x, x) \tag{5.3}$$

Two other commonly used activation functions are the sigmoid and hyperbolic tangent functions, Equation (5.4) and Equation (5.5), respectively. These are S-shaped asymptotic functions. The sigmoid has values that lie between 0 at $x = -\infty$ and 1 at $x = \infty$. The hyperbolic tangent function lies between -1 and 1 instead.

$$S(x) = \frac{e^x}{1 + e^x} \tag{5.4}$$
$$T(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{5.5}$$

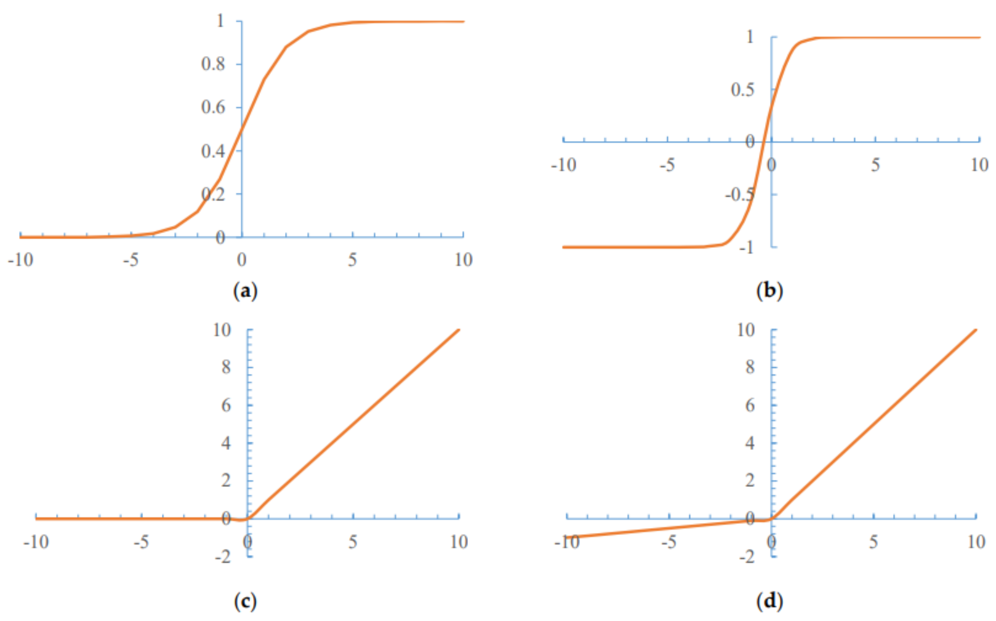


Figure 5.4: Four commonly used activation functions; sigmoid (a), hyperbolic tangent (b), ReLU (c) and leaky ReLU (d).

5.4 Convolutional neural networks

A layer that is used in convolutional neural networks (CNNs), and commonly in other neural networks as well, is the convolutional layer. This layer applies a number of kernels to its input, which can be seen as spatial features or patterns. Figure 5.5 by Pavlovsky [2019] illustrates how a kernel extracts features from an image. The activation of a kernel does not only depend on the values of the input neurons, but also on how these input neurons are related to each other. The kernels themselves are model parameters which are optimized during the training process.

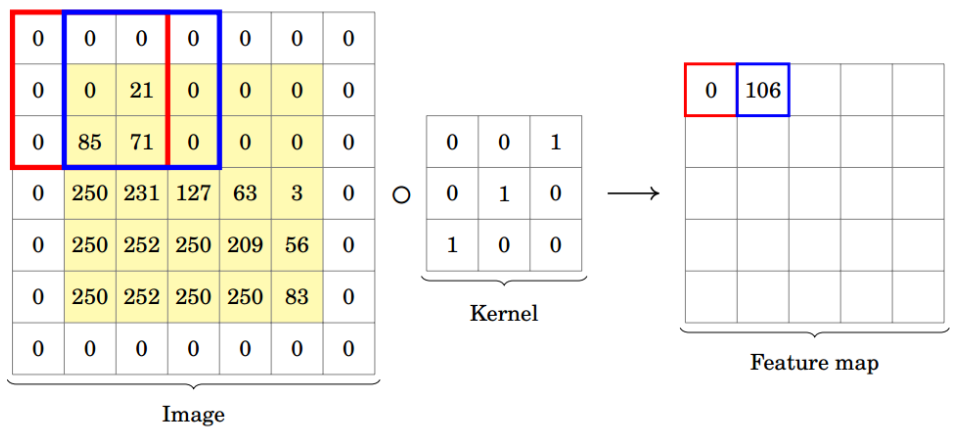


Figure 5.5: Illustration of the principle of a kernel, used in a convolutional layer [Pavlovsky, 2019]. At each pixel in the image, the surrounding pixels are multiplied by the kernel. The result is summed over in order to get a feature map. This particular kernel will detect mostly diagonal lines, but a convolutional layer consists of multiple kernels which each detect different features.

CNNs are useful for finding relations between a large number of neurons, and for features which are independent of position. For instance, a kernel might be good at recognizing faces in an image. The position of the face does not matter as the kernel applies to the entire image. In our case, we use a CNN with 1-dimensional convolutional layers in order to recognize features in a spectrum.

5.5 Autoencoders

The CNN we use has the structure of an autoencoder. A simple illustration of such a neural network is shown in Figure 5.6. In general, the goal of an autoencoder is to reproduce the input, except for any unimportant details. Hence, the size of the input and output layers should be the same. Another defining feature of an autoencoder is that the hidden layer(s) are smaller than the input and output layers. This means that some information of the input data is lost, as there are fewer neurons to store the data.

An autoencoder could therefore be separated into two distinct parts; an encoder and decoder. The first layers of the neural network should find a way to encode the input as efficiently as possible, making sure that the most important information is retained. The last layers should decode this compressed information in such a way that it replicates the input data. The smallest layer, the bottleneck, determines the amount of information that is lost, and thus the amount of data compression.

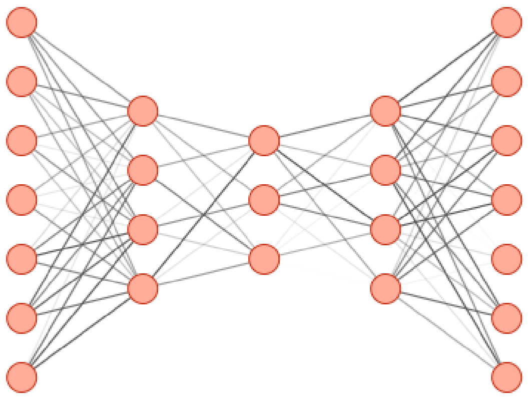


Figure 5.6: Basic structure of a (fully connected) autoencoder type of neural network. Figure made using the tool by LeNail [2019]. The left part of the network is known as the encoder as it reduces the number of neurons used to represent the data, while the right part is known as the decoder.

It may seem like a useless task to train a neural network to replicate its input data, but it has several use cases. Most notably, autoencoders are used for noise reduction. An example of this use case is shown in Figure 5.7. Because the encoding part of the autoencoder is trained to retain only the most important features in the data, it will mostly ignore the noise. Another useful property of an autoencoder is that the decoder part of the autoencoder could be separately used to generate data similar to the training data. This is often employed in image generation, for instance.

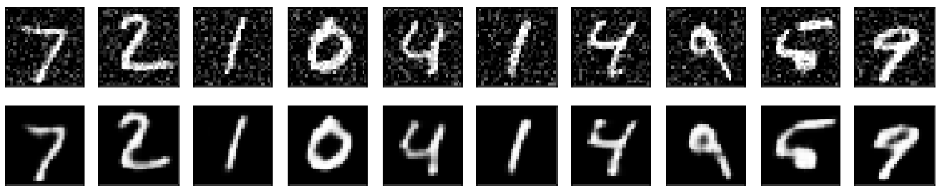


Figure 5.7: Autoencoders can be used for noise reduction. In this case, an autoencoder was trained to reproduce images of handwritten numbers from the MNIST dataset. After a sufficient amount of training, the network is able to reproduce the numbers. If the network is then fed noisy images (shown in the top row), it will output relatively noiseless images (bottom row).

Chapter 6

Training a neural network

To obtain a correctly working neural network, we need to tweak many parameters. There are two types of parameters found in a neural network. The parameters that change the general behaviour are called hyperparameters and are discussed in Section 6.6. The other parameters, such as weights and biases, are called model parameters and are discussed in Section 5.2. The hyperparameters are chosen before a neural network is trained, while the model parameters are found during the training process. This chapter treats the general theoretical concepts related to the data preparation and the training process. The specific settings used to train our algorithm are discussed in Chapter 7. For a general introduction to the training process of neural networks, see also Nielsen [2019].

At the first step of the training process, we use random values for the model parameters. Needless to say that the output will also be very bad. To get a reasonable result, the model parameters need to be improved step by step. The training is done by feeding input data into the neural network and comparing its output with the desired output. The cost function, which is discussed in Section 6.3, is a measure for the similarity between the two. The model parameters of the neural network are then readjusted according to this cost, using the backpropagation mechanism as discussed in Section 6.7.

6.1 Rescaling

To speed up the training process of the neural network it is necessary to preprocess the data in some way. The scaling of the data is closely related to how fast a neural network learns. During the backpropagation step, the derivative of the activation function is used to determine the amount by which the model parameters should be adjusted. Therefore, it is important to have large derivatives for the activation function. Depending on the activation function that is used, the derivatives are typically largest between -1 and 1 or -1 and 1. For this reason, the data should always be rescaled before the training phase to ensure that most of the values are within this range.

6.2 Data sets

Before training, the available data needs to be separated into two to three data sets; a training set, a test set and optionally a validation set. Each set contains both the input data and the desired output, which are the same in the case of an autoencoder.

In the training phase, the neural network is given the training data, which it uses to tweak the model parameters and improve the results. The training input data is fed into the neural network, and the output of the network is compared to the desired output.

Each time the model parameters are readjusted, we use the test set to evaluate how well the neural network performs. The test set is completely independent from the training process, unlike the training set. If the neural network provides good results on the training set but poor results on the test set, this indicates that the neural network is overfitted to the training data. Thus, instead of *learning*, the neural network has simply *memorized* the desired outputs of the training data. This is not what we want, because we want the neural network to perform well on new, unknown data. For this reason, the network should stop training before the test results deteriorate significantly with respect to the training results.

Lastly, we could use a validation set to find the best hyperparameters. The neural network can be trained multiple times with different hyperparameters, and the validation set can then be used to find out which hyperparameters give the best results. The validation set is therefore not completely independent from the training process. To once again ensure that the neural network preforms well on new data, we would use the test set. If the network performs well on the validation set but not so well on the test set, then the network's hyperparameters may just coincidentally be the best choice for the validation set only, and not for new data in general.

6.3 Cost

In order to improve the neural network by changing the model parameters, we need to evaluate how well the neural network performs. This is quantified by the cost. As we are using an autoencoder, we want the output of the neural network to match its input as closely as possible. The difference between the prediction of the neural network, $\hat{Y}$, and the desired output Y, is put into a cost function. One possible cost function is the mean squared error, as given in equation Equation (6.1).

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{6.1}$$

We generally want to minimize the result of the cost function, as this indicates that the difference between the validation set Y and the prediction of the neural network $\hat{Y}$ is smallest. The prediction values $\hat{Y}$ are calculated by the neural network and thus depend on its model parameters, $\vec{\mathbf{W}}$. Therefore, the cost function can also be seen as a high-dimensional function $C(\vec{\mathbf{W}})$, with one dimension for each model parameter. The location of the global minimum of the cost function is given by the most optimal set of model parameters.

6.4 Gradient descent

In order to find an optimal set of model parameters, we want to find a minimum in the cost function C. Due to its high dimensionality, sampling the cost function and finding the global minimum is an almost impossible task; it would take far too long. Instead, we can find a reasonable local minimum using the stochastic gradient descent technique.

Gradient descent is an iterative technique to find a local minimum of a function. An illustration is shown in Figure 6.1. The negative gradient $-\boldsymbol{\nabla} f$ of a function f gives the direction in which the function decreases fastest. A gradient is a vector, where each component gives us the maximum slope in a particular dimension. To find the minimum of the function f, the gradient of the function is first evaluated at some starting point x_0. The next point x_1 will then be chosen to be in the direction of the negative gradient. The gradient is evaluated again at the new point, and the process is repeated. When the gradient becomes small, it is likely that a local minimum is reached.

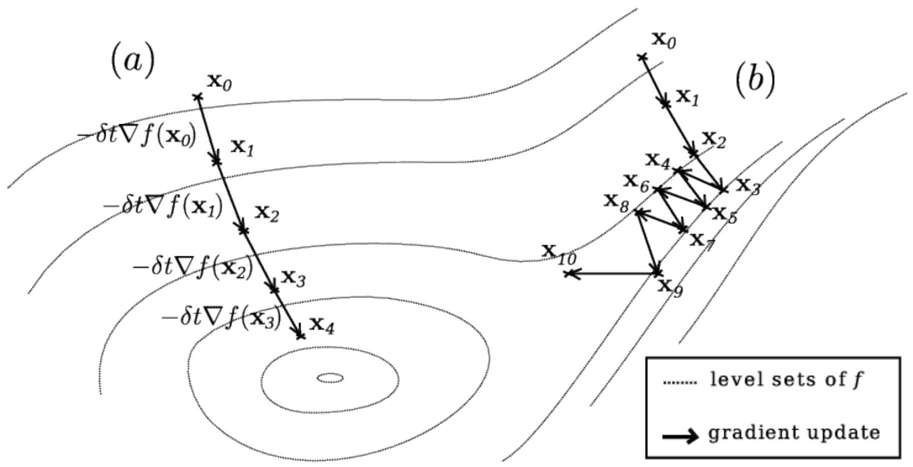

(a)
$\mathbf{x}_0$
$-\delta t\nabla f(\mathbf{x}_0)$
$\mathbf{x}_1$
$-\delta t\nabla f(\mathbf{x}_1)$
$\mathbf{x}_2$
$-\delta t\nabla f(\mathbf{x}_2)$
$\mathbf{x}_3$
$-\delta t\nabla f(\mathbf{x}_3)$
$\mathbf{x}_4$
(b)
$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$ $\mathbf{x}_6$ $\mathbf{x}_7$ $\mathbf{x}_8$ $\mathbf{x}_9$ $\mathbf{x}_{10}$
level sets of f
gradient update

Figure 6.1: Two trajectories that use gradient descent on a function f. In case (a), the trajectory goes straight to the minimum, while in case (b), the trajectory overshoots the valley and bounces back and forth. It is also possible to completely miss a local minimum if it lies in between two steps.

If the step size is too low, it will take many steps to get to a local minimum. However, if the step size is too large, it is likely that a local minimum is overshot and missed. For a neural network, the step size of the gradient descent directly depends on the learning rate. To increase the training efficiency and to avoid overshooting any minima, it is possible to decrease the learning rate as the cost function gets smaller.

The negative gradient of the cost function, $-\boldsymbol{\nabla} C(\vec{\mathbf{W}})$, gives us the direction in our model parameter space in which the cost function decreases fastest. Changing our model parameters towards this direction decreases the cost of the network most efficiently. Thus, we adjust the model parameters $\vec{\mathbf{W}}$ proportionally to $-\boldsymbol{\nabla} C(\vec{\mathbf{W}})$.

6.5 Epochs and batches

To speed up the training process and to get a better estimate for the true cost, the training data is fed into the neural network in batches. Each batch consists of a selection of training samples. After each batch, we evaluate the performance of the network using the cost function. Then we change the model parameters accordingly using backpropagation.

The process is repeated many times. Whenever the batches have gone through all the training samples, the same training data is reused, but shuffled to form new batches. Every time that all of the training data has been used, is classified as an epoch. Although each epoch uses the same training samples, these samples are shuffled so that we can make new batches. Usually it takes many epochs to fully train a neural network. The performance of the network is assessed after each epoch using the cost of the test set. This cost is not used in the training process, but it allows us to track the performance of the network as it is learning.

6.6 Hyperparameter optimization

Hyperparameters are parameters that determine the general behaviour of the neural network. For example, the sizes, types and number of layers affect the results of the network. Also the activation functions used in each layer are hyperparameters. Most hyperparameters are discrete, so we cannot use gradient descent to optimize them. Instead, the most used technique is to sample these parameters randomly within a given range. For instance, we might choose the size of a layer to be between 10 and 20 neurons, For each set of random hyperparameters, the neural network is trained. Using the lowest cost in the test data obtained by each network, we can determine the best set of hyperparameters.

6.7 Backpropagation

As discussed in Section 6.3, the cost function C is a measure for how well the network performs, and is a function of each of the model parameters in the network. To improve the model parameters and ultimately improve the results, the cost has to be minimized. This we can do by applying gradient descent. The process of adjusting the model parameters using the gradient of the cost function is known as backpropagation, and it describes the underlying mechanism of the training process.

The term backpropagation derives from the fact that the calculations are propagating backwards through the neural network, from the output layer to the input layer. First, the model parameters in the output layer are adjusted, then the model parameters in the previous layer and so on. In this section, a derivation is given for Equations (6.12) and (6.13), which are the most important equations used in backpropagation.

The gradient of the cost function $\boldsymbol{\nabla} C$ can give us the best way to adjust the model parameters in order to minimize the cost. The cost function directly depends on the model parameters $\vec{\mathbf{W}}$, which includes each of the weights w and biases b. Hence, its gradient is given by Equation (6.2). To simplify, we only calculate the cost of a single training sample i, and it is assumed that the neural network has L layers of one neuron each. Regardless of these simplifications, the derivation follows the same principles for more complex cases.

$$\boldsymbol{\nabla} C_i(\vec{\mathbf{W}}) = \begin{pmatrix} \frac{\partial C_i}{\partial b^{(1)}} \\ \frac{\partial C_i}{\partial w^{(1)}} \\ \vdots \\ \frac{\partial C_i}{\partial b^{(L)}} \\ \frac{\partial C_i}{\partial w^{(L)}} \end{pmatrix} \tag{6.2}$$

To calculate these partial derivatives, let's first only consider the last layer L. The activations of the last layer are given by $x^{(L)}$, and depend on the activations of the previous layer $x^{(L-1)}$ as given by Equations (6.3) and (6.4). The desired output values are y. For simplicity, we defined z to be the input in the activation function f.

$$x^{(L)} = f(z^{(L)}) \tag{6.3}$$

$$z^{(L)} = w^{(L)} x^{(L-1)} + b^{(L)} \tag{6.4}$$

The partial derivatives with respect to the weights w and biases b can now be expanded using the chain rule.

$$\frac{\partial C_i}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial x^{(L)}}{\partial z^{(L)}} \frac{\partial C_i}{\partial x^{(L)}} \tag{6.5}$$

$$\frac{\partial C_i}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial x^{(L)}}{\partial z^{(L)}} \frac{\partial C_i}{\partial x^{(L)}} \tag{6.6}$$

Three of these terms can be easily derived from Equations (6.3) and (6.4).

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = x^{(L-1)} \tag{6.7}$$

$$\frac{\partial z^{(L)}}{\partial b^{(L)}} = 1 \tag{6.8}$$

$$\frac{\partial x^{(L)}}{\partial z^{(L)}} = f'(z^{(L)}) \tag{6.9}$$

Here, f' is the derivative of the activation function of the layer, and should be a relatively well-known function. The last term $\frac{\partial C_i}{\partial x^{(L)}}$ depends on the cost function. Taking the often used mean squared error as the cost function, the cost for a single training sample can be written as

$$C_i = \left(x^{(L)} - y\right)^2, \tag{6.10}$$

where $x^{(L)}$ are the activations of the output neurons and y are the desired output values. The partial derivative of the cost function with respect to the activations $x^{(L)}$ is then given by:

$$\frac{\partial C_i}{\partial x^{(L)}} = 2\left(x^{(L)} - y\right) \tag{6.11}$$

Putting all of the partial derivatives from Equations (6.7) to (6.9) and (6.11) together, we can expand the partial derivatives in Equations (6.5) and (6.6) as:

$$\frac{\partial C_i}{\partial w^{(L)}} = x^{(L-1)} \cdot f'(z^{(L)}) \cdot 2\left(x^{(L)} - y\right) \tag{6.12}$$

$$\frac{\partial C_i}{\partial b^{(L)}} = 1 \cdot f'(z^{(L)}) \cdot 2\left(x^{(L)} - y\right) \tag{6.13}$$

Equations (6.12) and (6.13) allow us to determine by how much we should adjust the model parameters of layer L, given the activations of the L and $L-1$ layers, and the cost. In order to do the same for the previous layer $L-1$, all we would need to find is the partial derivative with respect to $x^{(L-1)}$ instead of $x^{(L)}$. We can again expand this using the chain rule:

$$\frac{\partial C_i}{\partial x^{(L-1)}} = \frac{\partial z^{(L)}}{\partial x^{(L-1)}} \frac{\partial x^{(L)}}{\partial z^{(L)}} \frac{\partial C_i}{\partial x^{(L)}} \tag{6.14}$$

$$= w^{(L)} \cdot f'(z^{(L)}) \cdot 2\Big(x^{(L)} - y\Big) \tag{6.15}$$

Now we can find the partial derivatives with respect to $w^{(L-1)}$ and $b^{(L-1)}$, and repeat the process, propagating backwards until the first layer. Although this derivation was simplified by considering only single neuron layers, the procedure is largely the same when using multiple neurons. The only difference is that the parameters become vectors and matrices, and that we should take a sum when calculating the cost function and its derivative with respect to the activation.

Chapter 7

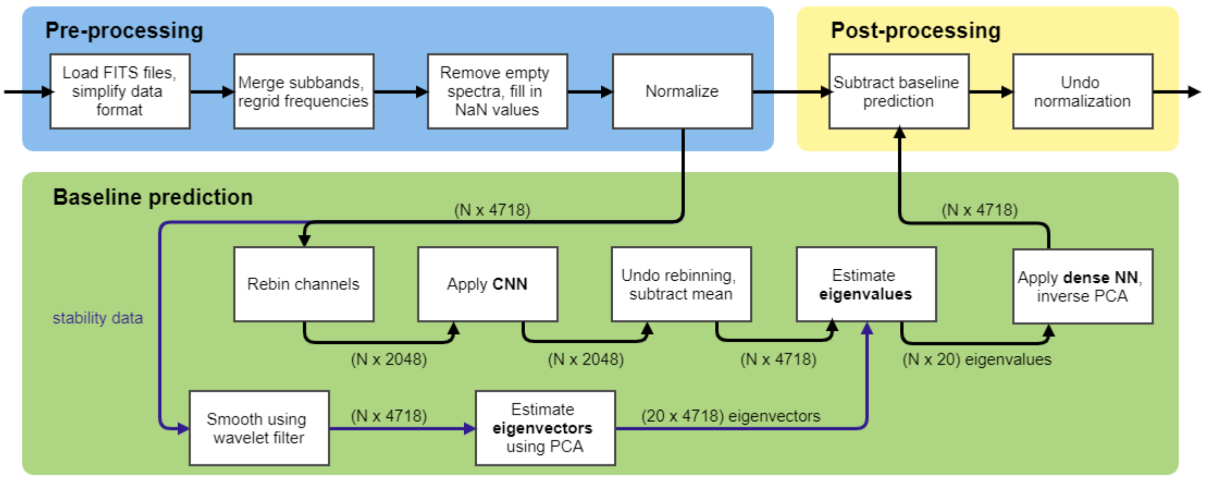
Methods

Spectral instruments using hot electron bolometer (HEB) mixers tend to have issues with electrical standing waves (ESWs). As discussed in Chapter 4, these ESWs show up as strong, broad features over the entire spectrum. To get rid of ESWs, we use a combination of several methods. The goal of each method is to extract only the features of the spectrum that are due to ESWs. The end result is the baseline component of the spectrum. This baseline component can then be subtracted from the original spectrum to give a spectrum devoid of any ESWs. To train the algorithm, we use stability data (see also Section 4.1)

7.1 Overview

The fundamental design of our algorithm is to use a linear combination of orthonormal components in order to describe a baseline. For HIFI, the baselines can be observed in stability spectra, discussed in Section 4.1. We fit a principal component analysis (PCA) to this stability data in order to get a set of orthonormal eigenvectors. Once these eigenvectors have been found, we can fit the baseline of any observed spectrum as a linear combination of these eigenvectors.

A flowchart of the entire algorithm can be seen in Figure 7.1. The first part of the algorithm deals with pre-processing the data, which is discussed in Section 7.2. This includes normalization and discarding faulty data. After preprocessing, we smooth the stability data using a wavelet filter, as most of the high-frequency waves in the spectrum are noise. Then we apply a PCA to find the set of eigenvectors.



Pre-processing
Load FITS files, simplify data format
Merge subbands, regrid frequencies
Remove empty spectra, fill in NaN values
Normalize
Post-processing
Subtract baseline prediction
Undo normalization
Baseline prediction
(N x 4718)
(N x 4718)
stability data
Rebin channels
Apply CNN
Undo rebinning, subtract mean
Estimate eigenvalues
Apply dense NN, inverse PCA
(N x 2048)
(N x 2048)
(N x 4718)
(N x 20) eigenvalues
Smooth using wavelet filter
(N x 4718)
Estimate eigenvectors using PCA
(20 x 4718) eigenvectors

Figure 7.1: Flowchart of the algorithm. As input, it takes level 1 spectral observations in FITS file format. The algorithm predicts the baseline, which is subtracted from the original spectrum at the end. This results in a spectrum with a flat baseline. At different steps of the algorithm, the typical shape of the data is given. In this case, there are N spectra with 4718 channels each. Before the CNN, they are rebinned to 2048 values. Assuming that 20 principal components are used, we end up with 20 eigenvectors with 4718 values each, and $N \cdot 20$ eigenvalues. These are then converted back to N spectra with 4718 channels each.

Once these eigenvectors have been found, we rebin the data to reduce its size, and we use a convolutional neural network (CNN). The CNN recognizes the features that describe the baseline, and ignores features that are not found in the stability data. The result is a smooth spectrum that does not contain any features from astronomical signals. In the next step, we estimate the eigenvalues using a PCA transformation with the previously found eigenvectors. This transformation greatly reduces the dimensionality of the data. Usually, spectra consist of a few thousand channels, while the number of required principal components is of the order of dozens. Unfortunately, strong local features affect the outcome of the PCA transformation over the entire spectrum. In the last step, we use a dense neural network to correct this. In the dense neural network, we also perform a reverse PCA transformation to convert the spectrum back to frequency space. This gives a prediction for the ESW component in the spectrum. If we subtract this baseline prediction from the original spectrum, we obtain a spectrum with a flat baseline.

Before the neural networks can be used, they need to be trained on stability data. Figure 7.2 gives an overview of the training process and how the stability data is used. The stability spectra do not contain strong astronomical signals, although they do contain noise. We smooth the stability spectra using a wavelet filter from the PyWT package [Lee et al., 2019], after which each spectrum should give a good approximation of a real baseline. We use these smoothed stability spectra as our desired output data. As explained in Section 7.2.2, the input data contains additional mock signals and noise to approximate real spectra with emission lines. This way the algorithm learns to ignore signals in the input spectra, and give a prediction of the baseline. Moreover, this allows us to drastically increase the size of the data set; we can reuse the same baseline while adding different mock signals and noise to get multiple input spectra. At each step during the training process, the baseline prediction is compared with the corresponding smoothed stability spectrum. This comparison is used to improve the model parameters of the neural networks.

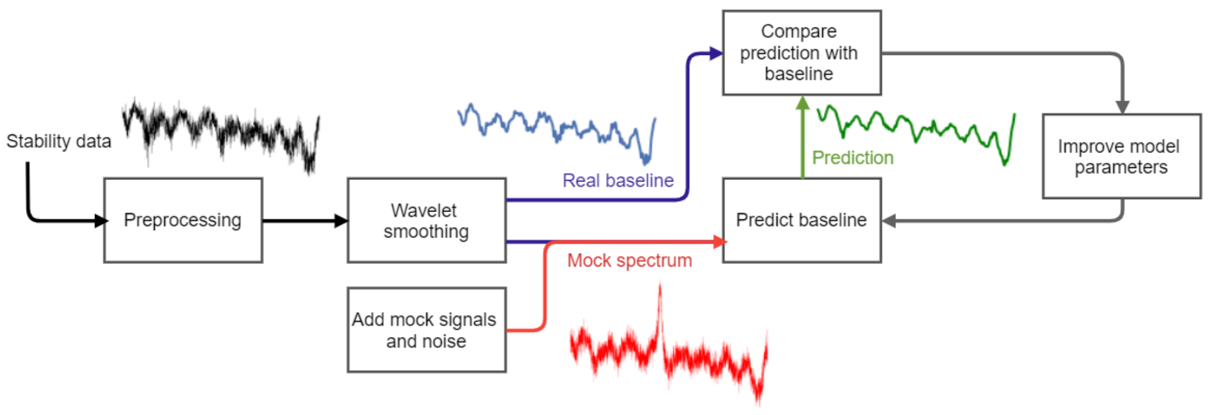


Figure 7.2: Flowchart demonstrating how stability data and mock signals are used to train the neural networks.

7.2 Data description and preprocessing

Before the neural networks can be used, they first need to be trained on data. We want the algorithm to deduce the standing wave, or baseline component from a spectrum. Thus the desired output data is a smooth baseline. For this we use smoothed HIFI stability data. This data is obtained using the Herschel Science Archive and processed using HIPE [Ott et al., 2010] to level 1 of the pipeline, excluding the ESW correction of course. The Jython script used to process the data in HIPE is given in Chapter 12 of the appendix. Table 7.1 shows a list of data that was used while developing the algorithm.

As described in Section 4.1, stability spectra do not contain any notable component other than ESWs, which is why we use this data to "model" the ESWs. By smoothing the data with a low frequency bandpass wavelet filter, we get rid of the high frequency noise which is not caused by ESWs but mostly by other instrument noise. We use the same spectra to generate the input data. This process is described in Section 7.2.2. We end up with 3072 spectra in the stability data, although this could vary slightly depending on the preprocessing. As we reuse the same stability spectrum to generate multiple input spectra, we would typically end up with around 100.000 input spectra to be used for the training process.

To assess the performance of our algorithm, we apply it to a selection of [CII] observations from the GOT C+ survey [Langer et al., 2010]. This survey probed the Milky Way disk with deep spectral integrations in band 7b of HIFI, albeit using a sparse spatial sampling due to Herschel's small field of view. In total, over 900 lines of sight were observed, from which we picked a sample of 8 observations. These observations had a variety of quality, as indicated by their pipeline quality control reports. For instance, the first five observations were performed at warm beta angles, where Herschel observed in a direction close to the Sun. The last two observations had good ESW suppression with the pipeline method, while the others could be improved.

Table 7.1: List of data that was used for training and testing purposes. The stability data is divided into a training and testing set, and is used to represent ESWs. The [CII] data, which is a small sample from the GOT [CII] data set [Langer et al., 2010], is used to compare the results from our algorithm with those of the pipeline. All data was observed with HIFI's Wide Band Spectrometer (WBS).

data type	obsid	RA (deg)	DEC (deg)	band	date	duration (s)
stability	1342192559	300.00	-42.00	6b	2010-03-21	6905
	1342194762	312.00	-5.00	7b	2010-04-16	6967
	1342196580	320.00	40.00	7b	2010-05-15	6938
	1342214400	47.00	-10.00	6a	2011-02-17	7232
	1342214451	310.00	-82.00	6b	2011-02-18	7258
	1342233277	290.00	41.00	7a	2011-11-28	6902
	1342235789	337.30	6.70	7b	2011-12-30	6902
	1342244375	318.42	-1.45	7a	2012-04-13	6902
CII	1342194765	276.51	-13.26	7b	2010-04-17	1055
	1342194766	276.05	-13.03	7b	2010-04-17	784
	1342194770	278.52	-8.41	7b	2010-04-17	784
	1342200733	189.23	-62.83	7b	2010-07-06	1494
	1342201649	200.39	-62.67	7b	2010-07-29	1494
	1342208547	277.70	-9.95	7b	2010-10-29	2535
	1342210310	293.61	20.00	7b	2010-11-27	2535
	1342210319	295.98	23.89	7b	2010-11-27	784

7.2.1 Preprocessing

The data is first processed up to level 1 of the HIFI pipeline using the HIPE software [Ott et al., 2010]. The Jython script that was used to process the data in HIPE is given in Chapter 12 of the appendix. At level 1 of the HIFI data, each spectrum is still split in multiple overlapping subbands. To make the observations suitable for the neural networks, we merge the subbands together into one spectrum. We do this by defining the frequencies at the middle of the overlapping frequencies as the cutoff points. Left of each cutoff point, the first subband is used, and right of the cutoff point, the second subband is used.

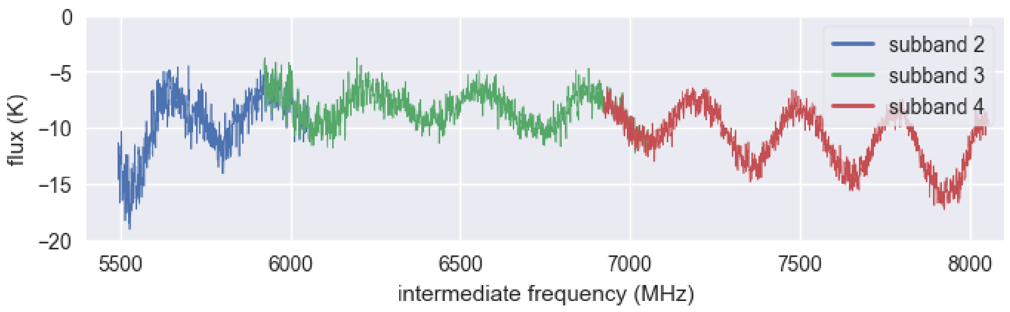


Figure 7.3: Each spectrum in the 7b band consists of separate measurements from three overlapping subbands.

Although we apply the `doFreqGrid` function from the HIFI pipeline to make the frequencies evenly spaced, this does not ensure that the frequency ranges are the same for each spectrum. It is important that the frequencies of each spectrum overlap, as the ESWs depend on the intermediate frequencies. To do this, we use 1-dimensional interpolation. The edges of the spectra are clipped slightly to ensure that each spectrum has complete data over the entire frequency range. It is therefore necessary that the frequency range of the observational data matches that of the data used to train the algorithm.

After the subbands are merged and the frequencies have been regridded, we remove spectra that are completely empty. In some spectra we interpolate single faulty data values that are marked as NaN. We then normalize the data using min-max scaling with zero mean, using Equation (7.1). As we apply a PCA transformation later, it is important that the mean of the data is zero. Furthermore, as explained in Section 6.1, scaling the data has an influence on how fast a neural network learns. It is critical that the activation functions used by the neural networks span the range of the scaled data. For instance, a layer that uses a sigmoid activation function can only produce values between 0 and 1, and should not be applied if the data is scaled between -1 and 1.

$$\begin{aligned} X &= X - \mathrm{mean}(X) \\ X &= X \ / \ \max(\mathrm{abs}(X)) \end{aligned} \tag{7.1}$$

Lastly, the spectra are smoothed using a wavelet filter from the PyWT package [Lee et al., 2019] to get rid of the noise and obtain clean baselines. These baselines are used for the desired output data and to get the PCA transformation. We end up with 2151 baselines from the stability data, although this may vary depending on the preprocessing steps applied,

If the data is used for training, we split the data in a training set and a test set. The test set is used to measure the performance of the algorithm objectively and is not used to improve the algorithm. We use 80% of the stability data for training purposes and 20% for testing. Commonly, about 70 to 80% of the data is used for training, but this ratio depends on the data set. As we have a somewhat small data set, we opt to use a larger percentage for training purposes. Using a small percentage for training would give worse results, whereas using a small percentage for testing would give a larger uncertainty on the accuracy of the results.

7.2.2 Generating mock spectra

The input data we use to train our algorithm consists of the smoothed stability spectra with added noise and mock signals. This way, we try to replicate an astronomical observation with emission lines. An example of such a simulated spectrum can be seen in Figure 7.4. As our desired output data, we use the smoothed stability spectra to represent the baselines in the spectra.

Using simulated spectra has the benefit that we can generate more data; we can use each one of the 3072 baselines from the stability data to generate multiple input spectra by adding different mock signals. This way, we end up with around 100.000 mock spectra. Furthermore, using mock signals has the advantage that we know exactly what the signal and baseline components are in the spectrum, which would not be possible with real observations. We need to know the true baseline component in each spectrum in order to train the algorithm.

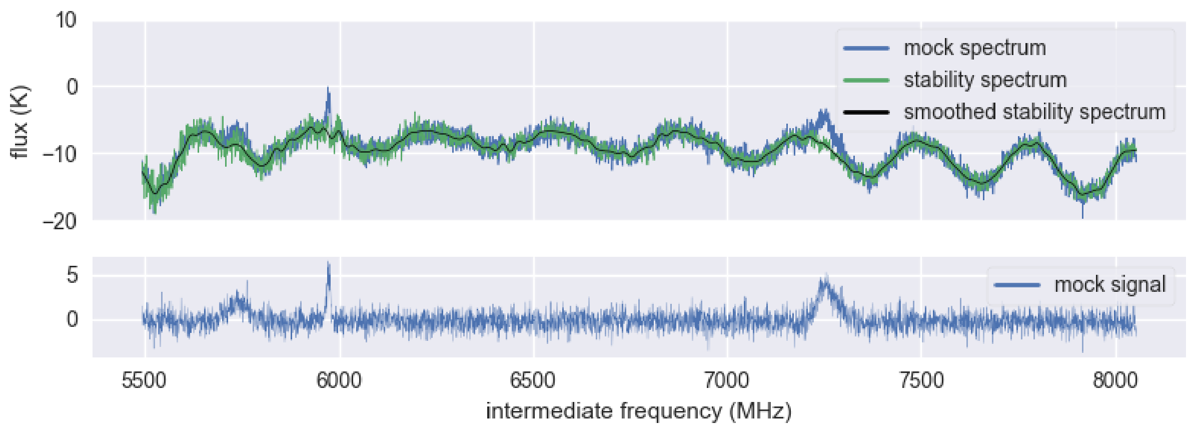


Figure 7.4: Example of a mock spectrum (top, blue), made by adding mock signals (bottom) to a smoothed stability spectrum (top, black).

The mock signals in the mock spectrum are generated as skewed Gaussians. The signals are allowed to overlap, which leads to compound signals with more complex shapes. Each signal has a random width, amplitude, skewness and position, and each spectrum has a random number of signals up to a maximum. Because the mock signals are present in the input data but not in the desired output data, the algorithm learns to ignore them. The parameter space for the mock signals was chosen to be quite large, so that the algorithm can deal with signals of any shape.

In principle the parameter space of the mock signals could be narrowed down so that the mock signals would better reflect the expected signals in a specific observation. For example, the location of the mock signal could be randomized to be close to the location of a known emission line. Potentially, this may make the algorithm more accurate for that specific observation, but it could also become more prone to ignoring unknown signals.

7.3 Convolutional neural network

The next step of the algorithm is a convolutional neural network (CNN). The CNN has an autoencoder layout (see also Section 5.5). This way, it learns to encode and decode only the features that are important for describing the baseline, and gets rid of some of the signal features in the spectrum that could not be smoothed out. To train the CNN to do this, we use the mock spectra as input, and the smoothed stability spectra as our desired output.

The CNN consists of many convolutional layers which, as described in Section 5.4, recognize patterns in multiple input neurons. Each kernel in a convolutional layer finds one pattern, so the result of the convolutional layer depends on the number and sizes of these kernels. The kernel size corresponds to the width of each features. In order for the algorithm to work properly, the kernel size should be smaller than the width of a single ESW ripple, but larger than a typical signal width. For this reason, the CNN will fail to remove features that are broader than an ESW ripple, for which we need to use PCA.

The individual layers used in the CNN are given in Table 7.2. Regardless of the number of channels in the data, the data is first rebinned to match the input shape of the `Input` layer. In order for the layers to work properly, the input shape should be a very divisible number such as a power of 2. The `Zero Padding` layer adds padding for when this is not the case. The next layers consist of `Convolutional`, `Max Pooling` and `Upsampling` layers. The `Max Pooling` layers take the maximum activation value of each two neurons, resulting in an output shape half as large as the input shape. Conversely, each `Upsampling` layer copies each value twice, doubling the size of the data.

Table 7.2: Layout used for the CNN. The first two columns give the type of layer and the index. The second column specifies shape of the output data. Here, the first dimension has no fixed size, as it is equal to the number of data samples. The last dimension specifies the number kernels used by the convolutional layers (see also Section 5.4)). For convolutional layers, the kernel size is also given in the next column. The last column gives the number of model parameters in each layer. The network was developed using Keras [Chollet et al., 2015] with the TensorFlow backend [Abadi et al., 2016].

#	Layer	Output Shape	Kernel size	# of Params
1	Input Layer	(None, 2048, 1)	-	0
2	Zero Padding	(None, 2048, 1)	-	0
3	Convolutional	(None, 2048, 64)	80	9408
4	Max Pooling	(None, 1024, 64)	-	0
5	Convolutional	(None, 1024, 32)	25	51232
6	Max Pooling	(None, 512, 32)	-	0
7	Convolutional	(None, 512, 32)	5	5152
8	Max Pooling	(None, 256, 32)	-	0
9	Convolutional	(None, 256, 32)	3	3104
10	Max Pooling	(None, 128, 32)	-	0
11	Convolutional	(None, 128, 32)	3	3104
12	Max Pooling	(None, 64, 32)	-	0
13	Convolutional	(None, 64, 32)	3	3104
14	Upsampling	(None, 64, 32)	-	0
15	Convolutional	(None, 128, 32)	3	3104
16	Upsampling	(None, 256, 32)	-	0
17	Convolutional	(None, 256, 32)	3	3104
18	Upsampling	(None, 512, 32)	-	0
19	Convolutional	(None, 512, 32)	5	5152
20	Upsampling	(None, 1024, 32)	-	0
21	Convolutional	(None, 1024, 64)	25	51264
22	Upsampling	(None, 2048, 64)	-	0
23	Convolutional	(None, 2048, 1)	80	9345
24	Cropping	(None, 2048, 1)	-	0

We use the mean squared error between the mock spectra and the smoothed stability spectra as the loss function, to quantify the performance of the CNN during the training process. To speed up the training process, the spectra are rebinned first to a lower resolution in frequency. The resolution should be high enough to distinguish any signals in the data, because at low resolutions the signals blend with the baseline. To further improve the efficiency of the training, we also reduce the learning rate as the results improve. Figure 7.5 shows the learning rate and loss for one training. The learning rate specifies the speed at which the gradient descent is done, and thus the speed at which the CNN learns. At the start of the training, the learning rate should be relatively high, so that the solution space is explored. As the loss approaches a local minimum, the learning rate is reduced so as to not overshoot the local minimum. Due to the stochastic nature of the training, it is possible that the model worsens as time progresses. This can be seen in Figure 7.5 at epoch 5. Therefore, to find the best model parameters, we use the model parameters that give the smallest loss for the test set, not the model parameters in the final epoch.

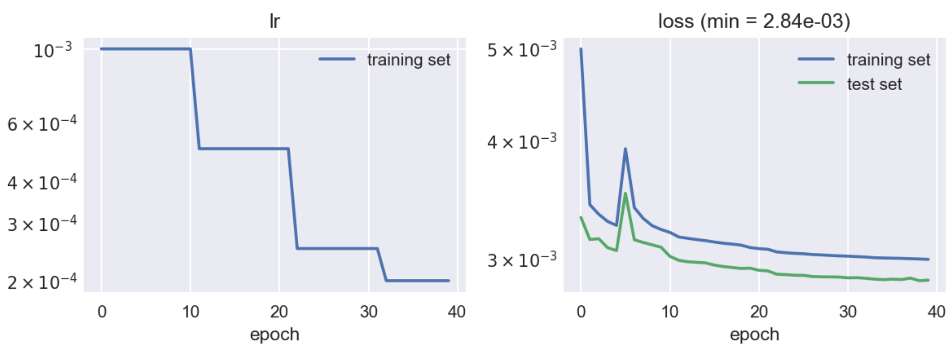


Figure 7.5: Learning rate and loss for each epoch, during the training of a CNN. The learning rate is reduced as the loss gets smaller.

7.4 Principal component analysis

After the CNN removed most of the prominent features, we apply a principal component analysis (PCA) to obtain a simpler representation for the data. PCA is a method that performs a change of basis, as shown in Figure 7.6. The orthogonal unit vectors that describe this new coordinate system are called principal components, or eigenvectors, while their respective magnitudes are known as eigenvalues. In our case, the eigenvectors are the same for each spectrum, but the eigenvalues are unique and can be used as a more compact representation of a spectrum.

The change of basis is done in such a way that the data has the most variance in the direction of the first eigenvectors, and less variance in each successive eigenvector. Theoretically, an N-dimensional PCA transformation would result in N number of eigenvectors. However, only the first few eigenvectors are needed to explain the majority of the data as they contain the most variance. Hence, we can reduce the dimensionality considerably by discarding the other eigenvectors. Instead of using the approximately 5000 channels for each spectrum, we can describe each spectrum with a dozen or so eigenvalues. The number of components that are used in the transformation is quite important, and is discussed in Section 7.4.1.

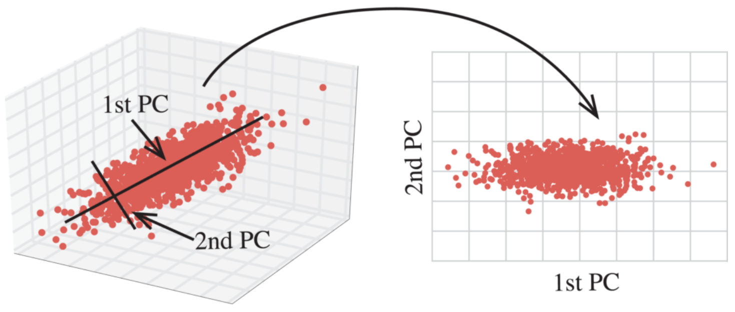


Figure 7.6: Example of a PCA transformation in two dimensions. This shows how three-dimensional data can be effectively described in two dimensions without much loss in information. Figure from Vadali [2018].

In our algorithm, the PCA transformation is ”fitted” to the smoothed baseline from the stability data. This results in a set of eigenvectors that most accurately describe the ESWs. The first few eigenvectors of such a PCA transformation are shown in Figure 7.7. After the PCA transformation, each spectra is uniquely described by a set of eigenvalues. These eigenvalues are used by the next neural network.

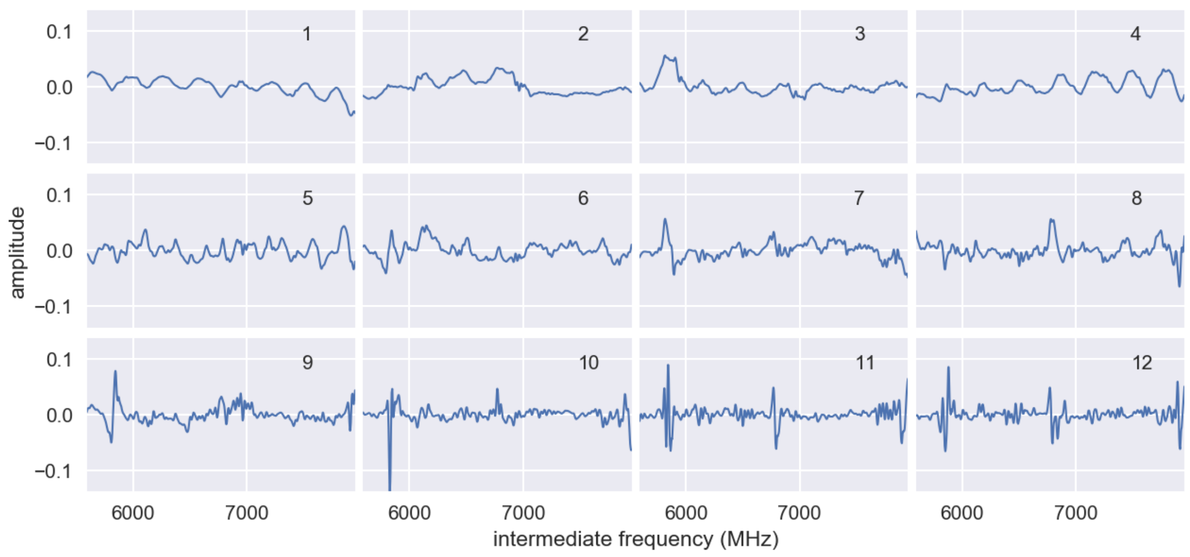


Figure 7.7: First few eigenvectors of a PCA fitted to the smoothed baselines.

7.4.1 Determining the number of components

The number of components, which is equal to the number eigenvectors and eigenvalues, determines how accurately the data can be described. In principle one could use N eigenvectors to perfectly describe N-dimensional data. However, there are two reasons why we would want to discard the least important eigenvectors. Firstly, this leads to a significant dimensionality reduction, which simplifies the problem and makes it easier to train the algorithm. Secondly, and most important for our purposes, we want the eigenvectors to represent the ESWs but not the noise.

If too few eigenvectors are used, the baseline cannot be reproduced accurately. However, if too many eigenvectors are used, there will be too many degrees of freedom. That would allow the reproduction of components in the spectrum that are not found in the stability data. Thus the number of eigenvectors should be as low as possible, while still high enough to be able to describe the majority of the variance in the data.

Figure 7.8 shows the explained variance ratio of each eigenvector, for a particular PCA transformation. The explained variance ratio is calculated as the variance in the eigenvalues for each eigenvector, divided by the sum of those variances. The figure shows that only the first few eigenvectors account for the majority of the variance in both data sets. This makes sense as these mostly describe the standing wave components. The presence of signals in the data leads to an increased variance ratio for less important eigenvectors, as the signals can be better described by the noisier eigenvectors. For this particular case, we chose to keep only the first 20 eigenvectors as these account for the majority of the explained variance.

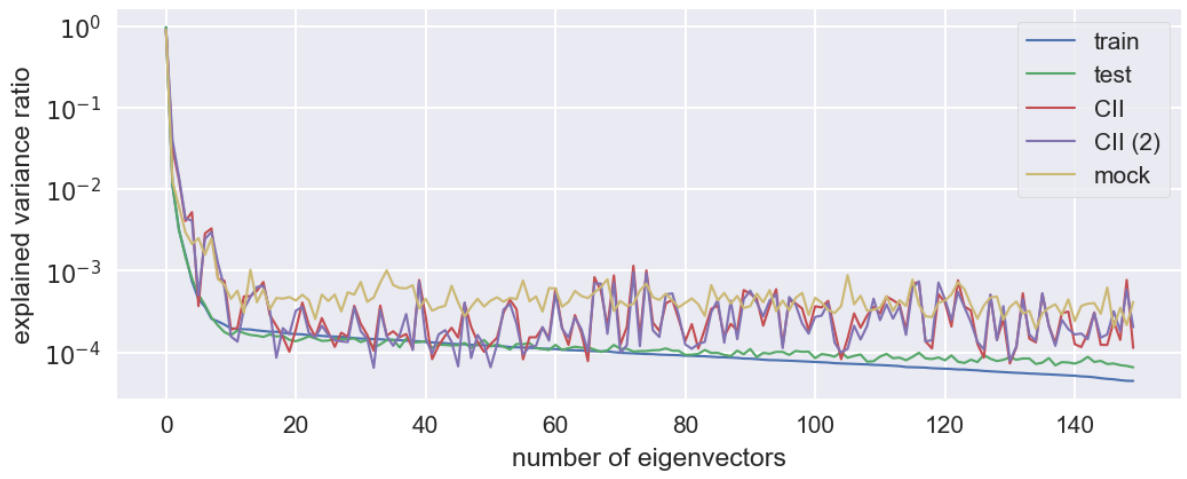


Figure 7.8: Explained variance ratio of each eigenvector, for different data sets. The stability data is divided into a training set used to fit the PCA (blue), and a test set (green). Also the explained variance in two GOT [CII] observations is included, with `obsid` 1342201649 (red) and 1342210319 (purple). For reference, the explained variance for the stability data including mock signals is also included (yellow).

7.5 Dense network

After the PCA transformation that transforms the spectra to eigenvalues, we use one more neural network that consists mostly of dense (or fully connected) layers. This network tweaks the eigenvalues and then performs an inverse PCA transform to transform the eigenvalues back to channel values. The results are again compared to the corresponding smoothed stability spectra without signals, that approximate the real baseline.

The purpose of this neural network is to correct unwanted artifacts in the spectra that occur due to the PCA transformation. Figure 7.9 shows a simplified example of the problem. If a signal is PCA transformed and then transformed back using an inverse PCA transform, some of the information is lost. This leads to inaccuracies over the entire spectrum. The signal contribution to the eigenvalues should be taken out before the inverse PCA transformation is applied, which is why we apply this second neural network to tweak the eigenvalues.

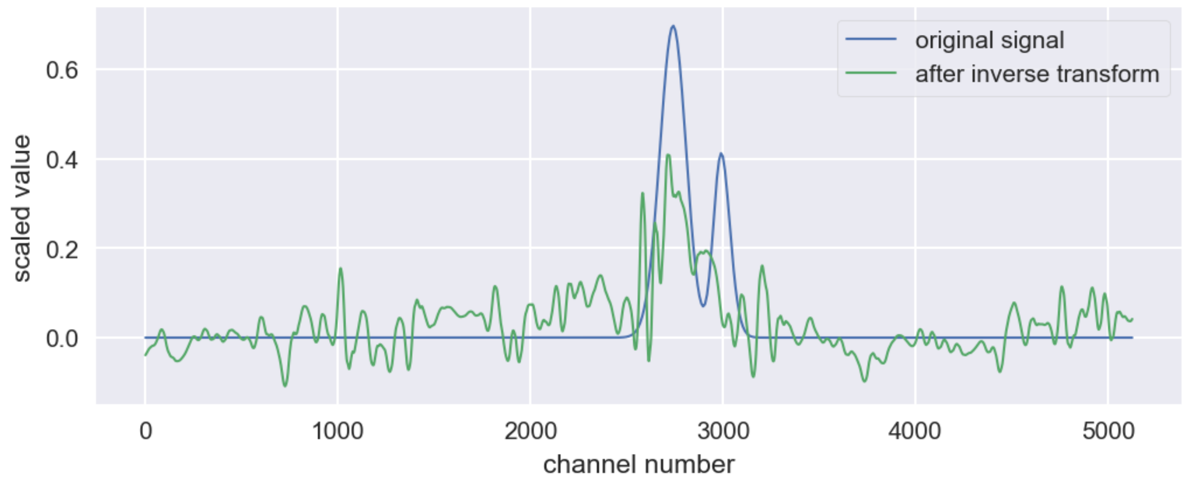


Figure 7.9: Example showing how the presence of a signal can lead to inaccuracies over the entire spectrum after the inverse PCA transformation. Some of the information has been lost because we discarded most of the eigenvectors. The few eigenvectors that are used in the PCA transformation cannot accurately describe a signal, as they mostly contain information about the baseline. This leads to noise at other locations in the spectra.

An incomplete summary of the layout of the neural network is given in Table 7.3. The Input layer of the neural network receives n_{ev} eigenvalues for each spectrum, where n_{ev} is equal to the number of components used for the PCA transformation. After the Input layer, the eigenvalues are first scaled using min-max scaling. Next, using the Concatenate layer, the eigenvalues are duplicated but with opposite signs to ensure that the neural network does not care about the sign of the spectra, as the ESWs should be independent of its sign. The resulting values with shape (None, 40, 1) are passed onto the next layers. Near the end, the Add layer adds the result to the values of the Input layer. The last layer, Tensordot, uses these predicted eigenvalues to perform an inverse PCA transformation. This converts the eigenvalues back to channel values, giving the final prediction for the baseline.

Table 7.3: An incomplete summary of the layers used for the dense neural network. The output shapes of the layers are dependent on the number of eigenvalues n_{ev} and the number of channels in the spectra n_{channels}. The last column gives the number of parameters for each layer, for which we've taken $n_{\text{channels}} = 20$. The first and final few layers have been omitted for clarity. The network was developed using the Keras [Chollet et al., 2015] with the TensorFlow [Abadi et al., 2016].

#	Layer	Output Shape	# of Params
1	Input	(None, n_{ev}, 1)	0
2	Concatenate	(None, $2n_{\text{ev}}$, 1)	0
3	Dense	(None, $2n_{\text{ev}}$, 160)	320
4	Upsampling	(None, $8n_{\text{ev}}$, 160)	0
5	Convolutional	(None, $8n_{\text{ev}}$, 160)	51360
6	Max Pooling	(None, $2n_{\text{ev}}$, 160)	0
7	Dense	(None, $2n_{\text{ev}}$, 160)	25760
8	Dense	(None, $2n_{\text{ev}}$, 1)	161
9	Split	(None, n_{ev}, 1, 2)	0
10	Add	(None, n_{ev}, 1)	0
11	Tensordot	(None, n_{channels}, 1)	0

Chapter 8

Results

To gain an understanding in which cases our algorithm would be applicable or preferred, we compared the results with the standard HIFI pipeline. We applied the algorithm to two sets of data; the stability data which was partially used to train the data, and [CII] observations from the GOT C+ survey [Langer et al., 2010]. Both of these data sets are described in Section 7.2. As explained in Section 3.6, the data was processed using the Herschel Interactive Processing Environment (HIPE) up to level 1 using the Jython script given in Chapter 12 of the appendix. Depending on whether the data was used as input for our algorithm or to compare it with the pipeline method, the preprocessing either excluded or included the pipeline standing wave correction. Note that these spectra are still in dual-sideband intermediate frequency (IF), not in sky frequency. Thus, each frequency in the IF corresponds to two physical frequencies as is explained in Section 3.2.

8.1 Stability data

Figure 8.1 gives the results on various mock spectra. The left column shows the mock spectra used as input, the baselines used to create those mock signals, and the baseline predictions of our algorithm. The right column shows the mock spectra after the baseline correction, and the true mock signal. In general, the results on the mock spectra are excellent. While some artifacts are present in some cases, for instance around 5.8 GHz in the first and second spectra, the mock spectra can be very clearly resolved. In the third spectrum the ESW is very well fitted, despite the broad signal covering two ESW bumps. Similarly, the fifth spectrum has multiple wide and overlapping signals extending over half the spectrum, but the algorithm does not interpret these to be ESWs.

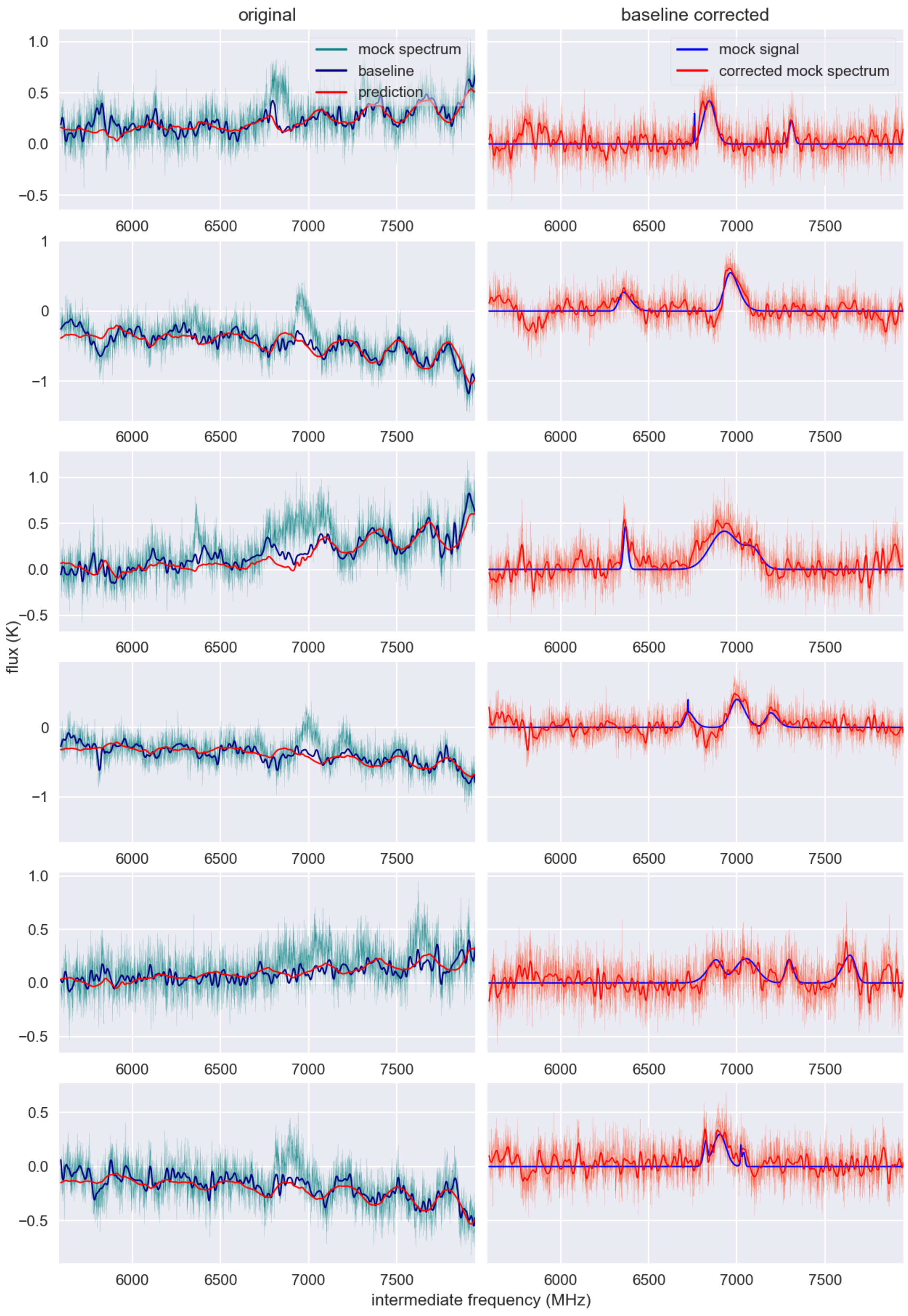


Figure 8.1: The results of our algorithm on a selection of mock spectra, which are based on the stability data. The left panels show the original mock spectra, their corresponding baseline, and the baseline prediction of our algorithm. The right panels show the same mock spectra after subtracting the baseline prediction, and compares it to the mock signal.

Figure 8.2 shows the intermediate results of each part of the algorithm, on a single mock spectrum. In this mock spectrum, the CNN does a good job of smoothing the baseline, and the strong signal at 7.1 GHz is largely ignored. After the eigenvalues have been found and the dense neural network applied some slight corrections, the mock signal is entirely ignored in the baseline fit. In the end result, the mock signal is clearly resolved. However, it does seem like there is some very low frequency noise present that could be an artifact of the baseline correction. Two small but wide bumps can be seen around 5.8 and 6.4 GHz. Also, some narrow bumps are visible, for example at 6.9 and 7.7 GHz. These are likely the unimportant high frequency components of the artificial baseline, which are the result of smoothing the noise in the stability spectra.

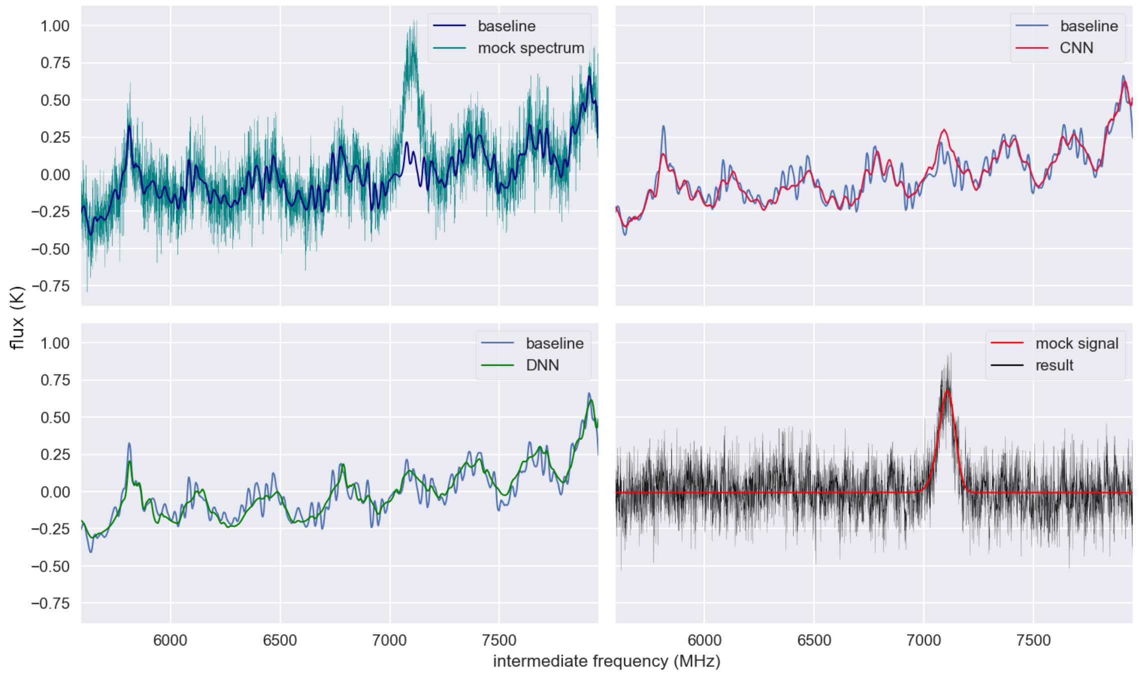


Figure 8.2: Intermediate results on a mock spectrum, at each step in the algorithm. The top left panel shows the mock spectrum, and the corresponding smoothed stability spectrum that was used for the baseline. The top right panel compares the baseline with the result of the CNN, after it is applied to the mock spectrum. The bottom left panel compares the baseline to the result of the PCA and dense neural network. The bottom right panel gives the original mock signal, compared to the baseline corrected mock spectrum. For easy comparison, in this figure, all spectra have the same data shape and are scaled to the physical amplitude in Kelvin.

8.2 CII observations

To compare the performance of our algorithm with the standard HIFI pipeline method described by Kester et al. [2014], we used Herschel CII observations from Langer [2007]. The observations were exclusively taken in band 7b. Figure 8.3 shows the results on the H polarization, and Figure 8.4 shows the results on the V polarization. In both figures, the first column gives the mean over all integrations, for each observation. The second and third column show the same spectra, after the baseline was corrected by our method or the pipeline method.

For the H polarization, our algorithm fits the baseline noticeably better than the pipeline method in the first, second, and second to last spectra. The third spectrum looks comparable, but slightly worse than the pipeline. These spectra have a baseline which is overall very flat and have nearly no ESWs, though they do have a small bump at the left edge. Unfortunately, the fourth, fifth and sixth spectra have considerable artifacts. The ESWs are slightly reduced, but instead, the baseline is much more erratic. In the last spectrum, only the peaks of the ESWs are left, while the baseline in between seems very well fitted. This is most likely an artifact from the CNN, which in some cases tends to smooth out the peaks of the ESWs. This indicates that the CNN should be optimized further.

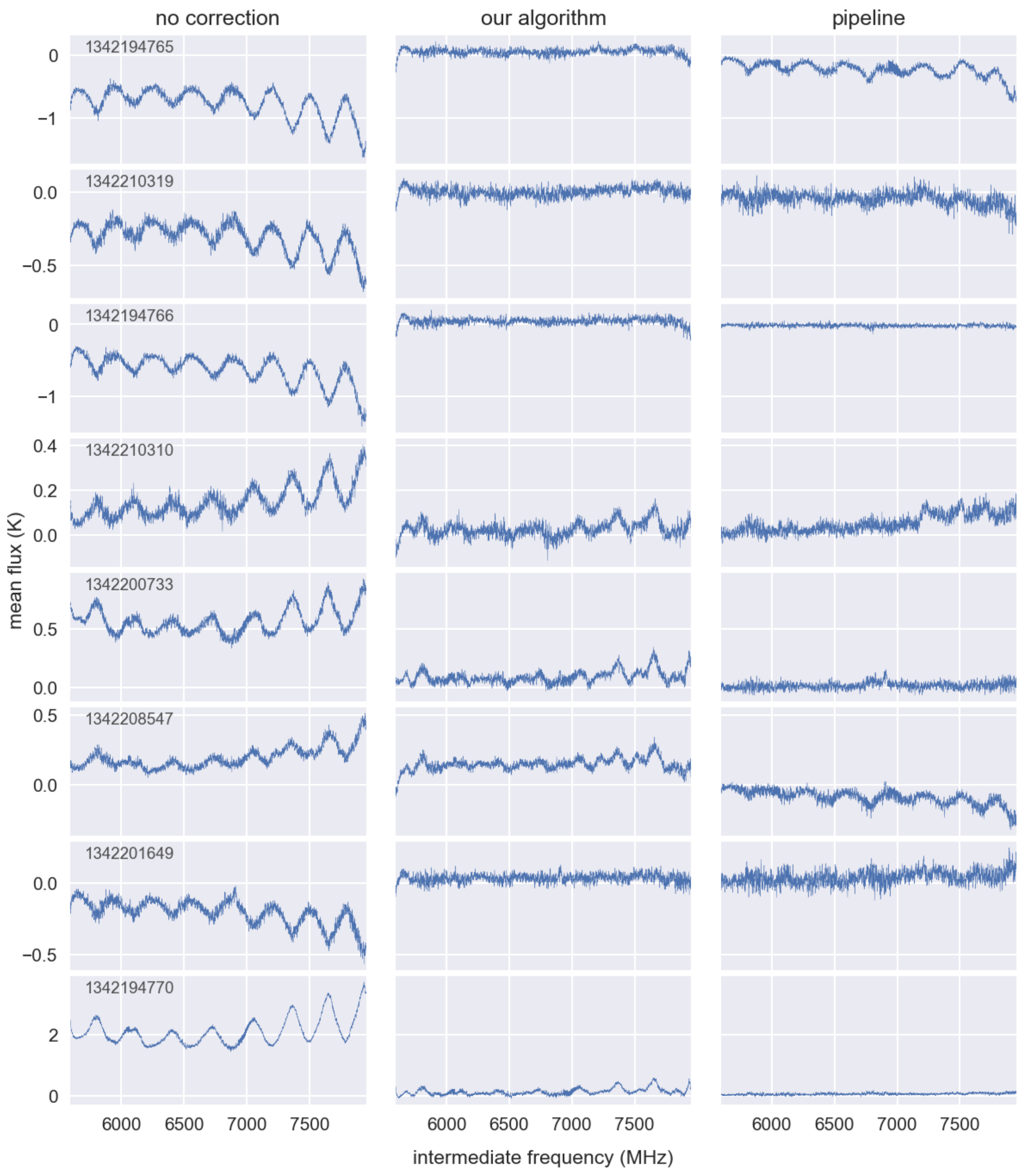


Figure 8.3: Results of our ESW removal method versus those of the pipeline, on 8, H-polarization CII observations from the GOT C+ survey [Langer et al., 2010]. The `obsid` of each observation is given in the top left. Each observation shown is the mean of all integrations.

For the V polarization, the results are slightly worse for both methods. While our results seem to be better for the sixth and last spectra compared to the pipeline method, neither of those have a very flat baseline, whereas the pipeline method manages to remove the majority of the ESW noise in 3/4 spectra. Most of the corrected spectra have strong drop-offs at the lower frequencies. The second to last spectrum especially stands out, as it shows very strong, irregular features in the baseline corrected spectrum, despite the original spectrum having fairly small ESWs. This is probably the result of these strong drop-offs, as the PCA step is very sensitive to outliers. If the individual integrations contain drop offs that are not removed by the CNN, this will cause very erratic results.

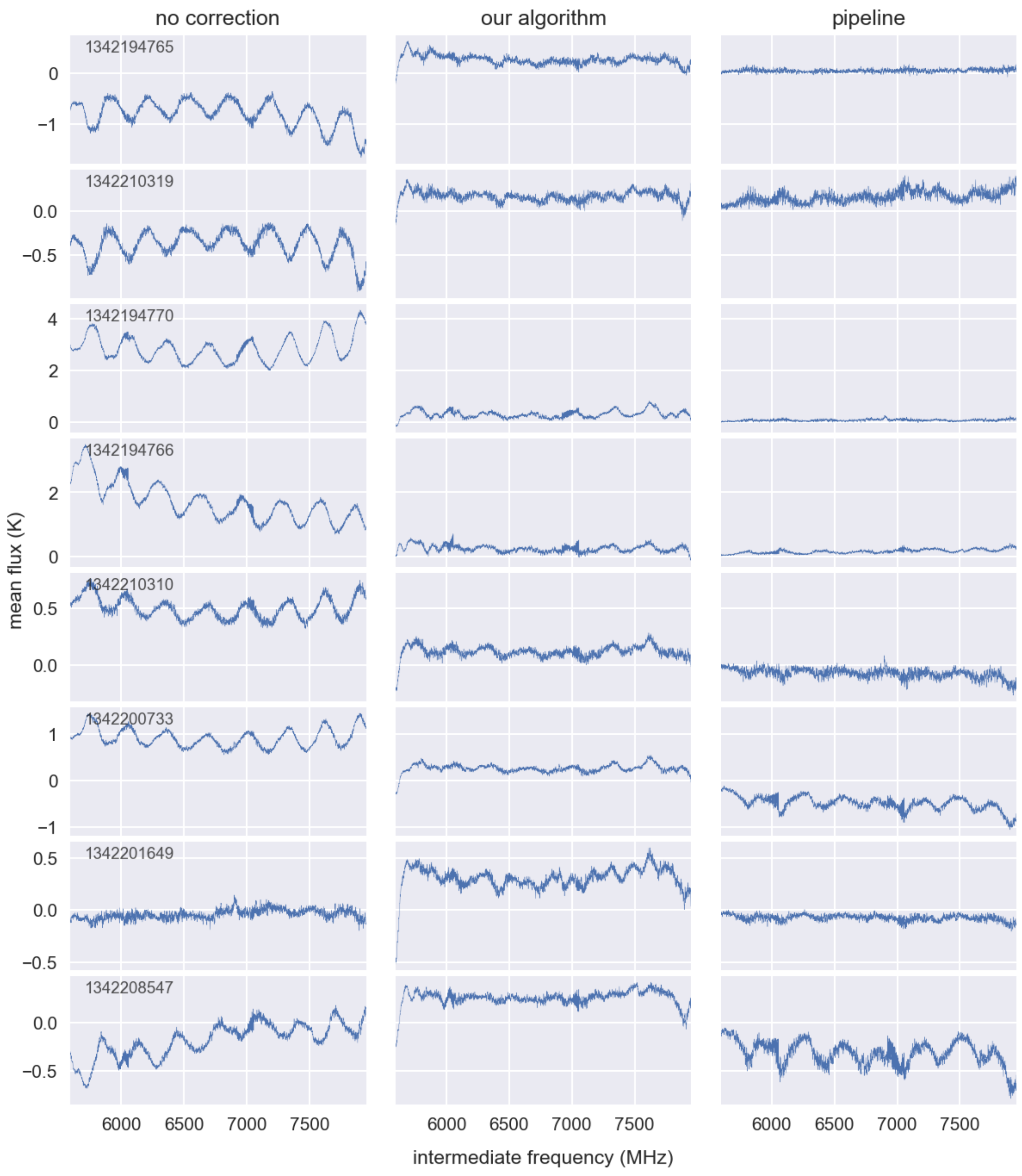


Figure 8.4: Same as Figure 8.3, but for the V polarization.

Chapter 9

Analysis

We need quantitative methods to test how well the algorithm performs. This is a tricky task, as we don't know the true baseline for any real spectrum. One way to check how important the baseline is, is to compare the mean absolute values of the [CII] spectra, with and without correction. This is done in Section 9.1. However, a single number does not give us much information. In order to get a more intuitive understanding on the accuracy of our algorithm, we used stability data to generate mock spectra, as detailed in Section 7.2.2. In each of those mock spectra, we know exactly what the baseline component is, and what the signal component is. During the training phase, we used the loss to assess the accuracy of the algorithm. We chose the loss function to be the mean squared error between the stability baseline and the prediction of the algorithm. In general, a low loss means that the algorithm works well, as the output of the algorithm is close to the baseline from the stability data. However, the loss does not tell us how well the algorithm works on new data, or how well we can recover an emission line from a spectrum. In Equation (9.2), we apply a reduced chi squared analysis to assess how well signals of varying signal to noise ratios can be recovered, with and without ESW correction. Section 9.4 explores how the number of principal components influences the results.

Based on the [CII] results discussed in Section 8.2, we can quantitatively conclude that in most of the cases for the H polarization, the pipeline method gives a flatter baseline. However, there are also some cases where our algorithm does better. In other cases the baseline looks slightly erratic, even though the ESWs may be reduced. If our algorithm introduces highly irregular features in the baseline, it would be difficult to distinguish signals from the baseline.

For the V polarization, most results are not that good. The main culprit seems to be the strong drop-offs at the lower frequencies of the spectra. The PCA will be strongly affected by outliers, so if spectra contain such drop-offs, the results will be erratic.

9.1 Absolute differences

Using the results from Figure 8.3, we can calculate the mean absolute value in each spectrum to estimate how important the baseline is. First, though, a third order polynomial is fitted and subtracted from the original spectrum such that the spectrum has zero mean and is relatively flat (apart from the ESWs). Then, taking the mean absolute value over the entire spectrum for each observation in the [CII] data, we get the values in Section 9.1. Based on this metric, in all of the observations, our algorithm gives a flatter baseline than when no correction is applied. In some of these samples, the amplitudes of the ESWs are more than a factor 3 smaller. In 5 out of the 8 cases, our algorithm gives a lower mean absolute value than the pipeline method.

Table 9.1: Mean absolute values of each of the H polarization [CII] spectra, before and after baseline correction.

obsid	no correction	our algorithm	pipeline
1342194765	0.133	0.035	0.068
1342210319	0.059	0.024	0.032
1342194766	0.105	0.029	0.015
1342210310	0.038	0.026	0.032
1342200733	0.070	0.039	0.022
1342208547	0.030	0.029	0.034
1342201649	0.047	0.021	0.035
1342194770	0.266	0.080	0.024

9.2 Signal recovery

In this section we try to evaluate how much better a signal can be distinguished after we remove the ESWs from the spectrum. To do this, we generate mock spectra, as described in Section 7.2.2. Each mock spectrum has a known baseline and mock signal component. After we apply our baseline correction to a mock spectrum, we can compare the result with the mock signal. If the difference between the baseline corrected mock spectrum and the mock signal is small, then the baseline is successfully removed. We evaluate the reduced chi square for mock signals with various signal to noise ratios. This tells us how well a signal could be recovered, given the signal to noise of the signal.

We first generate a set of mock signals and add these to the set of baselines in the test stability data. These baselines have not been used to train the algorithm. The mock signals that are added to the baselines have varying signal to noise ratios (SNRs). We define the SNR as the power of the mock signal divided by the power of the noise. Or, in other words, the sum of the absolute values of the mock signal divided by that of the noise, as given by Equation (9.1). The SNR is calculated over a small interval in the spectrum, near the mock signal. We use the absolute sum over channel values of the mock signal, given by $x_{\text{signal,i}}$, and divide by the mean value of the noise. To calculate the SNR, we only consider the channels where the value of the mock signal is more than 1% of the peak value of the mock signal. Although this number is somewhat arbitrary, this way we integrate over the majority of the signal.

$$\text{SNR} = \sum_i \left| \frac{x_{\text{signal,i}}}{\bar{x}_{\text{noise}}} \right|, \qquad \text{where } x_{\text{signal,i}} > 0.01 \max(x_{\text{signal}}) \tag{9.1}$$

We can then calculate the reduced chi squared of the fit, χ^2_ν, as a function of the SNR. The χ^2_ν is defined by Equation (9.2). Here, ν is the degree of freedom, which is equal to the number of channels minus one. E is the expected signal, for which we take the mock signal without noise. O is the observed signal, for which we take the mock spectrum after baseline correction. We take the sum of the differences for each channel i, where we again only consider the channels near the mock signal.

$$\chi^2_\nu = \frac{1}{\nu} \sum_i \frac{(O_i - E_i)^2}{E_i} \tag{9.2}$$

The χ^2_ν can be loosely interpreted to be the minimum possible χ^2_ν that a signal fit could give using our baseline correction. Normally, $\chi^2_\nu < 1$ would mean that a model is overfitting, but in our case it means that the baseline correction will not have a significant effect on the accuracy of a signal fit. A $\chi^2_\nu \geq 1$ means that the baseline does notably affect the signal.

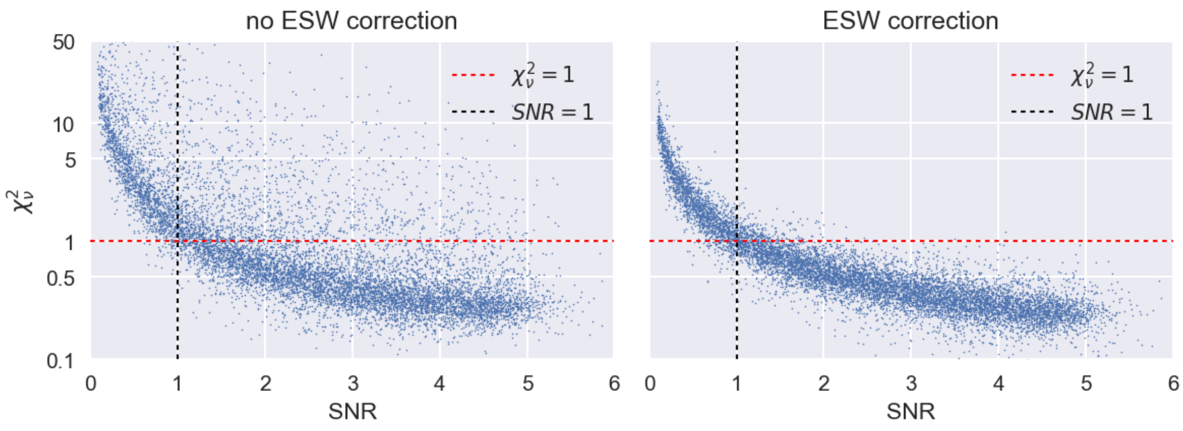


Figure 9.1: Semi-logarithmic plot of the reduced chi square (χ_ν) versus the SNR of 10000 stability spectra with added mock signals. The chi square is taken as the difference between the mock signal, and the corresponding mock spectrum with baseline. In the left figure no baseline correction is applied to the mock spectra, except for subtracting the local mean. The right figure uses our algorithm to correct the baselines.

Plotting the χ_ν^2 versus the SNR for many spectra gives a distribution similar to that shown in Figure 9.1. This figure shows the χ_ν^2 for spectra with and without ESW correction. Compared to using no baseline correction at all, our algorithm mostly lowers the uncertainty in the χ_ν, having far fewer outliers with high χ_ν^2. These are mostly the spectra with strong baselines.

Figure 9.2 shows the same figure, but using smoothed [CII] spectra as baselines instead of smoothed stability spectra. Although the [CII] spectra contain signals and are thus not completely unbiased, they can give us a rough estimate on how well signals are recovered in new data. For signals of all SNR's, the number of outliers with high χ_ν^2 is again reduced. For low SNR's, the average χ_ν^2 is slightly lower, but for high SNR it is actually higher after the baseline correction. Thus, the algorithm may confuse strong signals in the spectra for baselines, leading to a worse fit.

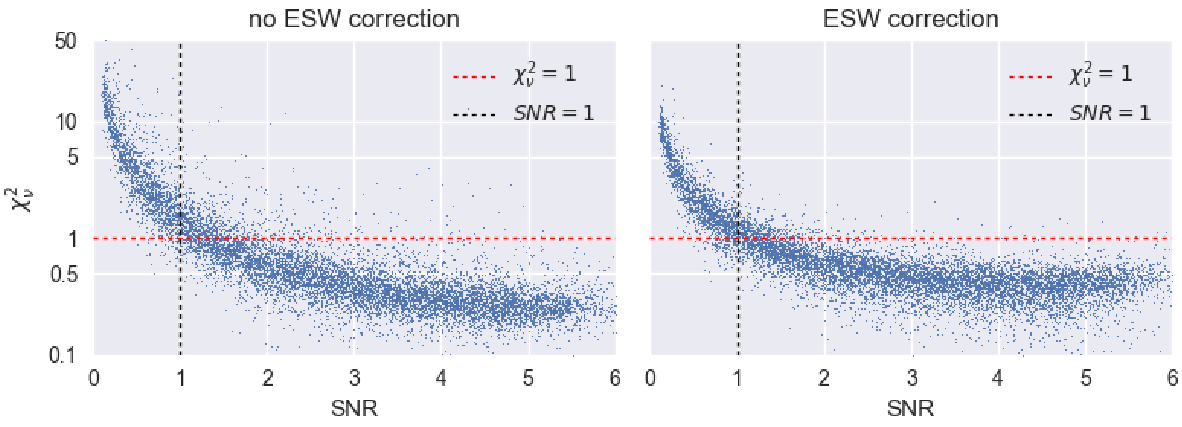


Figure 9.2: Same as Equation (9.2), but with mock spectra that use smoothed [CII] spectra as baselines instead of stability spectra.

9.3 Effect of rebinning size

As explained in Section 7.3, we rebin the spectra to a smaller size in order to speed up the training process of the CNN. However, rebinning the data too much may result in bad results on new data. Figure 9.3 compares the intermediate results of the CNN on the [CII] data. The output of the CNN should be slightly different, but similar to the smoothed spectrum. Both CNN's produced excellent results on the stability data, not only on the training set but also on the testing set. However, the 256 channel CNN does a very poor job on the [CII] data. This could be a sign that the CNN is overfitted to the stability data, even though it performs well on both the train and test set. In any case, a higher resolution CNN adapts better to new data.

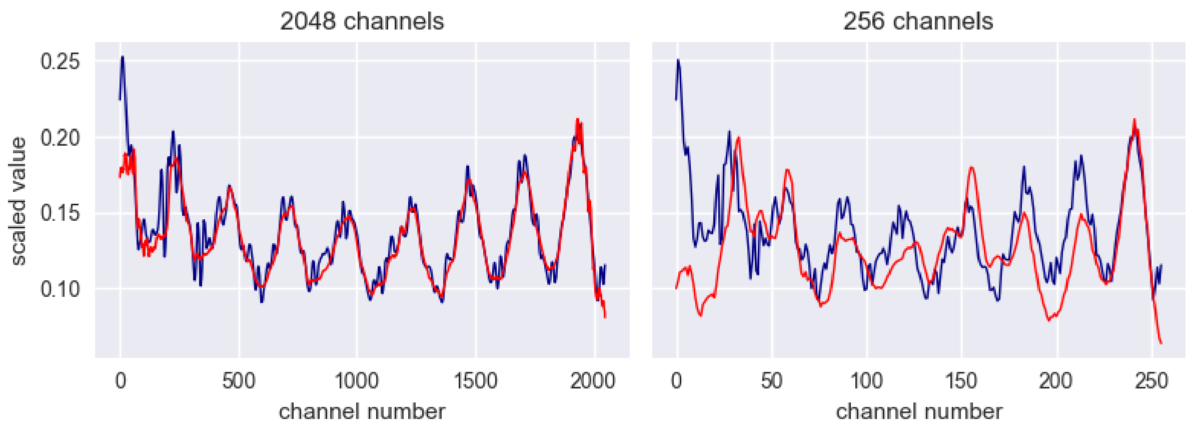


Figure 9.3: A representative spectrum from the [CII] data. The blue line is the smoothed spectrum, and the red line is the output of the CNN. The two panels show the intermediate result for two CNN's trained on 2048 and 256 channels, respectively. The output of the CNN should be relatively similar to the smoothed spectrum, but this is not the case for the CNN using only 256 channels.

9.4 Effect of number of PCA components

Figure 9.4 shows the influence of the number of PCA components on the baseline prediction, for two [CII] observations. Using fewer components poses more restrictions on the shape of the baseline, as fewer eigenvectors can be used to describe the baseline. However, although using just a few components worked great on the stability data, this will mostly result in a relatively poor fit on the [CII] data. Even though the standing waves are still correctly found in the two examples in Figure 9.4 that use only 5 components, the peaks and crests of the ESWs are inaccurate.

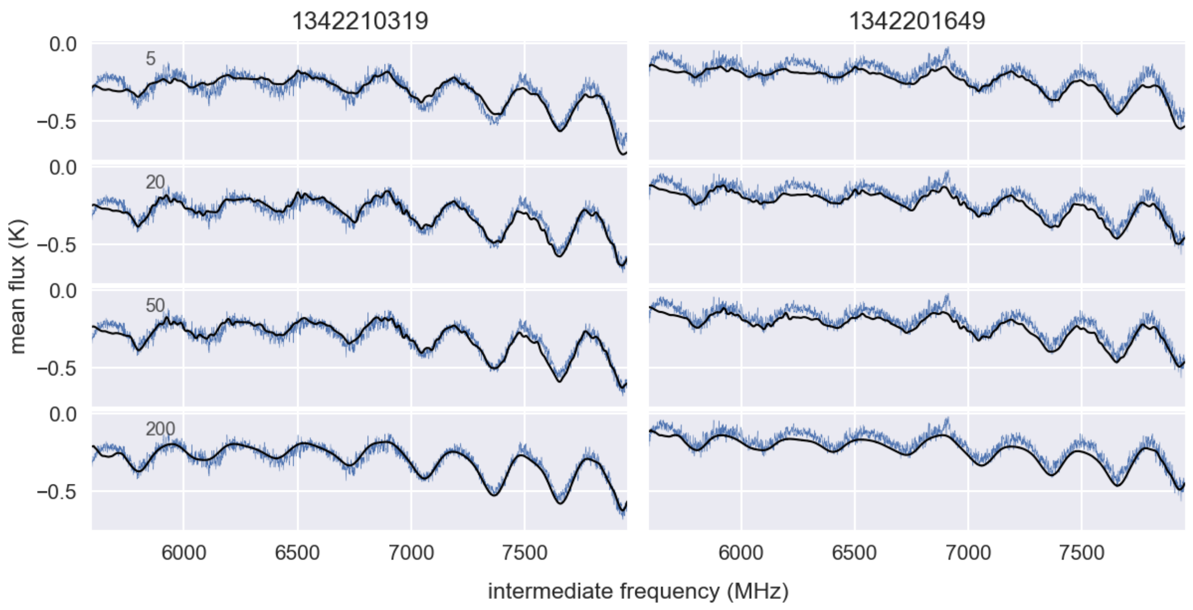


Figure 9.4: Baseline prediction for two observations from the [CII] data, for a various number of PCA components; 5, 20, 50 and 200 principal components.

As the number of components is increased, the output starts to resemble that of the CNN and thus the result is smoother. However, this also means that the PCA does not get rid of signals that are not removed by the CNN. If many components are used, there is a large degree of freedom to represent signals in the spectra, which is not what we want. Thus the number of components should be somewhere in between.

Chapter 10

Discussion

10.1 Applications

The algorithm is intended for general use on spectrometers using HEB mixers, not just HIFI. An example of a far-infrared spectrometer on which it could be applied would be GUSTO, which is an upcoming balloon observatory to be launched this year (2021). Although HIFI did map the Milky Way disk in [CII] emission, it did so at a sparse spatial resolution. GUSTO will provide a fully sampled map of three important emission lines; 158 μm [CII], 205 μm [NII] and 63 μm [OI]. In contrast to HIFI's stable environment at the L_2 Lagrange point in space, GUSTO will operate in the less stable upper atmosphere above Antarctica. Thus, the electrical components of GUSTO will experience more movements, leading to less stable ESWs. This is where our method could be useful: our method does not require the exact same ESWs to previously appear in stability measurements.

Our method should be more versatile than the HIFI pipeline method by Kester et al. [2014], as it allows for more flexibility in the shape of the baselines. Their method assumes that the shape of the ESW in a spectrum is exactly the same as one of their ESW templates. While our algorithm also relies on a set of "templates" (eigenvectors), we use a linear combination of these instead of a single template. This allows the shape of the baselines to change to some degree. However, the algorithm will need to be further tested on other data sets where the standing waves change more rapidly over time.

10.2 Requirements and Limitations

Our algorithm requires a data set containing repeated observations of source free regions, like the stability data described in Section 4.1. Without such data, it is not possible to train the CNN to ignore signals in the spectra. Moreover, we need the smooth baselines in order to get representative eigenvectors for the PCA transformation. If any signals are present in this data, they will also appear in some way in the eigenvectors, which should be avoided.

Mostly due to the PCA transformation, the method is quite sensitive to outliers that are not taken care of by the smoothing or the CNN. Examples of these are drop-offs at the end of spectra. It is therefore very important that the CNN works properly on a particular data set, and that the kernel size is adapted based on the width of the ESWs, as described in Section 7.3. If the width of the kernel is too small, the CNN may not filter out spectral lines, while too large kernel size may filter out parts of the baseline.

Although our method is designed to work on data where ESWs rapidly change shape, the training data should still contain most of the baseline features. As ESWs change over time, we recommend that the observation dates of the training data should be similar to that of the data that the algorithm is applied to. Care must be taken if this is not the case. Also, any signals found in observational data should be represented by at least some of the mock signals used for training. Otherwise, the CNN may misidentify such signals as being part of the baseline.

Lastly, our method also assumes several physical simplifications. In reality, the shape of the baseline is influenced by the amount of measured radiation. A bright source may cause additional optical and electrical standing waves that affect the spectrum. Because we only use stability data without strong (real) astronomical sources as our training data, the algorithm may work less accurately for very bright sources.

Chapter 11

Conclusion

We applied a series of machine learning methods to correct electrical standing waves (ESWs) in HIFI spectral observations. This is a type of instrument noise that causes a rippling baseline and is especially prevalent in HIFI bands that use an HEB mixer. Our approach enables us to predict a baseline, even if such a baseline has not been observed before. Therefore, our method is intended for general use on other heterodyne spectral instruments that use HEB mixers, where the baseline varies quickly over time. Examples of these are GUSTO, or the GREAT instrument of SOFIA. In order to remove the ESWs from a spectrum, we predict the ESW contribution from a spectrum, and subtract this baseline prediction. Using a set of HIFI's stability data to represent the ESWs, we find a set of eigenvectors using a principal component analysis (PCA). We use a linear combination of these eigenvectors to find the baseline in a spectrum.

The first part of the algorithm, a convolutional neural network (CNN), is intended to smooth the spectra and ignore local features that are not seen in the stability data. This works well on narrow lines, but does not remove broad features. Next, we apply the PCA transformation, which describes the baseline as a linear combination of eigenvectors. The purpose of the PCA transformation is to remove features that are not found in the baseline. Whereas the CNN only removes features base on their shape, the PCA uses the global shape of the baseline to remove anomalous features. Unfortunately, the PCA is quite sensitive to outliers such as strong signals or drop-offs at the edge of the spectrum. To mitigate these effects, we apply another neural network.

Using a reduced chi squared (χ^2_ν) analysis, we have shown that the algorithm works outstandingly well on stability data with mock signals. The number of spectra with very high χ^2_ν was significantly reduced for signals with any signal to noise ratio. The algorithm was also compared with the pipeline method [Kester et al., 2014] for two [CII] spectra from the GOT C+ survey [Langer et al., 2010]. Both methods showed highly varying results, often in different spectra. In the majority of the observations, the baselines were reduced. However, in some cases our algorithm introduced erratic features to the baseline. This often happened whenever the spectrum contained strong drop-offs at the edges of the spectra. As the PCA is sensitive to outliers, outliers lead to a bad baseline fit. Nonetheless, these results showed that our method is feasible.

Although we have only tested it on [CII] observations which were exclusively done in band 7b, our results should be largely applicable to other bands as well. To further validate the algorithm, it will instead need to be tested on data sets with more rapidly changing ESWs. Our main point of concern regarding the current results is the reliability of the CNN, which tended to smooth out the peaks of ESWs in the [CII] data. On the other hand, the CNN also managed to largely remove the features of mock signals. If the CNN could be optimized better, it might be feasible to use a CNN-only approach without applying PCA. Although this would work poorly on spectra with wide features, it would not require the ESWs to be stable over time.

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

LD Anderson, TM Bania, JM Jackson, DP Clemens, M Heyer, R Simon, RY Shah, and JM Rathborne. The molecular properties of galactic h ii regions. *The Astrophysical Journal Supplement Series*, 181 (1):255, 2009.

Michael G Burion. Excitation of molecular clouds and the emission from molecular hydrogen. *Australian Journal of Physics*, 45(4):463–486, 1992.

Francois Chollet et al. Keras, 2015. URL https://github.com/fchollet/keras.

David Teyssier, ESA. Hifi handbook, 2016. URL http://herschel.esac.esa.int/twiki/pub/HSC/HiFi/hifi_handbook.pdf. [Online; accessed September 9, 2020].

Th De Graauw, FP Helmich, TG Phillips, J Stutzki, E Caux, ND Whyborn, P Dieleman, PR Roelfsema, Henri Aarts, R Assendorp, et al. The herschel-heterodyne instrument for the far-infrared (hifi). *Astronomy & Astrophysics*, 518:L6, 2010.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

Paul F Goldsmith, Umut A Yıldız, William D Langer, and Jorge L Pineda. Herschel galactic plane survey of [n ii] fine structure emission. *The Astrophysical Journal*, 814(2):133, 2015.

Christopher Groppi, Christopher Walker, Craig Kulesa, Dathon Golish, Jenna Kloosterman, Sander Weinreb, Glenn Jones, Joseph Barden, Hamdi Mani, Tom Kuiper, et al. Supercam: A 64 pixel heterodyne array receiver for the 350 ghz atmospheric window. In *20th Int. Space Terahertz Technol. Symp*, pages 90–96, 2009.

Hafellner, M. Object detection, 2019. URL https://bitmovin.com/object-detection/. [Online; accessed September 1, 2020].

Daniel Ronan Higgins. *Advanced optical calibration of the Herschel HIFI heterodyne spectrometer*. PhD thesis, National University of Ireland Maynooth, 2011.

Ronan D Higgins and Jacob W Kooi. Electrical standing waves in the hifi heb mixer amplifier chain. In *Terahertz Technology and Applications II*, volume 7215, page 72150L. International Society for Optics and Photonics, 2009.

David J Hollenbach and AGGM Tielens. Photodissociation regions in the interstellar medium of galaxies. *Reviews of Modern Physics*, 71(1):173, 1999.

Do Kester, Ian Avruch, and David Teyssier. Correction of electric standing waves. In *AIP Conference Proceedings*, volume 1636, pages 62–67. American Institute of Physics, 2014.

W. Langer. KPOT_wlanger_1: State of the Diffuse ISM: Galactic Observations of the Terahertz CII Line (GOT CPlus). Herschel Space Observatory Proposal, October 2007.

WD Langer, T Velusamy, JL Pineda, PF Goldsmith, D Li, and HW Yorke. C+ detection of warm dark gas in diffuse clouds. *Astronomy & Astrophysics*, 521:L17, 2010.

Gregory R Lee, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron O'Leary. Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237, 2019.

Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

Malik, F. Neural network, 2019. URL https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da. [Online; accessed September 2, 2020].

Michael Nielsen. Neural networks and deep learning, Dec 2019. URL http://neuralnetworksanddeeplearning.com/index.html. [Online; accessed November 15, 2020].

Stephan Ott, Herschel Science Centre, and European Space Agency. The herschel data processing system-hipe and pipelines-up and running since the start of the mission. *arXiv preprint arXiv:1011.1209*, 2010.

Vojtech Pavlovsky. Introduction to convolutional neural networks, Dec 2019. URL https://www.vojtech.net/posts/intro-convolutional-neural-networks/. [Online; accessed December 3, 2020].

Jorge L Pineda, William D Langer, Thangasamy Velusamy, and Paul F Goldsmith. A herschel [c ii] galactic plane survey-i. the global distribution of ism gas components. *Astronomy & Astrophysics*, 554:A103, 2013.

James E Rhoads, Sangeeta Malhotra, Sahar Allam, Chris Carilli, Françoise Combes, Keely Finkelstein, Steven Finkelstein, Brenda Frye, Maryvonne Gerin, Pierre Guillard, et al. Herschel extreme lensing line observations: Dynamics of two strongly lensed star-forming galaxies near redshift z= 2. *The Astrophysical Journal*, 787(1):8, 2014.

Heiko Richter, Martin Wienold, Lutz Schrottke, Klaus Biermann, Holger T Grahn, and Heinz-Wilhelm Hübers. 4.7-thz local oscillator for the great heterodyne spectrometer on sofia. *IEEE Transactions on Terahertz Science and Technology*, 5(4):539–545, 2015.

NJ Rodriguez-Fernandez, J Braine, N Brouillet, and F Combes. [cii] emission and star formation in the spiral arms of m 31. *Astronomy & Astrophysics*, 453(1):77–82, 2006.

Sharma, A. Activation functions, 2017. URL https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/. [Online; accessed September 28, 2020].

RF Shipman, SF Beaulieu, D Teyssier, P Morris, M Rengel, C McCoey, K Edwards, D Kester, A Lorenzani, O Coeur-Joly, et al. Data processing pipeline for herschel hifi. *Astronomy & Astrophysics*, 608: A49, 2017.

JDT Smith, Kevin Croxall, Bruce Draine, Ilse De Looze, Karin Sandstrom, Lee Armus, Pedro Beirão, Alberto Bolatto, Mederic Boquien, Bernhard Brandl, et al. The spatially resolved cooling line deficit in galaxies. *The Astrophysical Journal*, 834(1):5, 2016.

SaiGayatri Vadali. Day 10: Dimensionality reduction with pca and t-sne in r, Jan 2018. URL https://medium.com/@TheDataGyan/dimensionality-reduction-with-pca-and-t-sne-in-r-2715683819. [Online; accessed June 28, 2021].

Mark G Wolfire, Christopher F McKee, David Hollenbach, and AGGM Tielens. Neutral atomic phases of the interstellar medium in the galaxy. *The Astrophysical Journal*, 587(1):278, 2003.

Acknowledgements

I'd like to thank my supervisor, Russ Shipman for putting his thoughts into this project, and for his casual meetings. We both started out somewhat new to the subject of machine learning, so I sincerely thank Youngmin Seo for his comments on this part of the project. His abundant feedback from the biweekly meetings, as well as his contributions to the development of the code, were extremely helpful in steering this project in the right direction.

Next, I'd like to thank Mark (mk. I and mk. II), who was my daily source of entertainment since we first watched Kung Fury, and throughout the project despite the lockdown. With me he generously shared his apples and even his passwords, and I could not have finished my degree without his socialist ideals. One day we will follow in Che's footsteps together. I'd also like to thank everyone else who kept me company in the therapy room this year and before, with special thanks to Thom who unexpectedly managed to out-live the first days of corona. Lastly, I'd like to thank my mother, and my father whose cell theory will hopefully soon revolutionize the entirety of science. They have both always supported me.

Chapter 12

Appendix

Jython preprocessing script

The following script was used in HIPE to process the stability data to level 1 of the HIFI pipeline. In order to obtain data on which the pipeline method for correcting ESWs was applied, the `ignore` parameter of `doHebCorrection` should be set to `False`. The parameter `obs_noHEB` should point to the data, which is loaded in using the `.xml` file `ObservationContext`, and the parameter `obsid` should point to the observation id number.

```
obs_noHEB = hifiPipeline(obs=obs, apids=['WBS-V','WBS-H'], upToLevel=1.0, computeRms=
    ↪ False, \
processOff=False, browseProduct=False, \
params={
    'doOffSubtract': {'useCalTree': False, 'ignoreFlagged': False, 'mode': u'addOff',
        ↪ 'interpolator': u'trunc_linear', 'validatorTolerance': None, 'ignore': '
        ↪ False'}, \
    'mkOffSmooth': {'useCalTree': False, 'widthUnit': u'MHz', 'mode': u'filter', '
        ↪ filter': u'Gaussian', 'width': -1.0, 'ignore': 'False'}, \
    'doSidebandGain': {'ignore': 'True'}, 'mkFlagSummary': {'ignore': 'True'}, \
    'checkFreqGrid': {'ignore': 'False', 'tolerance': 5.0}, \
    'doCleanUp': {'datasetSize': 1000, 'ignore': 'False', 'mergeDatasets': 'True'}, \
    'doVelocityCorrection': {'ignore': 'True'}, \
    'doFreqGrid': {'scheme': u'euler', 'ignore': 'False'}, \
    'doAntennaTemp': {'useCalTree': False, 'ignore': 'False'}, \
    'mkRef': {'ignore': 'False'}, \
    'mkFluxHotCold': {'indicator': u'buffer', 'useCalTree': False, 'isABBA': 'True', '
        ↪ validatorTolerance': None, 'ignore': 'False'}, \
    'checkPlatforming': {'ignore': 'True'}, \
    'doFluxHotCold': {'useCalTree': False, 'interpolator': u'trunc_linear', '
        ↪ validatorTolerance': None, 'ignore': 'True'}, \
# Turn HebCorrection off with ignore = True
    'doHebCorrection': {'redoRequired': 'True', 'ignore': 'True'}, \
    'doHpbw': {'ignore': 'True'}, \
    'mkRms': {'ignore': 'True'}, \
    'doChannelFlags': {'propagateFlag': False, 'clearFlags': False, 'ignore': 'True'},
        ↪  \
    'doFilterLoads': {'filterMethod': u'cubic_splines', 'scheme': u'scheme1', '
        ↪ splinePointsPerKnot': 140.0, 'relativeFilterParams': 'True', 'fftFilterMax'
        ↪ : 43.3333333333, 'fftFilterMin': 23.3333333333, 'ignore': 'False'}, \
    'mkFreqGrid': {'envelope': False, 'ignore': 'True'}, \
    'doAvg': {'return_single_ds': 'True', 'preserveGroups': 'True', 'ignore': 'True'},
        ↪  \
    'doChannelWeights': {'extrapolate': False, 'interpolator': u'TruncuatedLinear', '
        ↪ width': 20, 'ignore': 'False', 'definition': u'radiometric', 'smoothing': u
        ↪ 'Gaussian'}, \
    'mkUncertaintyTable': {'ignore': 'True'}, \
    'mkSidebandGain': {'ignore': 'True'}, \
    'doRefSubtract': {'indicator': u'pattern', 'useCalTree': False, 'startWithRef': '
```

```
        ↪ True', 'isABBA': 'True', 'offStartWithOpp': 'True', 'validatorTolerance':
        ↪ None, 'ignore': 'False', 'offStartsWithOpposite': False}
    })

# V
obs_noHEB_level1_WBS_V = obs_noHEB.refs["level1"].product.refs["WBS-V"].product
htp_V = doCleanUp(htp=obs_noHEB_level1_WBS_V, mergeDatasets=True, ignore=False)
cal = mkFreqGrid(htp=htp_V, envelope=False, ignore=False)
htp_V = doFreqGrid(htp=htp_V, grid=cal, ignore=False)
saveProduct(product=htp_V, pool='no_HEB_v3',tag=obsid.__str__())

# H
obs_noHEB_level1_WBS_H = obs_noHEB.refs["level1"].product.refs["WBS-H"].product
htp_H = doCleanUp(htp=obs_noHEB_level1_WBS_H, mergeDatasets=True, ignore=False)
cal = mkFreqGrid(htp=htp_H, envelope=False, ignore=False)
htp_H = doFreqGrid(htp=htp_H, grid=cal, ignore=False)
saveProduct(product=htp_H, pool='no_HEB_v3',tag=obsid.__str__())
```