KØBENHAVNS UNIVERSITET

UNIVERSITY OF COPENHAGEN

MSc THESIS IN IT AND COGNITION

# Spectral Classification of Astronomical Objects using Deep Neural Networks

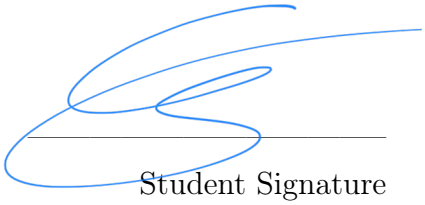
*András Botond Csepreghy*
Department of Computer Science DIKU

Supervised by
**Adriano Agnello**
**Fabian Gieseke**

20 July 2020

# Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Science, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.



Student Signature

20 July 2020

# Acknowledgement

The project this thesis is based on has originated at the 2019 course called Big Data Analysis at the University of Copenhagen that was taught by Troels Christian Petersen and had lectures taught by Adriano Agnello and Brian Vinter. With the supervision of Adriano 3 PhD students, Nikki Arendse, Zoe Ansari, Cecilie Hede and I chose Spectral Analysis as our exam project. I would like to thank Troels and the lecturers for the fantastic course, Nikki, Zoe and Cecilie for their hard work and dedication during the project work. I would also like to thank Fabian Gieseke for his supervision of the thesis. Furthermore I would like to thank Adriano Agnello, who had inspired the project in its very beginning, and has been crucial to its success. Adriano has been an inexhaustible source of knowledge, curiosity and inspiration. I would like to specially thank my girlfriend who did not only provide unconditional support throughout these intense two years, but in fact designed most of the original graphics for this thesis.

To conclude I must express my profound gratitude to my parents without whom I would not have been able to complete my masters degree, and whose love and support were felt at every moment throughout!

# Abbreviations

**AGN** active galactic nuclei

**CNN** convolutional neural network

**ML** machine learning

**PCA** Principle Component Analysis

**QSO** quasar

**RF** Random Forest

**SDSS** Sloan Digital Sky Survey

**SVM** Support Vector Machine

# Contents

# List of Tables

# Abstract

There are large amounts of data collected about astronomical objects by telescopes of wide field surveys, such as the Sloan Digital Sky Survey (SDSS). The way algorithmic classification is done today is by looping through spectral templates and picking one that fits a given spectrum best. In this thesis I explore more efficient methods to classify SDSS spectra, and re-assess the criteria used in the SDSS template classification.

First, a **convolutional neural network (CNN)** is trained exclusively on spectra to learn from visual features that are present in spectra in order to differentiate between object classes. In *mainclass* classification (`galaxy`, `quasar`, `star`), training on **51,200 sources** it reaches a **test accuracy: 0.989**.

Then, I implement a **mixed-input neural network**, which combines the spectral feature extraction from a CNN with other spectro-photometric information into a deep feed-forward network. This combination has proven beneficial, as it introduces quantities (redhisft, equivalent widths, magnitudes) that have a direct physical meaning. Training on 51,200 sources in mainclass classification, it reaches a **test accuracy: 0.993**. This model has been trained on **8 galaxy subclasses** with 39,992 samples (**test accuracy: 0.863**) and **15 star subclasses** with 7,842 samples (**test accuracy: 0.783**).

For future development, I briefly explore autoencoders to reduce dimensionality of spectra to a lower space of parameters that is entirely driven by the data. This small set of numbers can map to physical features of these objects without any prejudice on underlying physical models. Clustering in this low-dimensional space may reveal object classes and physical behaviours that are different from those upon which template matching has been based.

**Keywords -** astronomical spectra, SDSS, CNN, multilayer perceptron, spectral classification, autoencoder, galaxy, quasar, star, machine learning

# 1 Introduction

If we look up to the sky at night we can see just short of 10,000 luminous objects, most of which are stars in our solar neighborhood and a few planets in our solar system. Beyond this shallow layer visible by the human eye, there are billions of other sources in the observable universe with varying physical properties that include *stars*, *galaxies* and some of the most violent objects known to science called *quasars*. The observable universe is the part of space from which light has had time to reach us. Although these objects may seem indistinguishable to us without equipment, they contain a spectrum of colors whose luminosity fluctuates at different wavelengths. Their light encodes a fingerprint of their internal physics, which helps us better understand what exactly we see when looking at the night sky.

We can differentiate between three main classes of objects, namely *stars*, *galaxies* and *quasars*. The identifiable spectral differences between these categories of objects and the differences within classes is the central focus of this thesis.

## 1.1 Stars

Stars are spherical astronomical objects consisting of hot luminous plasma held together by its own gravity. The light comes from thermal radiation that is being generated by heat, which is produced by a fusion reactor in the core of every star. First hydrogen is being fused into helium, then at later stages of the life of a star heavier elements are also fueling this process. Although the light coming from these dense inner layers would result in a continuous spectrum, this is not what we observe. The reason for this is that stars also have atmospheres, thinner layers of gas that are composed of different elements that absorb light at different wavelengths. The combination of the emitted continuum and these absorptions at different wavelengths is what ends up hitting the CCD of a telescope here on Earth.

At first stars were classified by the strength of their hydrogen lines, labeling them from A to P, from strongest to weakest. Later Max Planck showed that all black-bodies emit electromagnetic radiation where the wavelength of light emit-

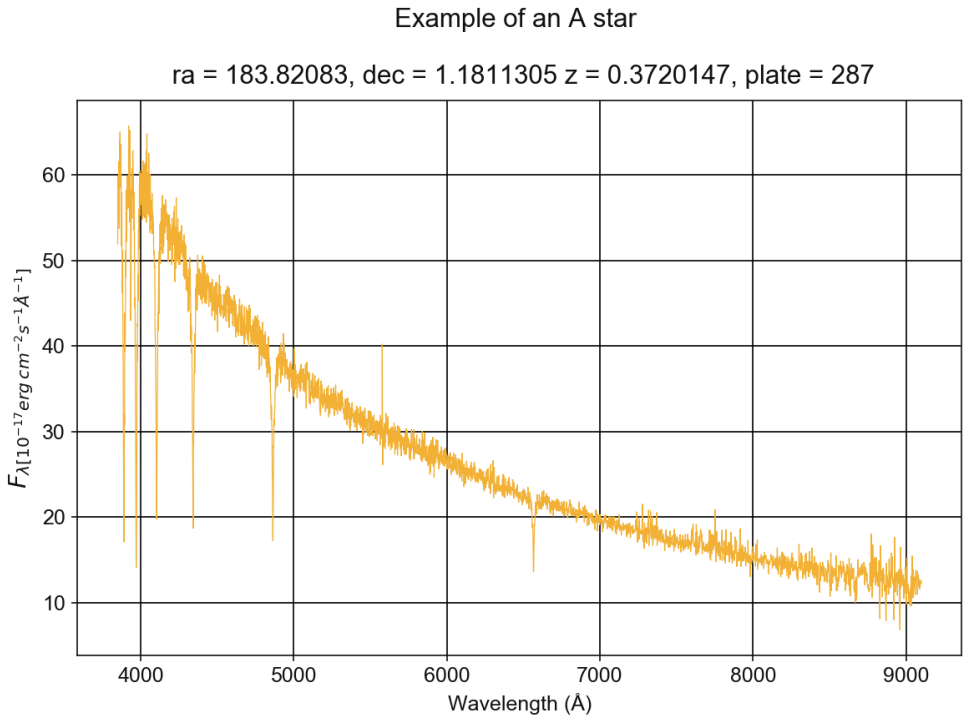



Figure 1: Example of a type A star spectrum.

ted is based on the temperature of the object. Wien's displacement law, which can be derived from Planck's law shows the black-body radiation curve peaks at different wavelengths for different temperatures where the wavelength is inversely proportional to the temperature given by: $\lambda_{peak} = \frac{b}{T}$ where $T$ is the absolute temperature of the object in kelvins and $b$ is Wien's displacement constant, which equals to $2.897... \times 10^{-3} t$ mK.

This means stars with higher temperatures give off light at the bluer end, while cooler ones peak at the reds. This realization formed the basis of a new classification system that had the temperature of the sun as its main component with letters O-B-A-F-G-K-M from hotter to cooler each having further subdivisions symbolized with numbers between 0 and 9. At last roman numerals at the end of the class show the luminosity of a star going rom I to V (1-5).

This system of classification is a rather unfortunate result of the history of star classification caused by the transition from the classification done by using

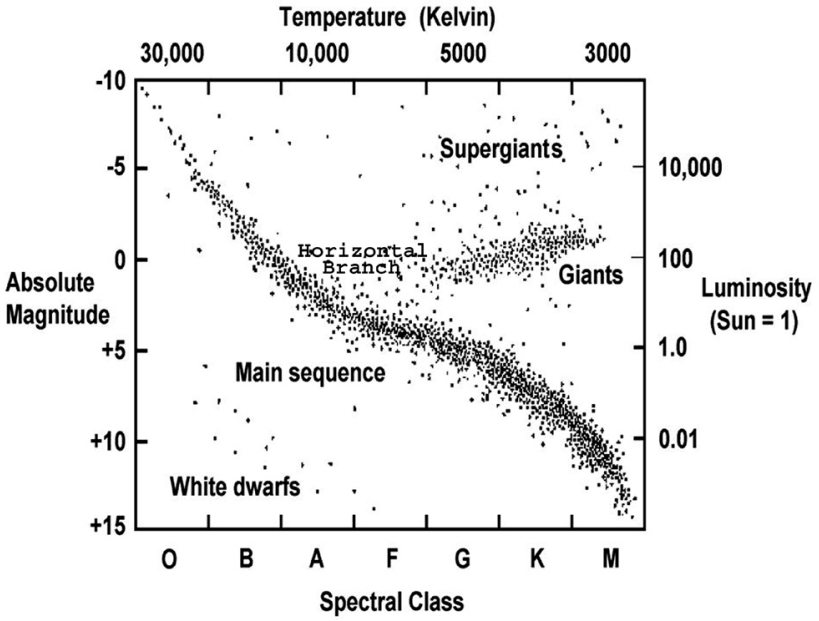



Figure 2: **HR Diagram** image source: CHANDRA

hydrogen lines that used letters A - P to the one we use now, which is based on temperatures. Large catalogs had already been created by the time of the transition, which led Annie Jump Cannon create a new catalog in 1901 that worked with letters already in use reordering and removing some of them based on the star's temperature. Figure 2 shows an HR diagram displaying the temperature-based classification used today.

In the heart of our solar system we have a star of class G2V, which makes our Sun a rather typical star approximately in the middle of the temperature range as well as the mass range.

## 1.2 Galaxies

Before the discoveries of the 20th century people had used galaxy almost synonymously with universe, since we hadn't had proof of the existence of other galaxies until very recently in scientific history. The word "galaxy" comes from the greek "galaxies kuklos" for "milky circle" which described the appearance of the Milky Way on the night sky. We have only recently realized that Milky Way, our own

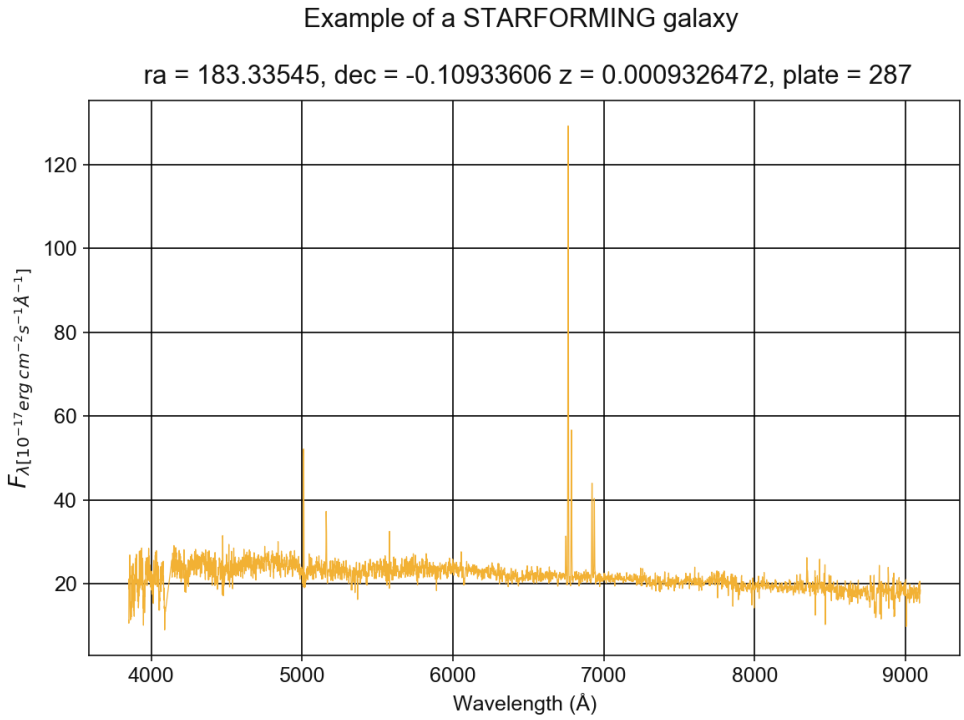



Figure 3: Example of a starforming galaxy spectrum.

galaxy is not the only one in the universe, but only one of hundreds of billions.

Astronomers had previously identified nebulae among the stars, which are interstellar clouds of dust. Some of these were called spiral nebulae from their shape. It was long debated whether spiral nebulae are not just dust clouds but separate star systems outside of our own, until Edwin Hubble in 1925 had located a Cephied variable star in the Andromeda, which we now know is the closest galaxy to the Milky Way approximately 2.54 million light-years away. This discovery had put an end to the debate and proved the Andromeda and many other nebulae are in fact galaxies, which made us realize that the universe is much larger than anyone had thought before. But how do astronomers define what a galaxy is today?

A galaxy is a gravitationally bound collisionless system usually consisting of stars, dark matter, gas and dust. Here a collisionless system refers to a medium in which the interaction cross-section between entities is so low that collisions between entities have no significant effect on the system. There are at least 200

billion galaxies in the observable universe while new research (Conselice et al. 2016) is suggesting this number may even be 2 trillion.

The most common galaxy classification scheme used today by astronomers is the Hubble classification scheme (Figure 4) also known as the Hubble sequence, which is based on galaxies optical morphology, in other words on their visual appearance. Although each class has subclasses we can differentiate galaxies based on these main categories:

- Elliptical
- Lenticular
- Unbarred spiral
- Barred spiral
- Irregular
- Dwarf

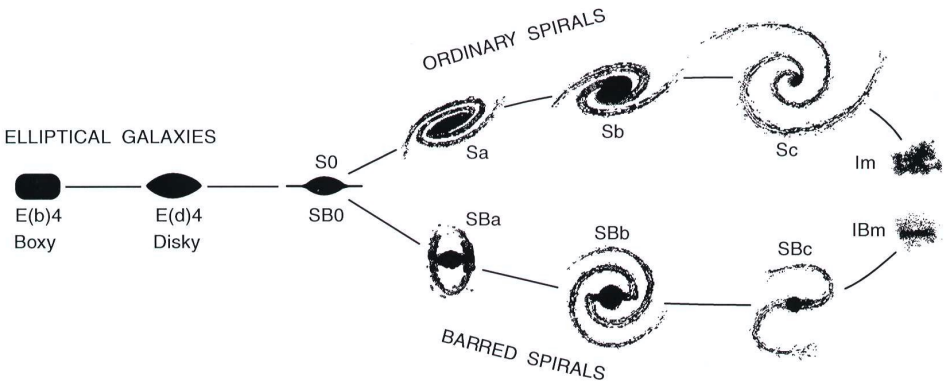



Figure 4: Galaxy classes across the Hubble sequence

Besides using the Hubble sequence, astronomers also separate galaxies according to the current stage of their star formation and the role of their central black hole. SDSS lists 3 subclasses for galaxies according to the following conditions:

| | |
|---|---|
| **Star Forming**: | Set based on whether the galaxy has detectable emission lines that are consistent with star-formation according to the criteria: $\log_{10}(\text{OIII}/\text{H}\alpha) < 0.7 - 1.2(\log_{10}(\text{NII}/\text{H}\alpha) + 0.4)$ |
| **Starburst:** | Set if the galaxy is star-forming but has an equivalent width of $H\alpha$ greater than 50Å |
| **AGN** | Set based on whether the galaxy has detectable emission lines that are consistent with being a Seyfert or LINER: $\log_{10}(\text{OIII}/\text{H}\alpha) > 0.7 - 1.2(\log_{10}(\text{NII}/\text{H}\alpha) + 0.4)$ |

This project will be focusing on these 3 subclasses of galaxies, since SDSS provides only these classes. While algorithmic spectral classification has been done along the Hubble sequence (Zaritsky, Zabludoff and Willick 1995), it is outside the scope of this project.

## 1.3 Quasars

Nearly all galaxies have a supermassive blackhole at their center. The ones that have surrounding dust and gas falling into them, thereby producing light around in the central areas, are known as active galactic nuclei (AGN). A *quasar (QSO)* or *quasi-stellar object* is an extremely luminous AGN whose blackhole can reach masses up to billions of solar masses. The gas and dust falling into the blackhole can reach extreme speeds and heat, producing light that outshines its host galaxy. Quasars are among the brightest and most violent objects known in the universe.

Quasars weren't discovered until the 1950s when astronomers had noticed radio sources that had no corresponding measurable light emission in the optical spectrum in all-sky radio surveys (Shields 1999, Matthews and Sandage 1963). Hundreds of sources like these were collected by 1960 without knowing their origins until 1963 when Allan Sandage and Thomas A. Matthews published a paper about three radio sources that were found to have a corresponding optical sources

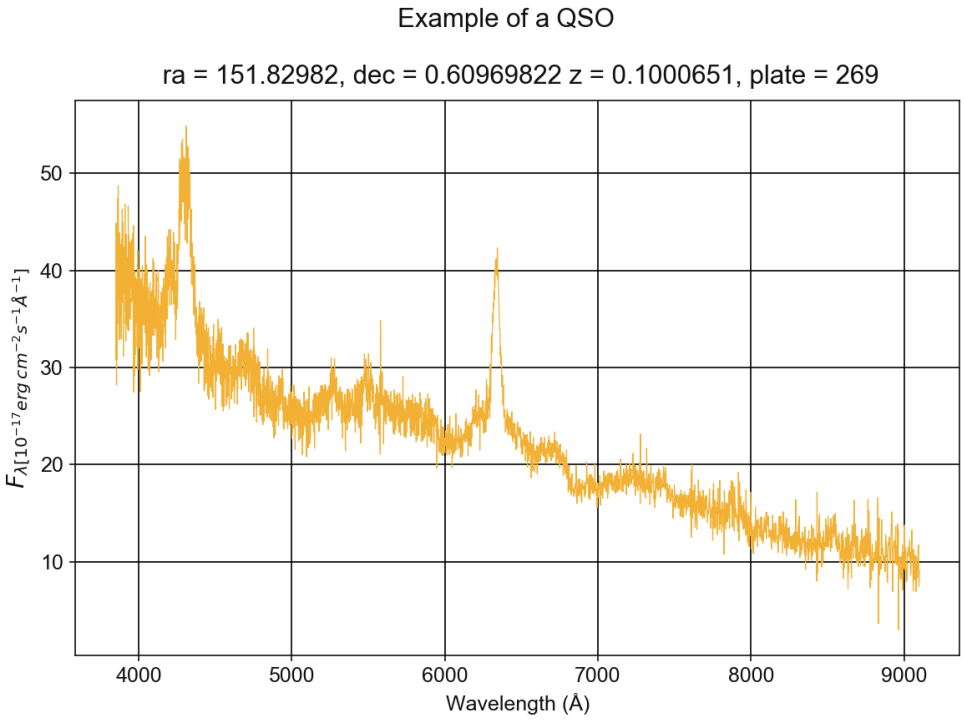



Figure 5: Example of a quasar spectrum.

of point-like objects that showed unexplained broad emission lines in their spectra. One of these sources, 3C 48 had a large redshift of z = 0.3675 (Matthews and Sandage 1963), however its luminosities were variable with a time scale that ranged from months to hours, which means it cannot be larger than a few light-weeks across. For comparison, our solar system has a diameter of approximately 287.46 billion kilometers, which equates to 11.098 light-days. (Throughout the thesis I will be using *redshift* and $z$ interchangably)

We now know that this radiation is coming from AGNs that are fueled by matter accreting onto a supermassive black hole with masses ranging up to several billion solar masses billions of light years from Earth. A typical quasar spectrum is characterized by a relatively flat continuum and broad emission lines, seen in Figure 5, a plot of a quasar downloaded from SDSS.

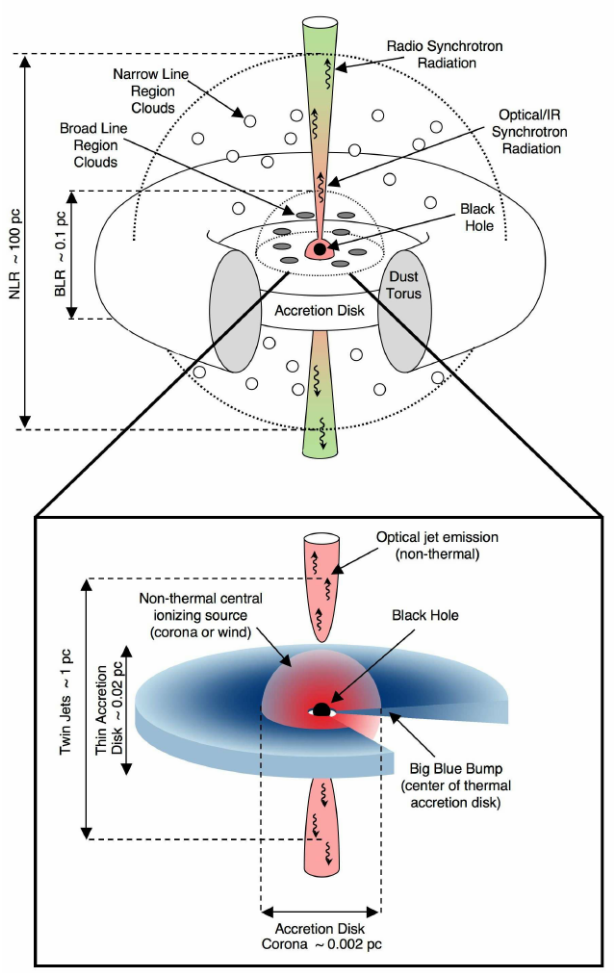



Figure 6: Schematic diagram of the structure of a quasar on scales from 0.002 100 pc. Image source: Unwin et al. 2006

With our modern telescopes we capture more and more of these spectra and store them in large data collections called surveys such as the Sloan Digital Sky Survey (SDSS). Because of this large rate of data collection it is becoming increasingly labor-intensive to manually label and analyze spectra. This project aims to address this issue by automating some of these processes using machine learning (ML) algorithms to classify and extract interesting features from stellar objects.

# 2 Spectroscopy

Light, or electromagnetic radiation carries a lot of information about the source that produced it. Spectroscopy is the study of the characteristics of this light, more precisely, spectroscopy is the study of emission and absorption of radiative energy as a function of frequency or wavelength predominantly in the electromagnetic spectrum. There are many different types of spectroscopy used in different industries that are working with a range of different radiations, not exclusively visible light. This project focuses exclusively on optical spectroscopy.

## 2.1 Astronomical Spectroscopy

Spectroscopy not just in astronomy but of all kind can be traced back all the way to Isaac Newton (1643-1727) when he discovered in 1666 that a prism splits white light and reflects different colors by different angles. From this he concluded that color is an intrinsic property of light, and that white light is composed of a spectrum of colors. Joseph von Fraunhofer (1787-1826) extended Newton's work in optics by making his own lenses. He was to able to create a large enough high quality prism, which when paired with a telescope he could see the spectrum of the Sun so clearly that he identified 574 fixed dark lines. In honor of Joseph von Fraunhofer these absorption lines are still called Fraunhofer lines. The explanation of Fraunhofer lines was unknown until the work of Gustav Kirchhoff (1824-1887) and Robert Bunsen (1811-1899), who suggested that they were caused by selective absorption of a continuous spectrum produced by the hot interior of the Sun by cooler gases at the surface. To this day about 25,000 absorption lines are known to exist in the solar spectrum between wavelengths 2,950 and 10,000 Ångströms (Å) where $1 = 10^{-10}m$

In modern times three types of astronomical spectroscopy can be differentiated based on the wavelength range being measured by the instrument. Radio spectroscopy measures light with wavelengths longer than infrared light called radio waves. Telescopes operating in this spectrum of wavelengths require large antennas to capture an adequate amount of information. X-ray and gamma-ray spectroscopy deals with light at high energy and low wavelengths, at the oppos-

ite end of the spectrum. Since most of the high-energy electromagnetic radiation is absorbed by the atmosphere, instruments have to be used at higher altitudes, which is usually achieved by balloons or satellites, however observatories built on Mauna Kea at 4,205 m above sea level are also capable of detecting x-rays.

In between these ranges lies the spectrum that optical spectroscopy is operating with which approximately covers the spectrum of light that is visible to the human eye. The main focus of this project is analyzing data from telescopes that are using optical fiber spectroscopy.

## 2.2 Sloan Digital Sky Survey

SDSS is a large multi-spectral imaging and spectroscopic redshift survey that has created the most detailed three-dimensional maps of the Universe. It uses spectroscopy to and this thesis will focus on analyzing optical spectra downloaded from SDSS.

The main telescope that SDSS uses is the Sloan Foundation 2.5m Telescope at Apache Point Observatory located at the Apache Point Observatory, in south east New Mexico. According to a paper on its construction by Gunn et al. 2006: "The telescope is instrumented by a wide-area, multiband CCD camera and a pair of fiber-fed double spectrographs.". In order to capture spectra they drill holes into metal plates and insert optical fibers into the narrow slits for the spectrographs.

Gunn et al. 2006 describes the construction and design of the telescope in much more detail, which is outside the scope of this thesis. I will be focusing exclusively on the structure of the data it records.

# 3 Algorithmic Spectral Classification in the Literature

Before diving into the details of machine learning and how it is applied in this context, I would like to mention a few examples of spectral analysis in the literature. There have been several efforts in doing spectral classification that are for the most part are focusing on subclass classification of stars and some that are

focused of identifying ratios of the emission line to absorption line contribution of galaxies or quasars.

In a paper by Elting, Bailer-Jones and Smith 2008, they trained a Support Vector Machine (SVM) on u-g, g-r, r-i colours to classify galaxies, quasars and stars. Using only the magnitudes and an SVM they were able to obtain relatively good results differentiating between these main classes.

Others such as S. Sharma and R. Sharma 2019 have experimented with many different ML algorithms for spectral classification including SVMs, Logistic Regression, Random Forest (RF) and Decision Tree classifiers to find which offers the best performance in classifying stars, galaxies and quasars.

However most of the literature has focused on binary classification using two classes e.g. stars and quasars (Zhang et al. 2012, Viquar et al. 2018) or subclass classification of a specific main class like stars (Bailer-Jones 2001) or galaxies (Zaritsky, Zabludoff and Willick 1995). Table 1 shows a list of classification efforts of various kinds using different, not exclusively ML methods.

| Data Source | Classes | Algorithms Used | Reference |
|---|---|---|---|
| SDSS | star, galaxy, quasar | Logistic regression, SVM, RF, decision trees | **(1)** |
| CRTS | identification of supernovae separation between galactic and extragalactic objects | Random Forest. MLP with Quasi Newton Algorithm, KNN | **(2)** |
| SDSS | star, quasar | SVM | **(3)** |
| SDSS | star, galaxy, quasar | SVM | **(4)** |
| SDSS | star, quasar | Asymmetric AdaBoost | **(5)** |
| 24 samples | Hubble sequence | Principle Component Analysis (PCA) | **(6)** |
| SDSS | galaxy subclasses | K-means, single-pass K-means | **(7)** |
| LAMOST | CVs and non-CVs | restricted Boltzmann machine | **(8)** |
| LAMOST | star subclasses, Defective spectra recovery | Autoencoders, deep ANN | **(9)** |
| Long-slit spectroscopic survey | Hubble sequence | Chi2 fitting of templates ratios of emission line to absorption line | **(10)** |
| Multiple | star subclasses | Gaussian probabilistic models, ANN, PCA | **(11)** |
| LAMOST, SDSS | star, galaxy, quasar | RF | **(12)** |

Table 1: **Algorithmic spectral classification** in the literature. References: **(1)** S. Sharma and R. Sharma 2019, **(2)** D'Isanto et al. 2016, **(3)** Zhang et al. 2012, **(4)** Elting, Bailer-Jones and Smith 2008, **(5)** Viquar et al. 2018, **(6)** Sodré and Cuevas 1994, **(7)** Ordovás-Pascual and Almeida 2014, **(8)** Chen 2013, **(9)** Wang, Guo and Luo 2016, **(10)** Zaritsky, Zabludoff and Willick 1995, **(11)** Bailer-Jones 2001, (**12**) Bai et al. 2018

# 4 Data Mining and Preprocessing

Data preprocessing is a crucial factor in improving the performance of machine learning algorithms. In order for a ML algorithm to be able to find existing patterns in feature space by minimizing a loss function the data needs to be prepared adequately.

Data preprocessing can sometimes feel like doing art, since many decisions are left for the judgement of the developer or researcher where there is no one right answer for all circumstances. It is important to ask questions such as:

- How to handle missing values?
- Does the data need normalization, standardization? If so, does all of it need to be normalized or just some parts?
- Should the dimensionality of the data be reduced? If so with what technique and to what degree?
- How to handle categorical features?
- What ML algorithm will be used? Does the data need specific changes for it?
- Are all features representative of the data?
- Are there different types of data that need to be handled differently? Images, text, categories, sound, dates, etc.

Although this list is by no means exhaustive, addressing these issues already helps one to optimize the data to fit a specific ML algorithm. This project has incorporated several data preprocessing methods at different stages of the pipeline.

## 4.1 Mining SDSS

The first step of the algorithmic spectral classification is to obtain the spectra to be classified. There are different ways of querying and downloading spectra from SDSS, however to stay consistent and make the whole process reproducible I

incorporated the querying as well as the downloading into a Python project that also houses all the machine learning code (see Appendix 10.1). For this I used `astroquery` (astroquery), a set of tools for querying astronomical web forms and databases. It has an `SDSS` class, that can be used to implement `SQL` queries to get specified columns.

When querying all sources without filtering on type it returns 83% galaxies. To compensate for this, after the initial downloads I added QSO-only and STAR-only queries to balance the dataset. After the correction the total number of sources adds up to 96,115 out of which there are 53,825 (56.0%) galaxies, 32,488 (33.8%) quasars and 9802 (10.2%) stars. An example query for only fetching quasars used for this project looks like the following:

```
query = "SELECT spec.z, spec.ra, spec.dec, spec.specObjID, \
                spec.bestObjID, spec.fluxObjID, spec.targetObjID, \
                spec.plate, spec.class, spec.subClass, spec.zErr, \
                spho.petroMag_u, spho.petroMag_g, spho.petroMag_r, \
                spho.petroMag_i, spho.petroMag_z, spho.petroMagErr_u, \
                spho.petroMagErr_g, spho.petroMagErr_r, \
                spho.petroMagErr_i, spho.petroMagErr_z \
         FROM SpecObjAll AS spec \
         JOIN SpecPhotoAll AS spho ON spec.specObjID = spho.specObjID \
         WHERE spec.zWarning = 0 AND spec.class = 'QSO'"

res = SDSS.query_sql(query, timeout=600)
df = res.to_pandas()
```

A version of this query was used in the Big Data Analysis course project and since then I have added improvements and expanded the columns queried. The current version fetches and merges columns for *SpecObjAll* and *SpecPhotoAll* tables. The full code and a link to the GitHub file for querying can be found in Appendix 10.2.

The table that results from querying is then saved to a CSV file to be later used for downloading spectral data, which was in fact the tricky part. It appears SDSS limits the number of sources queried at once to 500,000, which is plenty to work

with since downloading fluxes for sources takes a long time. We have found no convenient way of bulk downloading multiple GBs of spectra at once from SDSS. Instead we used astropy and astroquery to loop through all the coordinates in the CSV file and query a spectrum for each coordinate. A slightly simplified version of the downloading looks like the following:

```
from astropy.table import Table
from astropy import coordinates as coords
import astropy.units as u
from astroquery.sdss import SDSS

for i in range(n_coordinates):
    pos = coords.SkyCoord((ra[i]) * u.deg, (dec[i]) * u.deg, frame='icrs')
    xid = SDSS.query_region(pos, spectro=True)
    xid = Table(xid[0])

    sp = SDSS.get_spectra(matches=xid)

    df['fluxes'].append(sp[0][1].data['flux'])
    df['wavelength'].append(10. ** sp[0][1].data['loglam'])
    df['z'].append(xid['z'])
    df['ra'].append(xid['ra'])
    df['dec'].append(xid['dec'])
    df['objid'].append(xid['objid'])
```

The full code and a link to GitHub for downloading can be found at Appendix 10.3. Downloading was executed for 10,000 coordinates at a time, which resulted on average approximately 5000 unique spectra running for about 8 hours. The high drop-out rate is due to various factors such as `xid` being undefined, position returned by coords.SkyCoord being null and a significant number of duplicates being downloaded or sources that have an insufficient number of fluxes that are later removed.

The downloaded fluxes and wavelengths are appended to `pandas.DataFrame` (pandas) which is later saved and read from a `pickle` (pickle) or `parquet` (parquet) file. The project began by initially using `pickle` files for all types of storage both the temporary files that exist in between stages as well as the permanent files that are stored to be used for ML. Later the temporary storage switched to

using parquet and the permanent storage to use `HDF5` both of which offer several performance benefits compared to pickles, but further discussion on performance differences is outside the scope of this thesis.

For convenience the whole list of fluxes and wavelengths are stored their own separate cells to have all data in one place when saving right after downloading. This hurts read-write performance and file size but is later changed when merging data.

## 4.2 Data Preprocessing for Spectral Classification

This project has two stages of data preprocessing. One that organizes the data in a form that can be stored and accessed on demand for classification or making plots and is human-readable. The other stage happens right before classification and serves the purpose of shaping the data so that it can be fed to the neural network. This is slightly different for each architecture used throughout the project and it will be elaborated on in the chapter that describes these neural network architectures. This section is focused on describing the pipeline of data processing and storing.

I developed a specific pipeline tailored to this dataset whose steps have to be executed in order since each step results in a file that will be used by the next although some of the steps can be used in different order. The reason for not having written a single set of functions that would produce the end result is because some of the computation takes a long time to run, while if there is an error present in the later stages it would halt the whole process that would require a rerun of the whole pipeline.

**Step 1: Filtering out sources.** Around 1% of the downloaded spectra have an insufficient number of fluxes. This means they're missing from a few hundred up to in some cases over 50% of their fluxes. Due to time restriction and since these sources are not vast in number they are simply discarded. A future project could be to use interpolation or a neural network to fill in missing values. Data could be generated in which some sections of flux values are removed. Then a neural network

would learn to estimate the missing values by minimizing the mean squared error between its reproduction and the original data. This network could be used to approximate missing values, however its effectiveness should be established first by setting an acceptable threshold of error.

**Step 2: Cutting off values at the edges.** Most flux values fall between 3,850.34 Å and 9,099.13 Å. There are a few data points that are outside of this range, which may still be physically valid or were caused by measurement error. These fluxes are removed for sake of simplicity and due to the fact that machine learning algorithms requires the same number of input values for each sample.

**Step 3: Compute equivalent widths.** The functions used for computing equivalent widths were developed by Nikki Arendse during the Big Data Analysis course. To calculate the equivalent width ($EW$):

$$EW_w = \sum_{i=1}^{N} \frac{F_w^c(\lambda_i) - F_w(\lambda_i)}{F_w^c(\lambda_i)} \Delta\lambda \tag{1}$$

where $F_w^c(\lambda_i)$ is the relative flux of the continuum level, $F_w(\lambda_i)$ is the relative flux of the spectral line and $\Delta\lambda$ is the sampling in wavelength unit (Å).

# 5 Machine Learning

A Machine Learning algorithm is one that is able to learn patterns in data by observing examples. A formal definition of this from Mitchell et al. 1997a is "A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$." Machine Learning enables developers to find solutions to problems that are too complex to be solved by fixed programs written and designed by humans.

## 5.1 Brief History of Deep Learning

The history of deep learning can be traced back to a celebrated paper, McCulloch and Pitts 1943. In this paper they laid out a computer model that was inspired by the neural network of the human brain. Their initial work was followed by excitement by researchers such as A.L.Hodgkin, A.F.Huxley and John McCarthy as well as Alan Turing and whose groundbreaking work laid down the foundations of "artificial intelligence" a term coined by John McCarthy.

Early excitement and false promises led to two AI winters in the 1970s and 1990s, both fueled by high expectations that were impossible to meet with the technology of the time. Since 2012 machine learning and neural networks are blossoming again fulfilling promises that research of past decades could not. One may hope that this time all AI winters are behind us.

There are three main branches of machine learning namely supervised learning, unsupervised learning and reinforcement learning, but the boundaries between them can be blurry as they often share the use of same algorithms. This thesis will be mainly focusing on supervised learning, so what is supervised learning?

## 5.2 Supervised Learning

At its core a supervised learning problem can be written as

$$Y = f^*(X) \tag{2}$$

where $X$ is the input feature vector $X = (x_1, x_2, ..., x_k)$ where $x_i$ is a single feature, $Y$ is the output vector of labels $Y = (y_1, y_2, ..., y_k)$ that we want to predict and $f^*$ is the function that maps input $X$ to output $y$. The main objective of supervised learning is to learn a function that best approximates $f^*$ and accurately predicts $Y$ with input values that the learning algorithm has not seen. In other words we can define the mapping as

$$Y = f(X; \theta) \tag{3}$$

where we learn the parameters $\theta$ that best approximate $f^*$. The learning of these parameters is called training. But how can we define what "best" is and how can we measure how far we are from it?

The measure of the performance of a ML algorithm is defined by the *loss function* or *cost function*. In its general form a loss function $L$ is defined by $L(y_i, \hat{y_i})$ where $\hat{y_i} = f(x_i; \theta)$, what our model predicts from input $x_i$. $L(y_i, \hat{y_i}) = 0$ when the prediction is perfect. When training a model we are trying to minimize $L(y_i, \hat{y_i})$. The choice of $L$ depends on the type of output variable we are trying to predict and on the dataset we are working with. The value of $y$ can just be a rational number in case of regression problems (e.g. predicting the price of a house from features like location, number of rooms, $m^2$ etc.) or a categorical value often represented in a one-hot encoded vector. An example of a one-hot vector would be the following

| | AGN | BROADLINE | STARBURST | STARFORMING |
|---|---|---|---|---|
| $y_1$ | 1 | 0 | 0 | 0 |
| $y_2$ | 0 | 1 | 0 | 0 |
| $y_3$ | 0 | 0 | 1 | 0 |
| $y_4$ | 0 | 0 | 0 | 1 |

Table 2: One-hot encoded representation of galaxy classes

where $y_1^\top = (1, 0, 0, 0)$ is an AGN. This is the categorical representation of labels I will be using in this thesis. The loss functions used in this project are explained in Section 5.5 because their use are dependent on the ML algorithm being used.

## 5.3 Perceptron

To understand how neural networks work, we must start with the *perceptron*, its core building block also called a node or neuron. A perceptron is a binary classifier that takes a vector of real-valued inputs and computes a linear combination of these inputs. If the result is above a certain threshold it outputs 1, and -1 otherwise. Formally, given $k$ input variables $x_1, x_2, ..., x_k$, the output $\hat{y}$ is computed by the perceptron is

$$\hat{y} = \begin{cases} 1 & \text{if } b + w_1 x_1 + w_2 x_2 + ... + w_k x_k > 0 \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

where each $w_0, w_1, ...w_k$ and $b$ are real-valued constants. In this example $w = w_1, w_2, ..., w_k$ is a vector of weights or connections whose elements represent how much a certain input $x_i$ contributes to the output, and $b$ is the bias or threshold. We can rewrite the above equation as

$$\hat{y} = \begin{cases} 1 & \text{if } W \cdot X + b > 0 \\ -1 & \text{if } W \cdot X + b < 0 \end{cases} \tag{5}$$

Written as an equation we can define an $n - 1$ dimensional hyperplane

$$w \cdot X + b = 0 \tag{6}$$

where $n$ is the number of elements that vector $X$ has. If the data set is linearly separable the perceptron algorithm returns values for $w_1, w_2, ..., w_k$ and $b$ such that all data points on one side of the hyperplane are of one class and data points on the other side are of the other. So far we used the sign function as our *activation function*, however we may choose from a number of different activation functions, which determines the effectiveness of learning. The activation function of a node

defines the output of a node given an input or set of inputs. A popular one is the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ where we define $z$ as $w \cdot X + b$. As activation function for the neural networks in this project I will be using the rectified linear unit (ReLU): $ReLU(x) = \max(0, x)$ where $x$ is the input of a node. It has many advantages over the sigmoid function, including better gradient propagation, efficient computation and being scale-invariant.

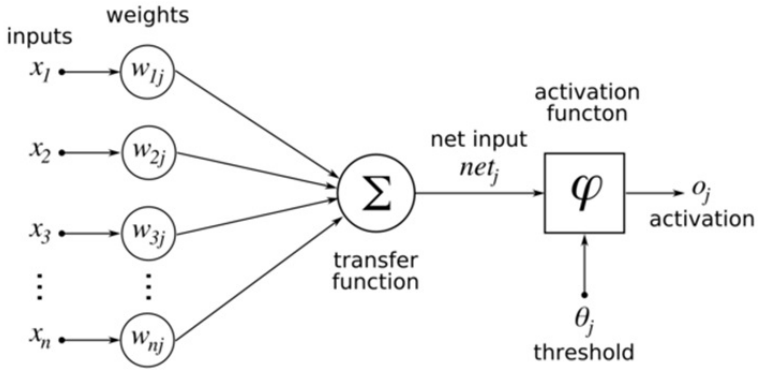



Figure 7: Perceptron
image source: Mitchell et al. 1997b

The process by which the perceptron learns the mapping between input $x$ and desired output $y$ is called the perceptron learning rule. This learning rule is an example of supervised training, which is provided with a set of examples of inputs and desired outputs $\{x_1, y_1\}, \{x_2, y_2\}, ..., \{x_n, y_n\}$. As each input is provided to the perceptron, it is compared to the desired output, the learning rule then adjusts the weights in the weight vector $w$ and the biases of $b$ in order to influence the output of the perceptron to be closer to the desired output. There are many different learning rules or learning algorithms for different types of machine learning models. The perceptron learning rule is interesting for historical reasons but throughout the project I will be using a different algorithm mentioned in section 5.5 and 6, so the he exact details of the perceptron learning rule is outside the scope of this project.

## 5.4 Universal Approximation Theorem

The universal approximation theorem states that a feed forward neural network with one hidden layer can approximate any arbitrary real-valued continuous function $f(x)$ arbitrarily well given a finite number of perceptrons. Although going through the formal definition of this theorem is outside the scope of this project I thought it is worth mentioning since it emphasizes a feature of neural networks that is incredibly important to the field of computer science. The ability to compute an approximation of an arbitrary function is truly remarkable and is the reason why neural networks can generalize so well.

## 5.5 Artificial Neural Networks

A feed forward deep neural network is a fully connected multi-layer perceptron that has any number of layers between the input and output layer called hidden layers. Each neuron of each of its hidden layers is connected to each neuron in the next one. Deep feed forward neural networks are of extreme importance to machine learning experts. They form the basis for many applications and many specialized networks have been built that are all rooted in feed forward networks. An example

would be a convolutional neural network often used for image classification, which I will be using for spectral analysis.

The goal of a feed forward neural network is to approximate some function $f^*(x)$ which maps an input $x$ to an output $y$. It is called feed forward, because information flows from input to output through intermediate computations without feedback connections. Networks that incorporate feedback connections are called recurrent neural networks. A network that has three layers can be written as $f(x) = (f^{(3)}(f^{(2)}(f^{(1)}(x)))$ where $f(x)$ is the whole network, $f^{(1)}$ is the first layer, $f^{(2)}$ is the second layer and so on. The final layer of a network is called the **output layer** and the number of neurons it contains matches the number of classes to be predicted. For regression tasks it often has one final neuron that outputs a scalar prediction. The number of layers in a neural network gives its depth. The name "deep learning" comes from neural networks that have many hidden layers, hence considered deep.

For the sake of simplicity, let's consider a single neuron or node

$$a_0^{(1)} = ReLU\left( w_{0,0}a_0^{(0)} + w_{0,1}a_1^{(0)} + \ \cdots \ + w_{0,n}a_n^{(0)} + b_0 \right) \tag{7}$$

where $a_0^1$ is the 0th neuron from layer 1, $w_0, n$ is the weight connecting $a_n^0$ to $a_0^{(1)}$, $b_0$ is the bias of $a_0^{(1)}$ and $ReLU$ is its activation function, which we have seen in the previous chapter $ReLU(x) = \max(0, x)$. To incorporate all neurons of a layer we can rewrite the above in a matrix form

$$a^{(1)} = ReLU\left( \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{2,2} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right) \tag{8}$$

where the activation function $ReLU$ applies to each neuron separately

$$ReLU\left( \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \right) = \begin{bmatrix} ReLU(a_1) \\ ReLU(a_2) \\ \vdots \\ ReLU(a_n) \end{bmatrix} \tag{9}$$

Condensing equation 8 we can say

$$a^{(1)} = ReLU\,(\,\tilde{W}a^{(0)} + b\,) \tag{10}$$

Going back to 3 layers we can say

$$f(\mathbf{x}) = ReLU\,(\,\tilde{W}^{(3)}\,ReLU\,(\,\tilde{W}^{(2)}ReLU\,(\,\tilde{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\,) + \mathbf{b}^{(2)}\,) + \mathbf{b}^{(3)}\,) \tag{11}$$

where the first layer takes input vector $x$ and weight matrix $W^{(1)}$, computes their dot product, adds first layer's bias vector $b_1$ and which will be the input of the activation function $ReLU$ whose output will be the input of the second layer and so on.

Learning is the process of finding the right values of this weight matrices $W$ and bias vectors $b$ so that it minimizes the loss function chosen for the given problem.

The loss function used throughout this thesis for classification is called categorical crossentropy. It is a popular loss function for ML problems where there is only correct prediction for each example, the classes are mutually exclusive. Categorical cross-entropy loss function is defined as

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}\,(\,y_{ij}\,\log(\,\hat{y}_{ij}\,)\,) \tag{12}$$

where $y$ is the one-hot vector representation of the true class and $\hat{y}$ is the predicted value. This loss function compares the distribution of the output of the model (output of last activation function) with the true distribution of the one-hot encoded true class. In other words the closer the two vectors are to each other the lower the loss will be.

Training is done by minimizing the loss function by tuning the parameters of the model with an optimization algorithm. My choice of algorithm is called **Adam** (Kingma and Ba 2014)

Adam is an adaptive learning rate optimization algorithm proposed in 2014 specifically designed for training of deep neural networks. It is an extension of stochastic gradient descent (SGD) with momentum, the main difference being that while SGD uses a uniform learning rate for all parameters (weights and biases), Adam separately finds individual learning rates for each parameter during the optimization. It also takes advantage of Adagrad (Duchi, Hazan and Singer 2011)

that performs well with sparse gradients but struggles in non-convex optimization, and RMSprop (Ruder 2016) which solves some of the problems Adagrad introduces. Detailed discussion of the mathematics behind the Adam optimizer algorithm is beyond the scope of this thesis.

# 6 CNN with hyperparameter optimization

To classify spectra there are two main ideas presented in this thesis. One is to use all available information about the source beyond the flux values of the spectrum. This includes redshift, Petrosian magnitudes, equivalent widths and plate. In the other approach I tried focusing only on the fluxes, using a 1 dimensional convolutional neural network that extracts successively larger visual features in a hierarchical set of layers. The intuition behind which is that since an astronomer is able to recognize visual patterns by only looking at the spectrum, a CNN in principle should be able to do the same. To maximize performance I used Keras Tuner (keras-tuner) for hyperparameter optimization or hypertuning. 40 trials were run 16 of which are presented in Table 3 and 4. The CNN with the best performing hyperparameter combination were trained for 60 epochs classifying the 3 main classes. The complete dataset contained **96,115 spectra** consisting of **53,825 galaxies** (56.0%), **32,488 quasars** (33.8%) and **9,802 stars** (10.2%). I randomly sampled 64,000 spectra that were used for training with the result: **training accuracy: 0.999** and **test accuracy: 0.989**. Note: the dataset described here was used later in the thesis with the same distribution of sources.

## 6.1 CNN, from convolution to classification

A convolutional neural network is a class of deep neural networks that are often used in image classification that make use of mathematical operation called **convolution**. CNNs learn **filters** that detect visual features in a given dataset at different scales by convolving these filters with the input matrix (vector in the case of 1D CNN). Their main advantage is that they are shift or space invariant, which means they can learn features regardless their spatial size or position. CNNs are applied on datasets whose elements have learnable spatial or temporal patterns, such as images for the 2 dimensional case, but also sound waves and time series for the 1 dimensional case or spectra in this project. After layers of feature extractors (filters) there are some regular fully connected layers, or dense layers that use the the features from the previous convolutional layers to classify the feature representations. The number of these layers and the number of neurons in each

layer depend on the complexity of the problem.

A convolution is the integral that expresses the overlap between function $g$ and $f$ as $g$ is shifted over $f$. In the CNN implementation of convolution we have a small local function $g$, to which I will refer as filter or kernel. The convolution of $g$ and $f$ is given by

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{13}$$

In its discrete form we can rewrite the above as

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{14}$$

So, how does this convolution relate to neural networks?

Very similar to a layer described in feed forward neural networks a convolutional layer or conv layer has computational units whose parameters are tuned in the learning process. Instead of matrix multiplication of the weight matrix and input vector a convolutional layer consists of filter tensors, whose dimensionality matches the input of the data (most often 2D matrix for images and 1D vector for time series).

Let's consider a one dimensional case, since this is the one we will be working with throughout this project. Each filter of a convolutional layer convolves across the input vector (vector of fluxes in the case of the first layer) and outputs a set of values that undergo **max pooling**. Max pooling is a type of pooling layer that reduces the dimensionality of the output of the convolution by a factor that is a tunable parameter. This reduction happens over sections of values determined by the size of the max pooling layer by picking the maximum value in each given section. If we take a max pooling layer of size 3, it would mean we slice the input vector into vectors of length 3, pick the maximum from each of these groups and return it. For a visual representation of this process see Figure 9. Pooling is important to reduce the complexity of filters and to capture larger and larger features. The output of pooling will be the input of the next convolutional layer and so on.

After the last convolutional layer outputs the result of its computation, its output will be the input of a dense layer that may be followed by any number of others. To see the architecture used in this project, see Figure 10. The hyperparameters in a convolutional layer is the number of filters, their sizes (uniform size for each layer) and the size of the max pooling. The parameters that are learned during training are the values of the filter vectors that recognize features that help the dense layers to classify based on these visual features.

This architecture works well because now the fully connected layers are provided learned features that are shift invariant instead of the raw values. Note that regular feed forward networks are capable of classifying images if the subject is in the center and is the same size across all images, although less accurately than CNNs especially regarding images with higher resolution.

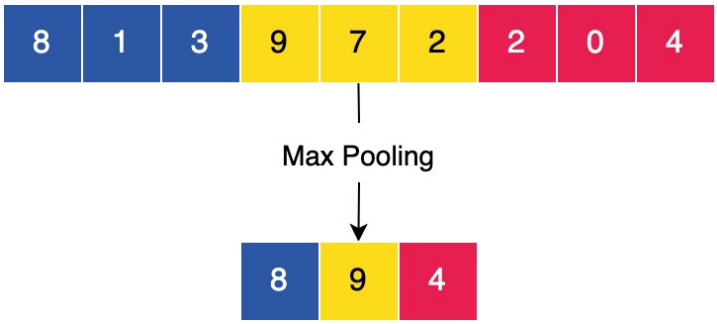



Figure 8: An example of a max pooling computation of size 3 which reduces 9 values to 3

## 6.2 Data Preprocessing

Before diving into the specific architecture of this solution I would like to spend some time on the last layer of data preprocessing that takes place before training the model. Since the CNN is only provided fluxes, the preprocessing done at this stage is realtively simple, since most of the preperation has been executed prior.

The dataset can be defined as

$$F = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{k1} & f_{k2} & \cdots & f_{kn} \end{bmatrix} \tag{15}$$

where $F$ is the matrix of fluxes from all sources, $f$ is a single flux, $k$ is the number of sources and $n$ is the number of fluxes per source.

In ML it is generally accepted practice to standardize or normalize the dataset before training. Normalization scales all datapoints to a range of $[0, 1]$ and standardization scales the data to have $\mu = 0$ and $\sigma = 1$ and is calculated by

$$z = \frac{x - \mu}{\sigma} \tag{16}$$

Standardization is crucial especially when dealing with data points of different scales. The reason for its use here is that standardization makes computing the gradient of the loss function more efficient and some fluxes have large emission lines or some artifacts cause values to be off scale.

Furthermore sources are randomly shuffled and split into 80& training and 20% test sets of which training is further subdivided into 80& training and 20% validation shown in Figure 10. Throughout the project all splitting of the dataset for training, validation and test was done using these ratios.

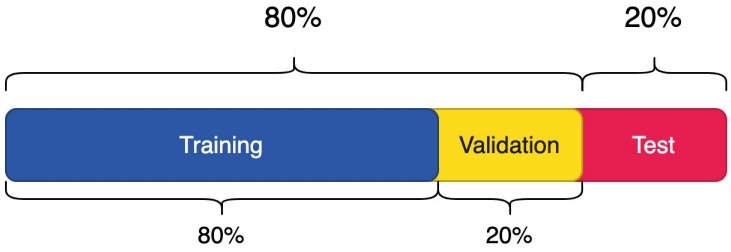



Figure 9: Train test validation split used throughout this project

Lastly the target labels were transformed into one-hot encoded vectors detailed in section 5.2.

## 6.3 Architecture

The general architecture of the CNN used in this project is relatively simple, since we are only using the fluxes and inputs to classify a source. In order to maximize its performance I implemented automated hyperparameter optimization. This means defining a search space which incorporates all tunable hyperparameters whose values will be dynamically change depending on what combination gives the best performance on the validation set while training. To optimize the network architecture to maximize its performance I used keras-tuner, which is a package offered by keras for hyperparameter optimization. Unfortunately Keras only mentions "state of the art hypertuner algorithms" to explain how the hypertuning works, so I will have to trust these state of the art algorithms. The complete network architecture including the tunable parameters can be seen in Figure 11. Using the Keras API building the model is done with the following code snippet. The code is slightly shortened so it can better be displayed here.

```
HP = {
    'n_conv_layers': hp.Int('n_conv_layers', 1, 4),
    'input_conv_layer_filters': hp.Choice('input_conv_layer_filters',
                                          values=[32, 64, 128, 256, 512],
                                          default=256),
    'input_conv_layer_kernel_size': hp.Choice('input_conv_layer_kernel_size',
                                              values=[3, 5, 7, 9]),
    'n_dense_layers': hp.Int('n_dense_layers', 1, 4),
    'learning_rate': hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
}

for i in range(HP['n_conv_layers']):
    HP[f'conv_layer_{i}_filters'] = hp.Choice(f'conv_layer_{i}_filters',
                                             values=[32, 64, 128, 256, 512],
                                             default=256)
    HP[f'conv_layer_{i}_kernel_size'] = hp.Choice(f'conv_layer_{i}_kernel_size',
                                                  values=[3, 5, 7, 9])
for i in range(HP['n_dense_layers']):
    HP[f'dense_layer_{i}_nodes'] = hp.Choice(f'dense_layer_{i}_nodes',
                                            values=[32, 64, 128, 256, 512],
                                            default=256)
```

This code creates the hyperparameter search space. All the values that are dynamically chosen by Keras-Tuner lay in this space, namely:

- number of convolutional layers
- number of filters of each layer
- size of kernel used for each layer
- number of fully connected dense layers
- learning rate

Just specifying hyperparameters for one convolutional layer it would look like the following.

```
Conv1D(filters=hp.Choice('conv_filters', values=[16, 32, 64, 128, 256, 512]))
```

Here the reason for defining all possible values is to have a large difference between the smallest and largest possible value, while limiting the options to the powers of two and not all integers between the specified range. This wouldn't be necessary when defining the number of convolutional layers for example, since one only has to define the lowest and largest number of layers the hypertuner may try out. The two for loops set the number of nodes per layer (and the kernel size in the case of convolutional layers), which is dependent on how many layers have been chosen for the current experiment. There are default values set, which come from having run the hyperparameter optimization prior and picked out values that performed well in general, so the system would find an optimal configuration faster.

Using these values we can build a model:

```
model = Sequential()

model.add(Conv1D(filters=HP['input_conv_layer_filters'],
                 kernel_size=HP['input_conv_layer_kernel_size'],
                 activation='relu',
                 input_shape=(self.input_length, 1)))

for i in range(HP['n_conv_layers']):
    model.add(Conv1D(filters=HP[f'conv_layer_{i}_filters'],
                     kernel_size=HP[f'conv_layer_{i}_kernel_size'],
                     activation='relu'))
```

```
    model.add(MaxPooling1D(pool_size=hp.Int('max_pool_size', 1, 4)))

model.add(Flatten())

for i in range(HP['n_dense_layers']):
    model.add(Dense(HP[f'dense_layer_{i}_nodes']))

model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(HP['learning_rate']),
              metrics=['accuracy'])
```

A link to GitHub and the full code can be found in Appendix 10.4. This is how the dynamic hyperparameters are inserted into the models that were defined in the previous code snippet. When doing the hypertuning, Keras builds the model with a fixed set of hyperparameters, trains it for a set number of epochs (12 in this case) and evaluates its performance on the validation set, then picks a different set of values, rebuilds the model and retrains. Note that in the actual implementation I used `hyperparameters` instead of `HP`, here it is modified for displaying purposes. At the end of 40 trials the code returns the best set of hyperparameters with which a model is trained for 60 epochs.

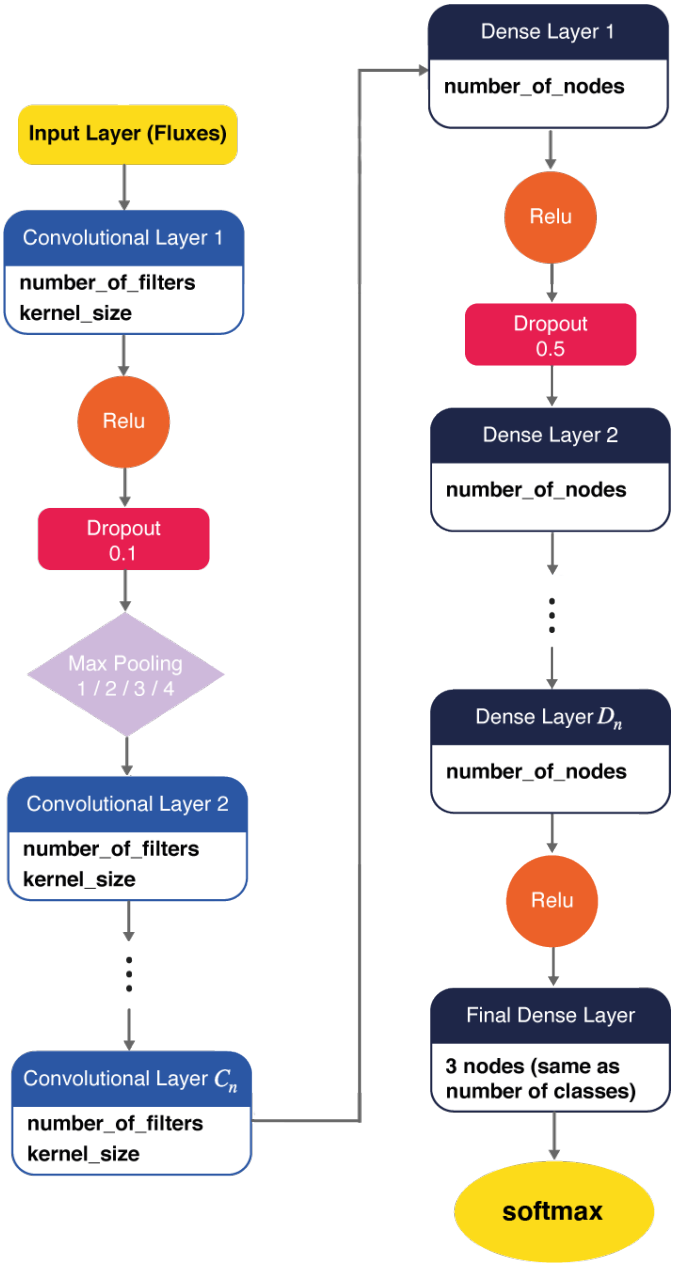



Figure 10: **CNN architecture** where the number of filters, filter size, the number of layers, number of nodes in dense layers and the last activation are all dynamically tuned. The three dots are a placeholder for an arbitrary number of repeating layers including `ReLU`, `Dropout` and `MaxPooling`. If max pooling of 1 is chosen by the hypertuning algorithm it means that there is no pooling applied.

## 6.4 Results

An experiment was run in which hypertuning algorithms of Keras went through 40 combinations of hyperparameters (40 trials) training 51,200 samples for 10 epochs after which the hyperparameters and the validation scores were recorded. Table 3 and 4 shows 16 trials, including the best performing one highlighted in bold. I have saved the hyperparameters of the model which had the highest validation accuracy and trained the model for 60 epochs recording its train and validation loss (Figure 12 and 13).

Even though the CNN was not provided important information such as redshift or Petrosian magnitudes, it was able to reach good performance across all classes. Training for 60 epochs on 51,200 sources and testing on 12,800 it had a **training accuracy: 0.999** and **test accuracy: 0.989**. Important to note, that the baseline performance for 3 classes isn't 0.33, since 56% of the dataset consists of galaxies. Table 2 shows the confusion matrix on the test set, Figure 12 shows the recorded training and validation accuracies and Figure 13 shows the recorded training and validation losses.

| Confusion Matrix | | Target Class | | |
|---|---|---|---|---|
| | | GALAXY | STAR | QSO |
| Predicted | GALAXY | 9955 | 21 | 7 |
| | STAR | 58 | 1110 | 9 |
| Class | QSO | 24 | 11 | 1605 |

Table 3: **CNN**: Confusion Matrix tested on 12,800 sources from the test set. The model was trained using 51,200 sources with no Gaussian smoothing. This amounts to a test accuracy of 0.9898

In the following two figures I would like to present typical predictions of the CNN model described above. Figure 14 shows 9 typical examples of predictions where the model was correct, and Figure 15 shows 9 typical incorrect predictions.

It is important to note that some of these plots include negative flux values, which are not physically valid, since fluxes are proportional to the number of

| | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Trial 5 | Trial 6 | Trial 7 | Trial 8 |
|---|---|---|---|---|---|---|---|---|
| validation accuracy | 0.980 | 0.951 | 0.962 | 0.973 | 0.979 | 0.978 | 0.921 | 0.968 |
| learning rate | 0.0001 | 0.0001 | 0.0001 | 0.001 | 0.0001 | 0.0001 | 0.0001 | 0.001 |
| no. conv layers | 5 | 6 | 5 | 5 | 6 | 6 | 6 | 6 |
| max pool size | 2 | 1 | 2 | 2 | 3 | 4 | 2 | 3 |
| conv layer 1 no. filters | 512 | 64 | 128 | 512 | 512 | 128 | 64 | 64 |
| conv layer 1 kernel size | 5 | 5 | 3 | 5 | 7 | 5 | 3 | 5 |
| conv layer 2 no. filters | 128 | 128 | 64 | 512 | 64 | 128 | 64 | 256 |
| conv layer 2 kernel size | 5 | 3 | 7 | 7 | 5 | 7 | 3 | 7 |
| conv layer 3 no. filters | 64 | 256 | 128 | 128 | 512 | 512 | 256 | 128 |
| conv layer 3 kernel size | 5 | 3 | 3 | 3 | 7 | 3 | 3 | 5 |
| conv layer 4 no. filters | 64 | 128 | 128 | 128 | 64 | 256 | 128 | 128 |
| conv layer 4 kernel size | 3 | 3 | 5 | 7 | 7 | 3 | 3 | 5 |
| conv layer 5 no. filters | 128 | 256 | 64 | 64 | 128 | 64 | 256 | 128 |
| conv layer 5 kernel size | 3 | 5 | 5 | 3 | 3 | 3 | 3 | 3 |
| conv layer 6 no. filters | - | 64 | - | - | 128 | 128 | 256 | 128 |
| conv layer 6 kernel size | - | 5 | - | - | 5 | 3 | 3 | 7 |
| no. dense layers | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| dense layer 1 no. neurons | 64 | 512 | 128 | 256 | 256 | 64 | 128 | 128 |
| dense layer 2 no. neurons | 128 | 512 | 512 | 256 | 256 | 256 | 128 | 256 |
| dense layer 3 no. neurons | 512 | 64 | 256 | 128 | 512 | 512 | 64 | 128 |
| dense layer 4 no. neurons | 512 | 128 | 256 | 512 | 512 | 64 | 256 | 256 |
| dense layer 5 no. neurons | 128 | 512 | 512 | 128 | 512 | 128 | 512 | 512 |

Table 4: Different hyperparameter values explored by the hypertuning algorithm

| | Trial 9 | Trial 10 | Trial 11 | Trial 12 | **Trial 13** | Trial 14 | Trial 15 | Trial 16 |
|---|---|---|---|---|---|---|---|---|
| validation accuracy | 0.974 | 0.969 | 0.976 | 0.969 | **0.982** | 0.977 | 0.971 | 0.981 |
| learning rate | 0.0001 | 0.001 | 0.0001 | 0.0001 | **0.0001** | 0.0001 | 0.001 | 0.0001 |
| no. conv layers | 6 | 2 | 6 | 6 | **6** | 4 | 5 | 6 |
| max pool size | 3 | 2 | 2 | 1 | **3** | 2 | 2 | 3 |
| conv layer 1 no. filters | 256 | 512 | 128 | 512 | **256** | 256 | 64 | 512 |
| conv layer 1 kernel size | 3 | 3 | 5 | 7 | **7** | 7 | 5 | 7 |
| conv layer 2 no. filters | 256 | 256 | 128 | 64 | **64** | 512 | 128 | 512 |
| conv layer 2 kernel size | 5 | 3 | 3 | 3 | **7** | 3 | 7 | 3 |
| conv layer 3 no. filters | 64 | - | 64 | 512 | **512** | 256 | 128 | 64 |
| conv layer 3 kernel size | 5 | - | 5 | 5 | **3** | 3 | 5 | 3 |
| conv layer 4 no. filters | 64 | - | 256 | 256 | **128** | 256 | 256 | 256 |
| conv layer 4 kernel size | 7 | - | 7 | 5 | **5** | 3 | 7 | 3 |
| conv layer 5 no. filters | 256 | - | 128 | 64 | **128** | - | - | 256 |
| conv layer 5 kernel size | 5 | - | 7 | 3 | **7** | - | - | 3 |
| conv layer 6 no. filters | 64 | - | 128 | 128 | **256** | - | - | - |
| conv layer 6 kernel size | 3 | - | 7 | 5 | **3** | - | - | - |
| no. dense layers | 4 | 1 | 4 | 4 | **5** | 4 | 4 | 4 |
| dense layer 1 no. neurons | 64 | 64 | 128 | 512 | **512** | 512 | 512 | 128 |
| dense layer 2 no. neurons | 256 | - | 128 | 512 | **512** | 256 | 128 | 256 |
| dense layer 3 no. neurons | 512 | - | 64 | 128 | **128** | 256 | 128 | 64 |
| dense layer 4 no. neurons | 128 | - | 128 | 512 | **256** | 256 | 256 | 512 |
| dense layer 5 no. neurons | - | - | - | | **256** | - | - | - |

Table 5: Different hyperparameter values explored by the hypertuning algorithm

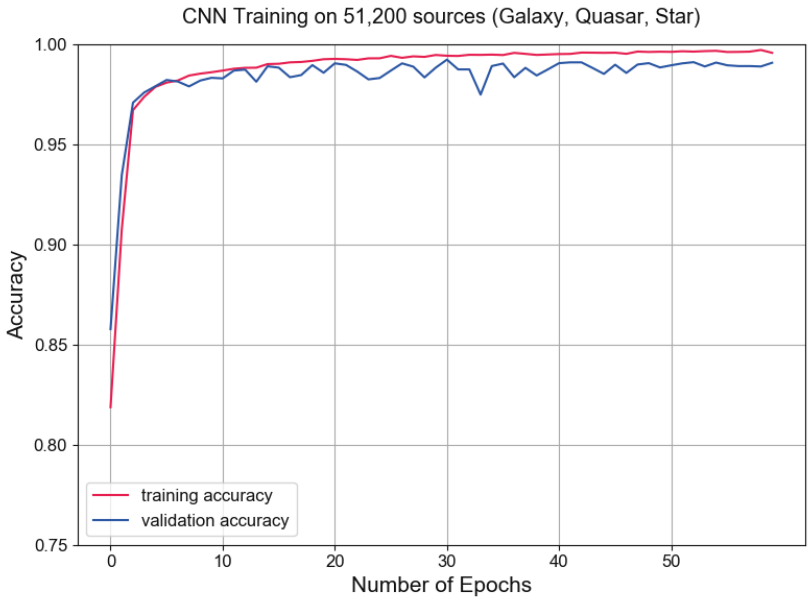



Figure 11: **CNN**: Training and validation accuracies after each epoch, trained for 60 epochs

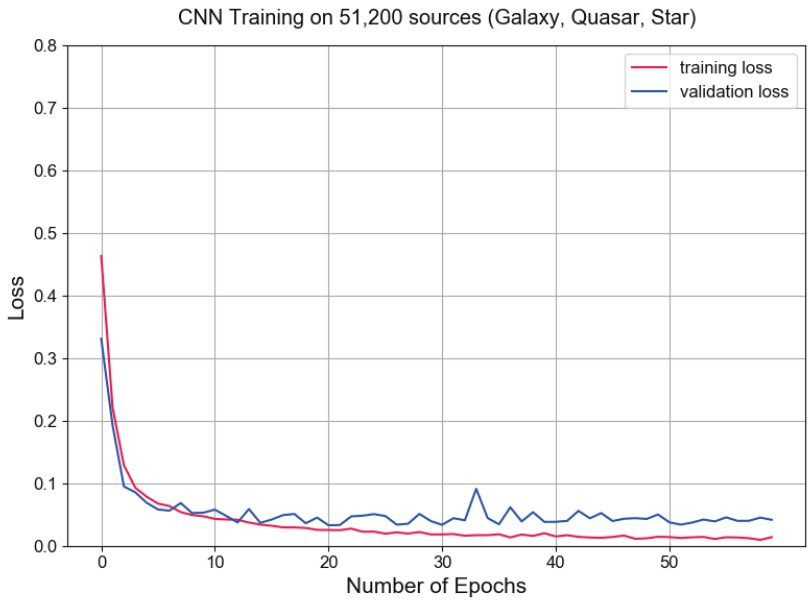



Figure 12: **CNN**: Training and validation losses after each epoch, trained for 60 epochs

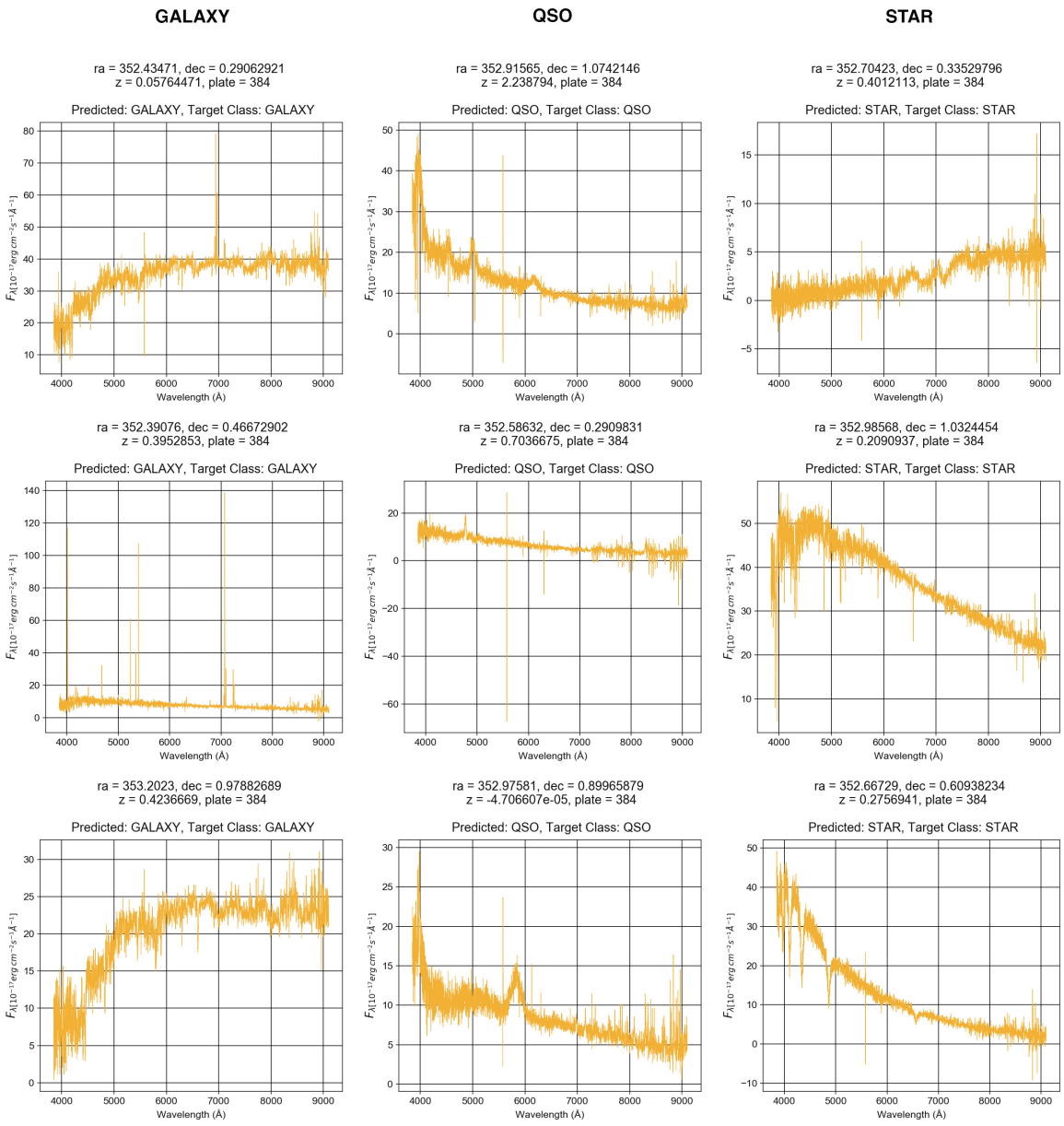



Figure 13: **CNN**: 9 typical correct predictions by the model, 3 of each class

photons hitting the measuring device at different frequencies, which can never be negative. I have left the negative values in the plots because the model was trained on the fluxes directly as they were downloaded from SDSS. A future opportunity

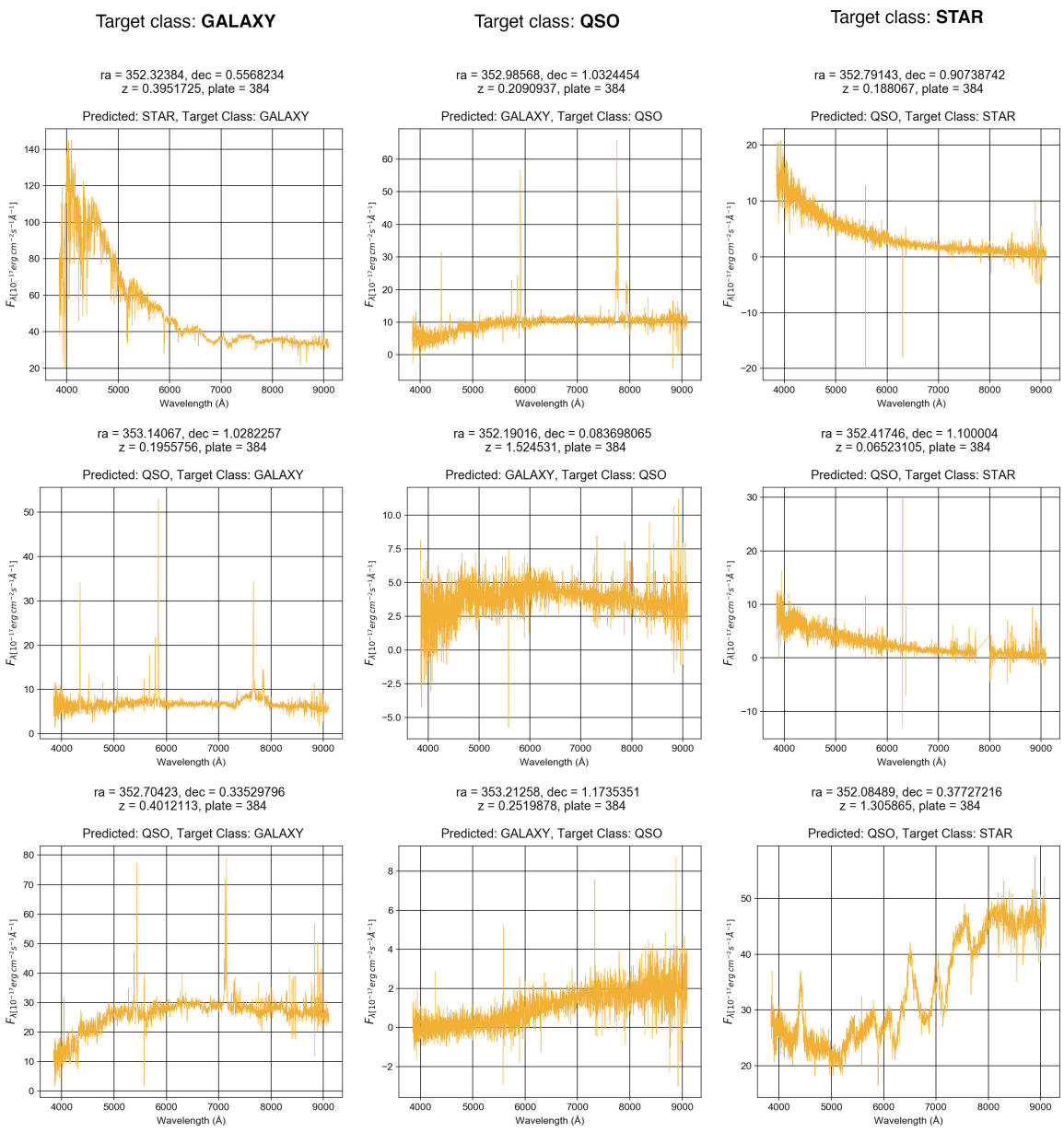



Figure 14: **CNN**: 9 typical incorrect predictions by the model, 3 of each class

could be to do processing that would remove these artifacts before training.

Another interesting point I would like to reflect on is that though correctly identifying sources as stars was relatively accurate, when the model made an error,

a star was often mistaken for a quasar, or the reverse. I'm not quite certain why this happens, but I would like to present two sources next to each other that may explain part of the problem (Figure 16). Another reason could be that there is a varying signal to noise ratio across different sources, which may confuse the network on some examples.

Looking at the correctly identified examples one may see the similarities between the shape of the two spectra, although the clear absorption lines should be indicative of a star. Note that the model was not provided with the value of $z$, a problem I will address in the next section.

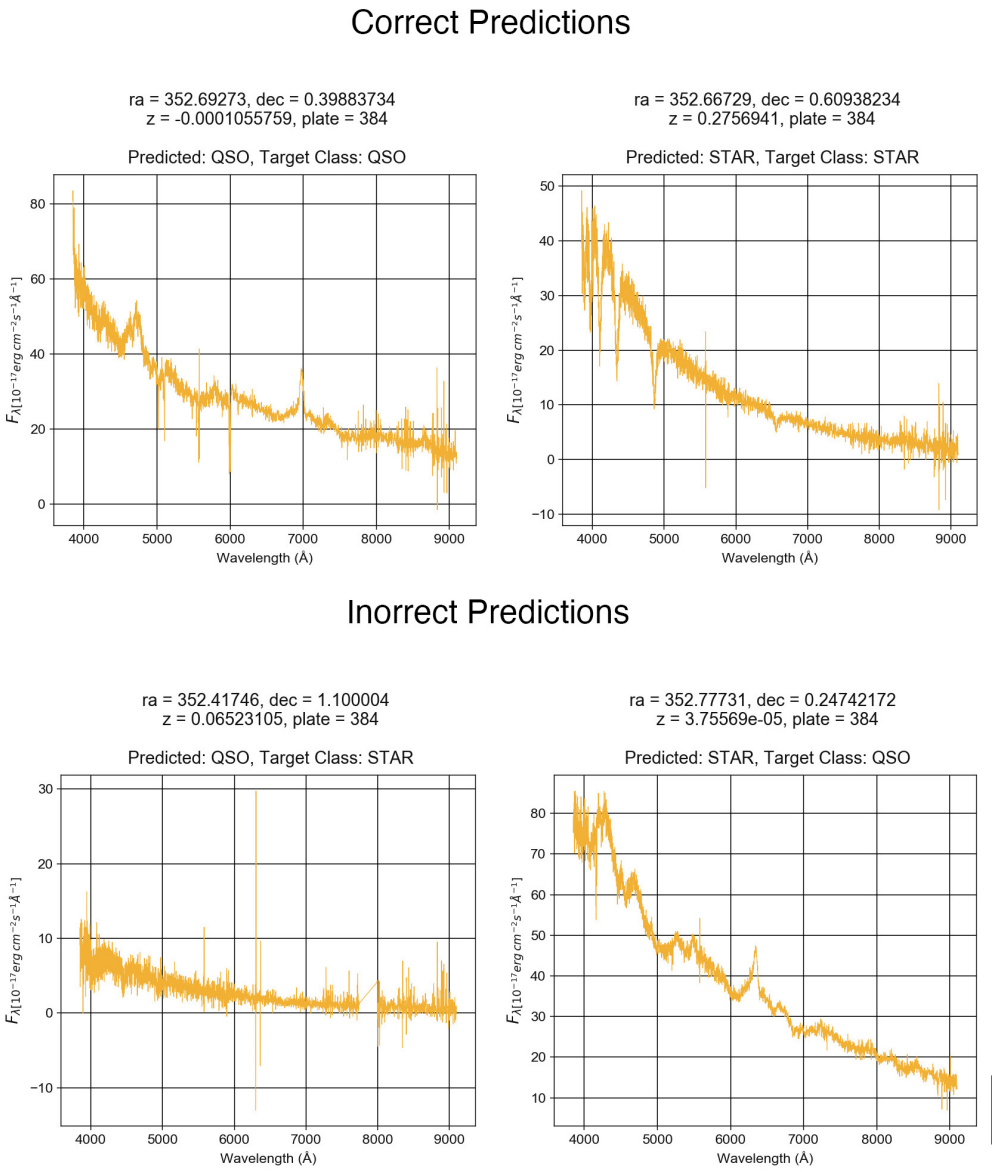



Figure 15: **CNN**: Corrrectly identified QSO and STAR in the first row with that have clear visual similarities, and two misclassified examples where a QSO was mistaken for a STAR and the reverse case

# 7 Mixed Input Neural Network

One of the neural network architectures proposed for this dataset is a mixed input neural network or mixed feature based neural network. The intuition behind it is to separate the spectra (the arrays of fluxes) from spectro-photometric features such as the magnitudes, z and the emission / absorption lines previously computed. Although the spectral lines might be learnt by the model from the fluxes only, adding the precomputed values improved performance. Fluxes were fed into a 1D CNN as input, while a simple feed forward neural network was used for learning other numerical data. Finally the two are combined with a MLP that has n neurons as its last layer, n being the number of target labels. Figure 17 describes the architecture of the network used. The dataset contained 96,115 spectra consisting of 53,825 galaxies (56.0%), 32,488 quasars (33.8%) and 9,802 stars (10.2%). When classifying the three main classes of `galaxy`, `qso` and `star`, the network reached a **training accuracy: 0.998** and a **test accuracy: 0.993**.

## 7.1 From Data to Training

Before further discussion on the architecture of the network and its performance, it is important to spend some time on the data preprocessing that is done right before training and is specific to the mixed-input architecture. As discussed in section **3.2**, the dataset has previously been prepared and stored in an HDF5 file, which consists of three tables. The first table contains the fluxes expanded into columns, being the number of fluxes per object or more formally:

$$F = \begin{bmatrix} f_{11} & f_{12} & \dots & f_{1n} \\ f_{21} & f_{22} & \dots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{k1} & f_{k2} & \dots & f_{kn} \end{bmatrix} \tag{17}$$

where $F$ is the matrix of fluxes from every source, $f$ is a single flux, $k$ is the number of sources and $n$ is the number of fluxes per source.

The second table simply contains a list of wavelengths that correspond to the fluxes stored in the first table.

The third table is similarly structured as the first one except that each holds a value of different type. The complete list of columns is: *objid*, *fluxObjID*, *plate*, *class*, *subClass*, *z*, *zErr*, *petroMag_u*, *petroMag_g*, *petroMag_r*, *petroMag_i*, *petroMag_z*, *petroMagErr_u*, *petroMagErr_g*, *petroMagErr_r*, *petroMagErr_i* and *petroMagErr_z* and 14 spectral lines that had previously been computed.

The first step of the data preprocessing is to remove all columns that are not used for training. The reason for removing these values is to prevent the overfitting of the model on features that are non-generalizable and to improve training performance by removing noise that does not constitute learnable useful information.

**objid**, **fluxObjID**: IDs do not and should not contain useful information and arer often generated randomly.

**ra**, **dec**: In modern cosmology a widely adopted hypothesis is that the Universe homogeneous and isotropic at all scales in all directions, especially at large cosmological distances. Based the on this, the place at which we find the source should hold no information about its characteristics

**class**, **subClass**: Depending on the type of classification one of these will become the target label or the dependent variable.

Then the data is separated into two matrices of independent variables and one matrix of dependent variables. The dependent variables or target classes can either be the three main classes (GALAXY, QSO, STAR) or subclasses of any of these main classes. To feed categorical variables to a ML algorithm it is necessary to first convert them into a numerical representation. For this project I used the one-hot encoding. In the case of the three main classes with one source in each class, the representation is:

| | GALAXY | QSO | STAR |
|---|---|---|---|
| $y_1$ | 1 | 0 | 0 |
| $y_2$ | 0 | 1 | 0 |
| $y_3$ | 0 | 0 | 1 |

## 7.2 Network Architecture

The architecture used here is made of three main parts. A CNN that receives fluxes as its inputs, a MLP that receives spectro-photometric features as its inputs and a final MLP with two dense layers whose inputs are the outputs of the two previous networks and its output layer contains the same number of nodes as there are labels. See Figure 17 for the exact architecture used for classification.

## 7.3 Results

The main focus of this project was to classify sources between the three main classes (`galaxy`, `qso`, `star`) but to see how the neural network performs in subclass classification of galaxies and stars I have trained a slightly modified model to classify 8 classes of galaxies and 15 classes of stars whose results will be discussed in section 7.3.2 and 7.3.3.

### 7.3.1 Classification on the 3 main classes

Experiments were done on various slices of the data set ranging from 1,000 to 96,115 (all sources) classifying whether a source is a `galaxy`, `qso` or `star`. I found steady increase in accuracy as more sources were used, which leveled off after 64,000. To show this I trained the model for different sizes of training and validation data sets both with and without Gaussian smoothing. The results were gathered by splitting the dataset to 80% train and 20% test set, then splitting the training set to training and validation with the same ratio (Figure 10). The model was trained on the training set and validated on the validation set for 10 epochs, then using the weights from the model that had the best performance on the validation set I measured its performance on the test set. The prediction accuracies are written in table 5.

For this experiment I randomly sampled 64,000 sources from the dataset. 51,200 sources were used for training where it was split to 80% training and 20% validation and 12,800 were used for testing. The model trained on these 51,200 sources for 60 epochs where I recorded the training and validation losses and ac-

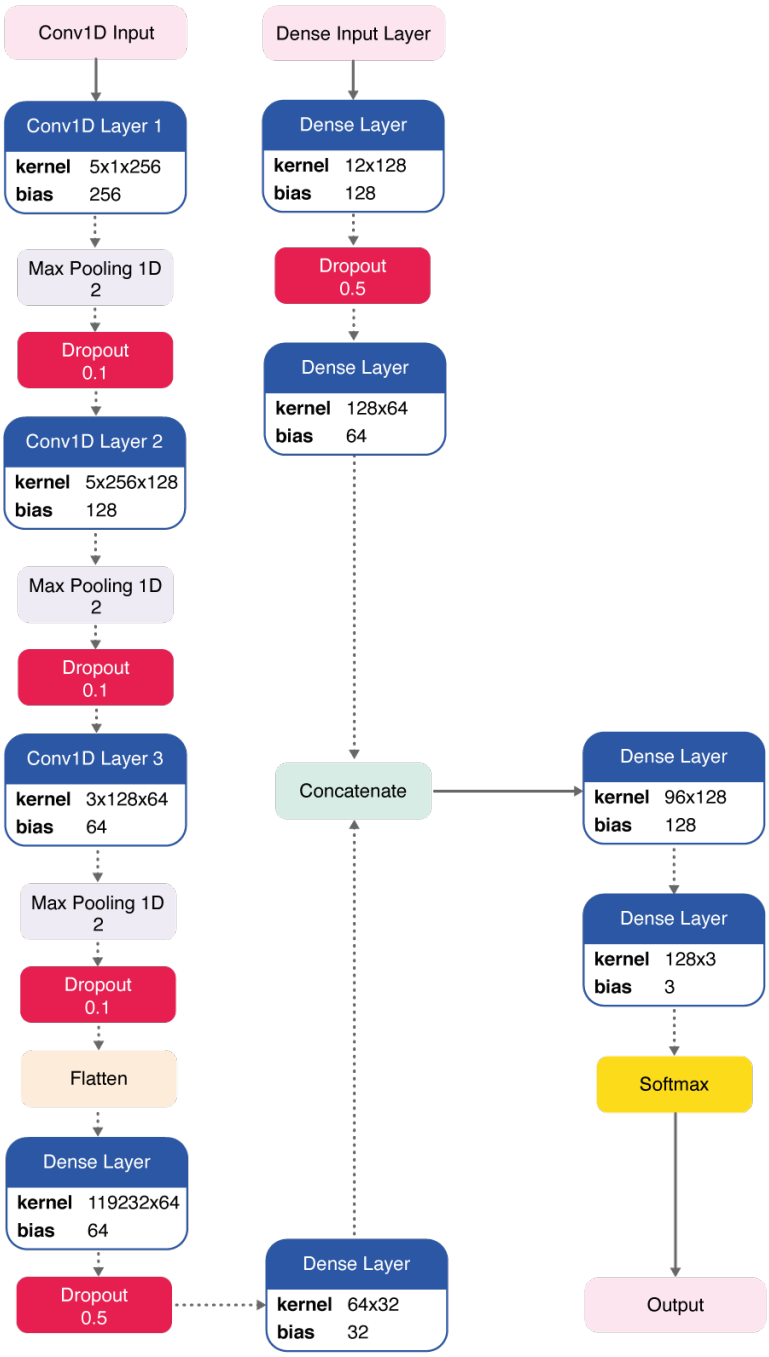



Figure 16: **Mixed-input Neural Network** architecture

| No. Sources | 1,000 | 2,000 | 4,000 | 8,000 | 16,000 | 32,000 | 64,000 | 94,114 |
|---|---|---|---|---|---|---|---|---|
| $\sigma = 0$ | 0.970 | 0.947 | 0.975 | 0.979 | 0.982 | 0.986 | 0.991 | 0.986 |
| $\sigma = 4$ | 0.960 | 0.980 | 0.989 | 0.983 | 0.985 | 0.986 | 0.989 | 0.984 |

Table 6: **Mixed Input Neural Network**: validation accuracies from different-sized data sets. Best chosen from the first 5 epochs. $\sigma = 0$ is the original fluxes, while $\sigma = 4$ is modified fluxes which had a 1D Gaussian filter convolution applied on them

curacies (see Figure 18 and 19). Regardless of the learning rate or the whether dropout is used the model converges on a 0.99 accuracy relatively early and stays there throughout training, which in turn gives good performance on the test set, roughly equal to the validation. In this example the model had a **training accuracy** of 0.998 and a **test accuracy** of 0.993. Important to note, that the baseline performance for 3 classes isn't 0.33, since 56% of the dataset consists of galaxies. The confusion matrix produced on 12,800 examples is shown in table 6.

| Confusion Matrix | | Target Class | | |
|---|---|---|---|---|
| | | GALAXY | STAR | QSO |
| Predicted | GALAXY | 9921 | 28 | 5 |
| | STAR | 49 | 1001 | 2 |
| Class | QSO | 6 | 0 | 1788 |

Table 7: **Mixed Input Neural Network**: Confusion Matrix tested on 12,800 sources from the test set. The model was trained using 51,200 sources with no Gaussian smoothing. This amounts to a test accuracy of 0.993

Figure 20 shows 9 typical correct predictions made by the model and Figure 21 shows 9 typical incorrect predictions. Note that the misclassified stars shown here are outliers rather than typical predictions of the model. In fact these were the only 3 examples of stars in the test set that the model did not classify correctly. It is probably due to the fact that redshift is provided to the network as spectro-photometric feature, and almost all stars have redshifts close to 0.

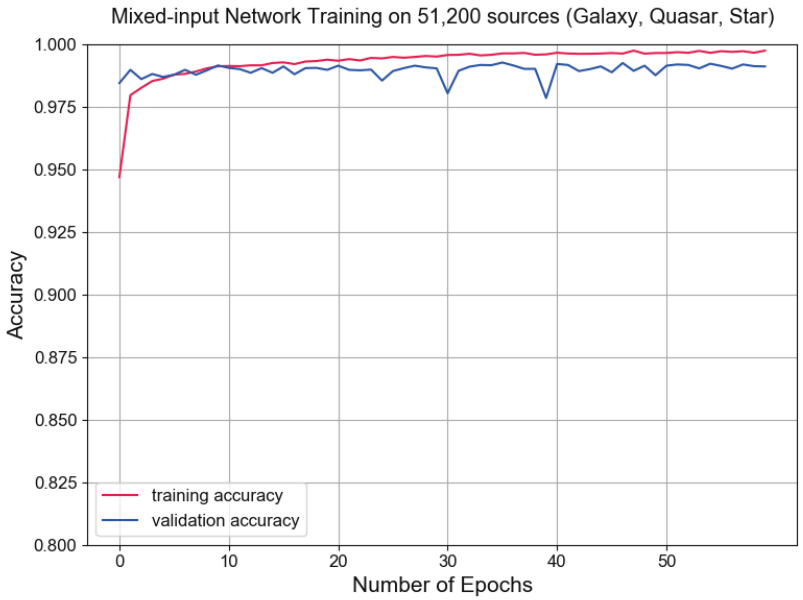



Figure 17: **Mixed-input Network**: Training and validation accuracies after each epoch, trained for 60 epochs

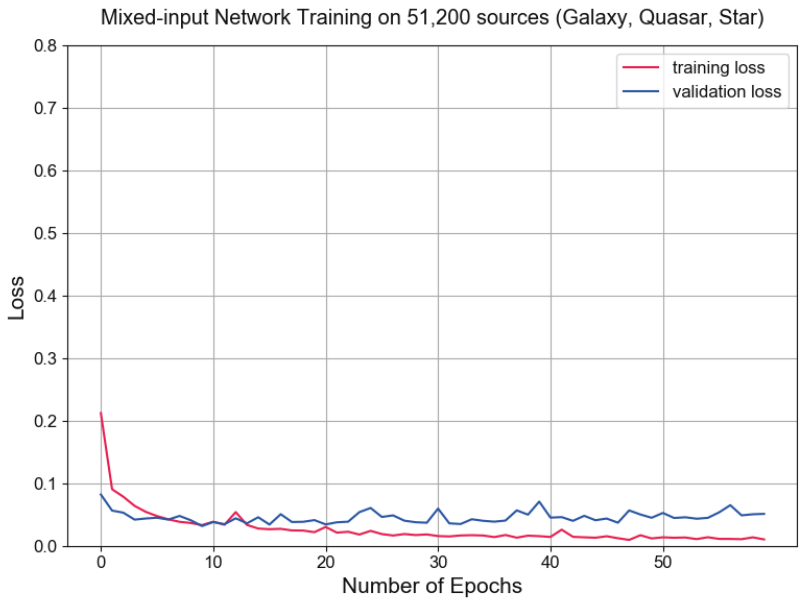



Figure 18: **Mixed-input Network**: Training and validation losses after each epoch, trained for 60 epochs

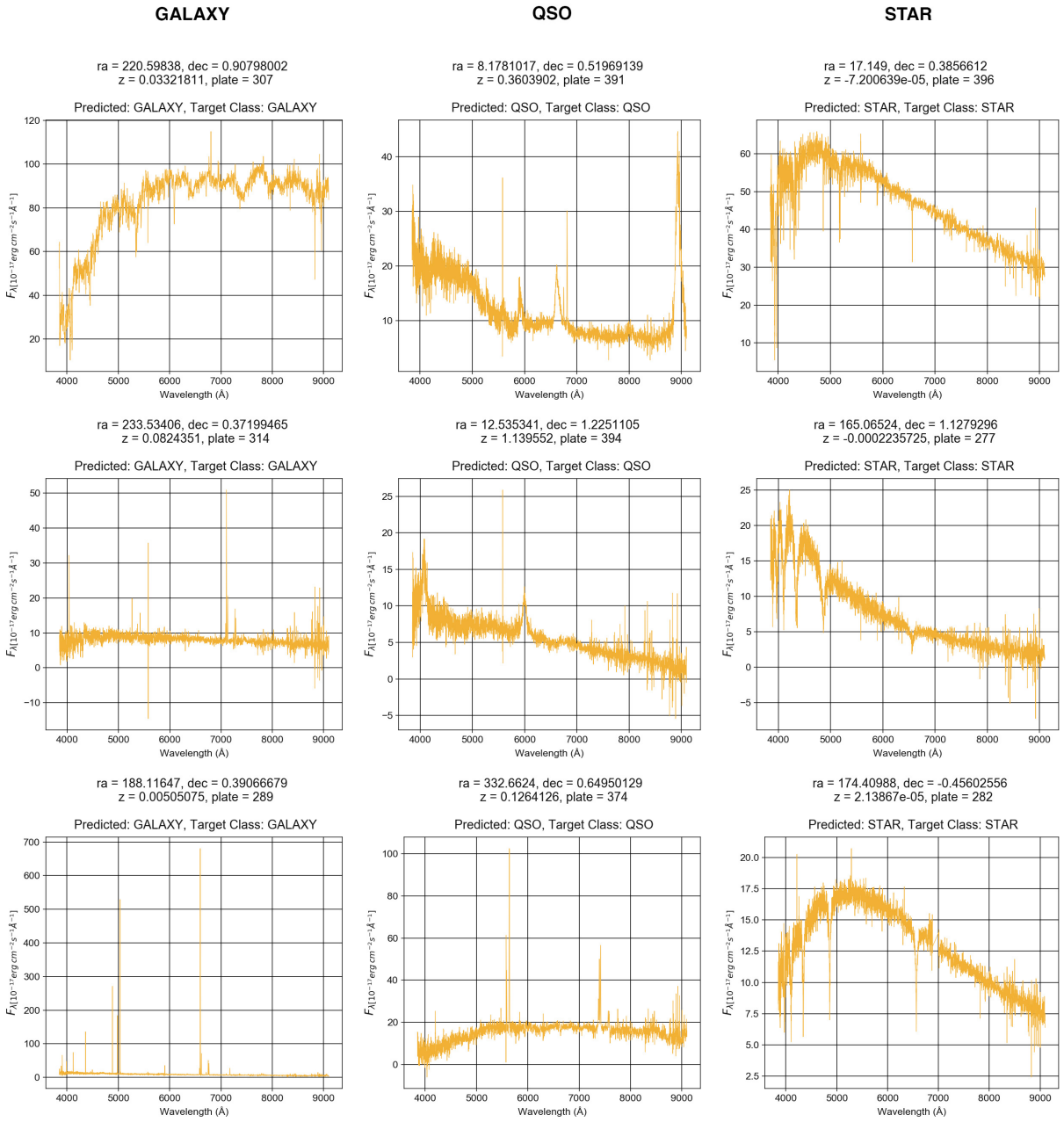



Figure 19: **Mixed-input Network**: 9 typical correct predictions by the model, 3 of each class

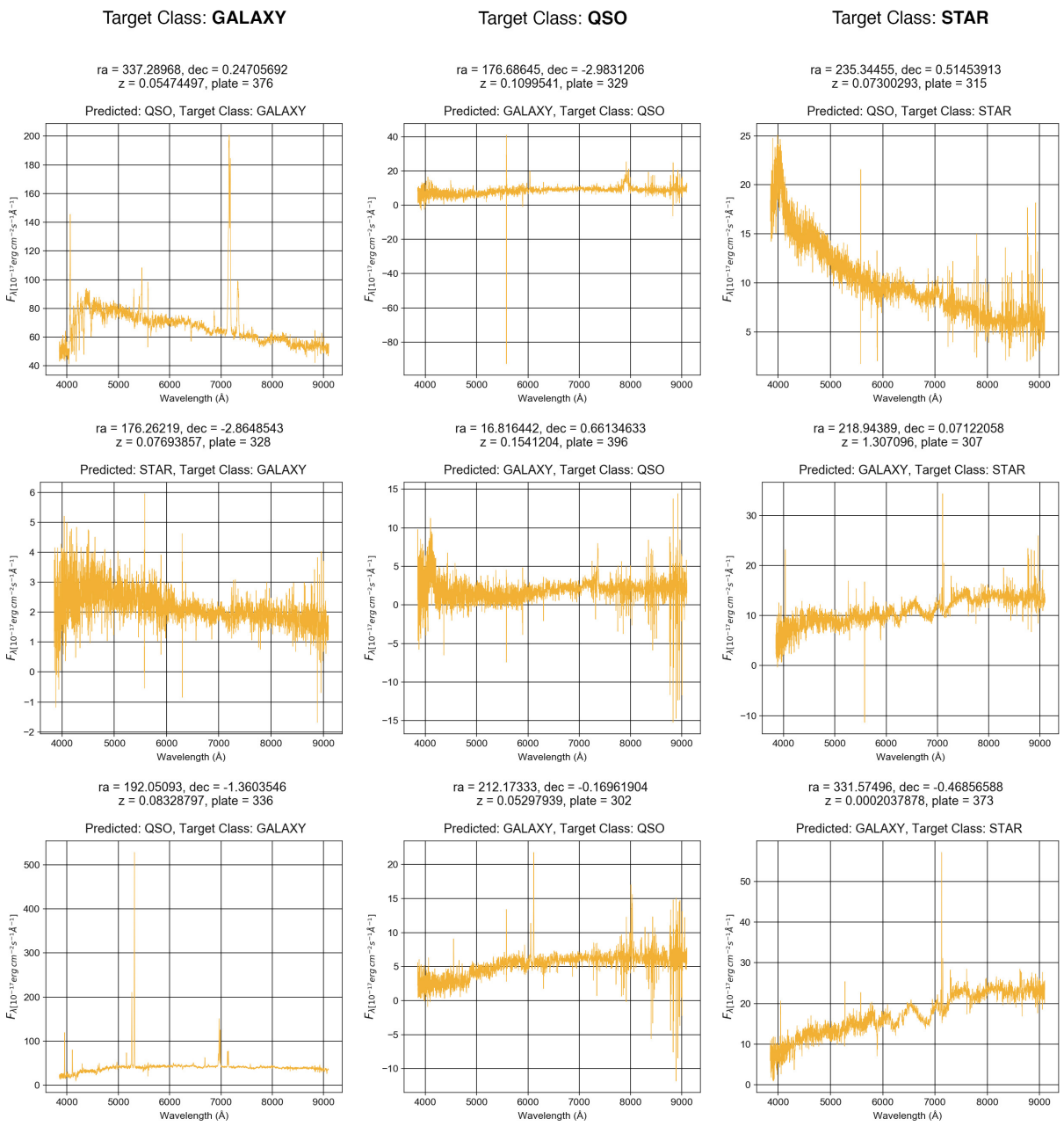



Figure 20: **Mixed-input Network**: 9 typical incorrect predictions by the model

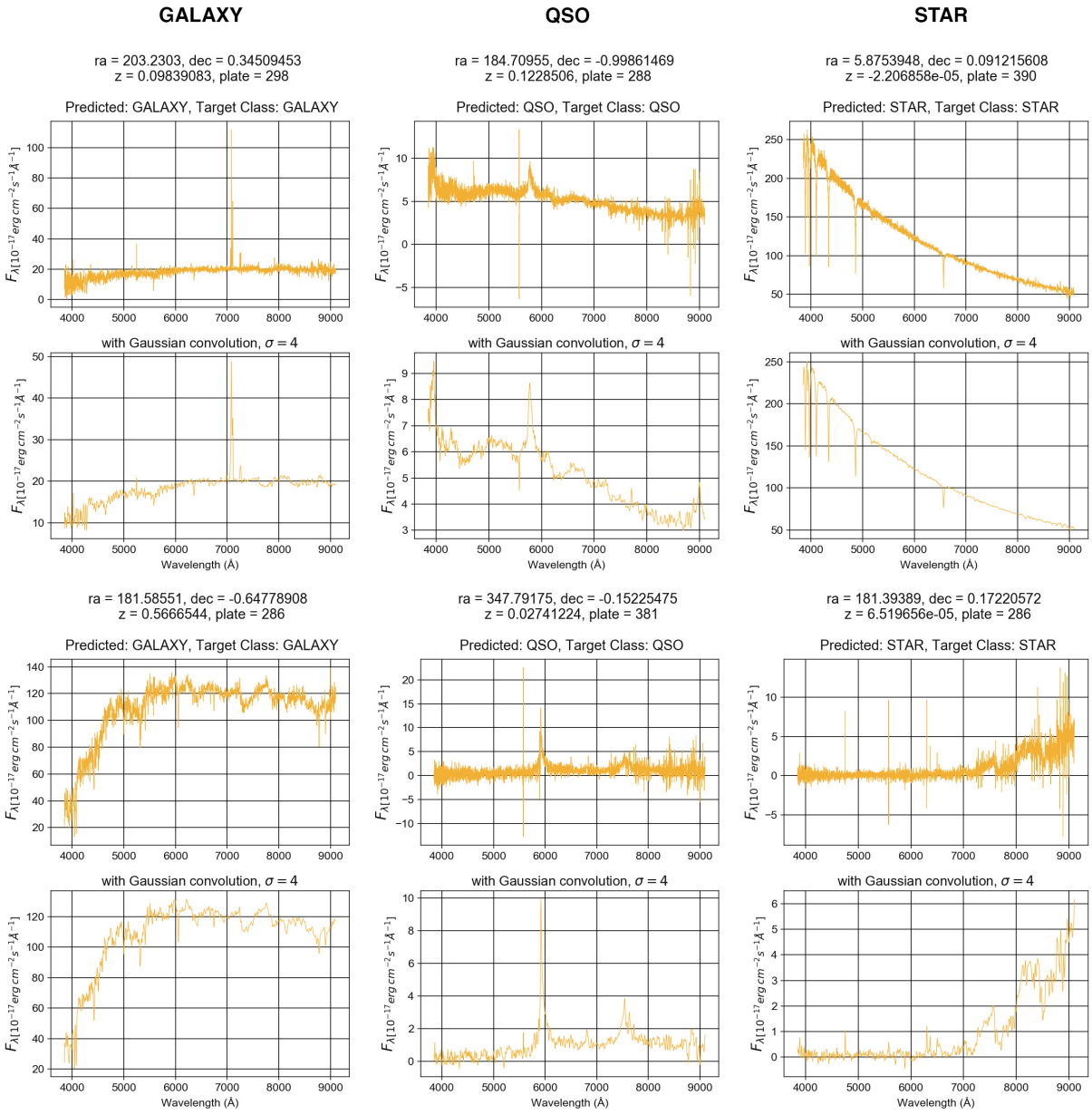



Figure 21: **Mixed-input network**: 6 typical correct predictions by the model on sources whose fluxes had been modified with Gaussian smoothing with $\sigma = 4$

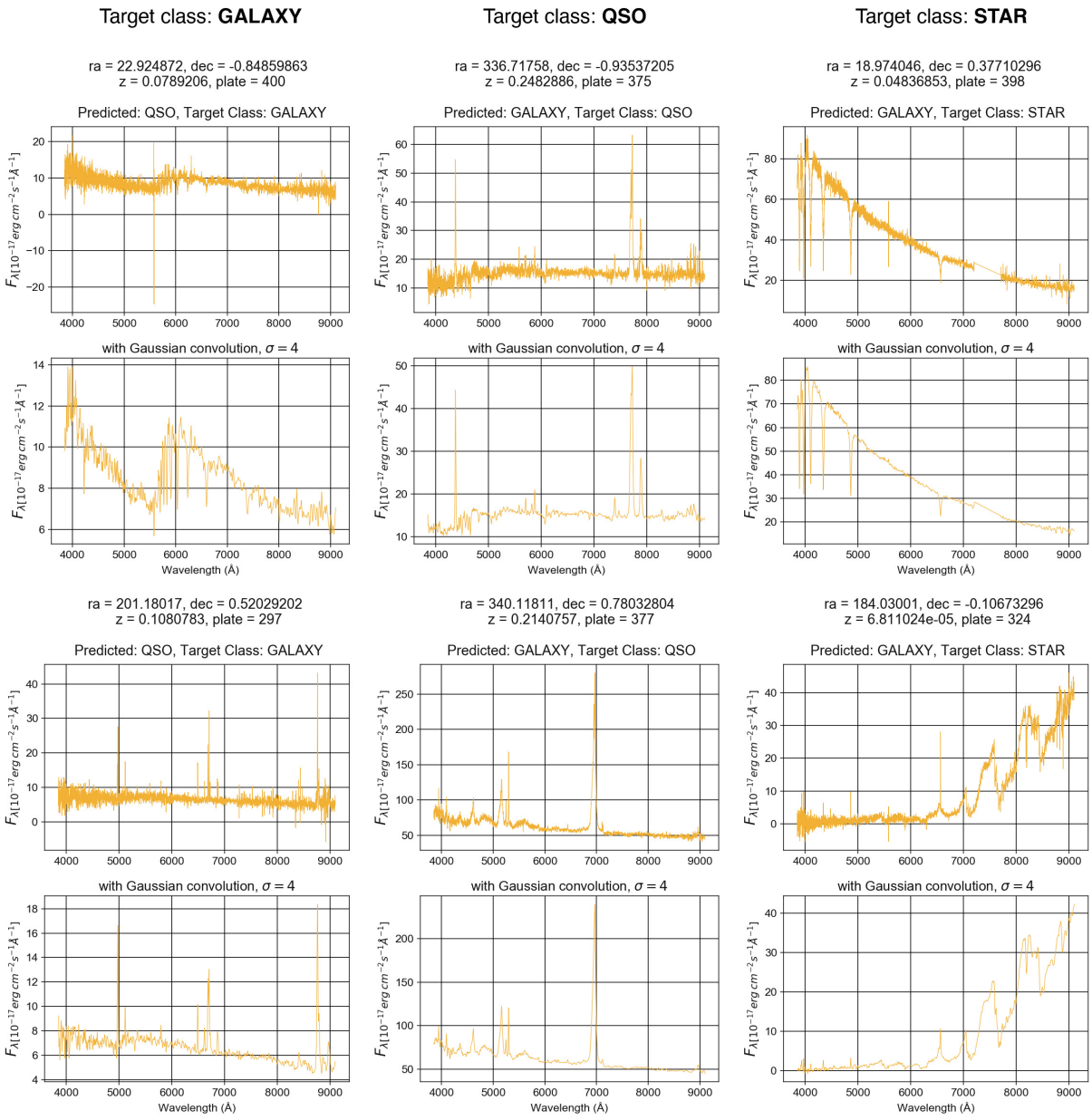



Figure 22: **Mixed-input network**: 6 typical incorrect predictions by the model on sources whose fluxes had been modified with Gaussian smoothing with $\sigma = 4$. Note that the star misclassification cannot be considered typical, since these are the only 2 cases in the test set where the model predicted incorrectly.

### 7.3.2 Classifying Galaxies

The other classification this project is focusing on is to classify sources based on their subclasses. For this problem I trained the same mixed-input network described above since it has proved to recognize patterns in spectra with good accuracies. The model was trained for 60 epochs on 39,992 sources and tested on 9,998. It model reached a **train accuracy** of **0.966** and a **test accuracy** of **0.863**. Note that there were 8 classes in total where 58.9% of the sources had no class that I included with class `NULL`. Figure 26 and Figure 27 shows 9 typical correct and incorrect predictions of the model. These predictions were hand-picked out of a pool of 100 plots with the aim of presenting versatile examples.

It appears that this model does not generalize as well for galaxy subclasses, which may be because of a gradual change throughout galaxies where there is no sharp distinctions between some of the classes. Another reason may be that unfortunately a large percentage of downloaded galaxies from SDSS have no subclass, which show up as NULL in the confusion matrix. This in fact may have caused most of the problems in the training process since these galaxies with no subclasses could belong to any other subclasses therefore causing confusion in the model. A future opportunity could be to remove all NULL galaxies from the dataset and train a model on the rest to see if the performance improves. Then all the NULL galaxies can be classified using the trained model and look at examples manually to see how accurate it is with respect to galaxies that SDSS did not classify.

| Confusion Matrix | | Target Class | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Null | AGN | AGN B. | B. | Starburst | Starburst B. | Starforming | Starforming B. |
| Predicted Class | Null | 5893 | 5 | 0 | 1 | 16 | 0 | 414 | 0 |
| | AGN | 76 | 46 | 0 | 0 | 0 | 0 | 61 | 0 |
| | AGN B. | 15 | 6 | 0 | 0 | 1 | 0 | 3 | 0 |
| | Broadline | 119 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| | Starburst | 15 | 0 | 0 | 0 | 517 | 0 | 225 | 0 |
| | Starburst B. | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Starforming | 340 | 7 | 0 | 0 | 40 | 0 | 2174 | 0 |
| | Starforming B. | 12 | 1 | 0 | 0 | 0 | 0 | 7 | 0 |

Table 8: **Mixed-input network**: Confusion Matrix of the predictions from galaxy subclass classification. **B. refers to Broadline**

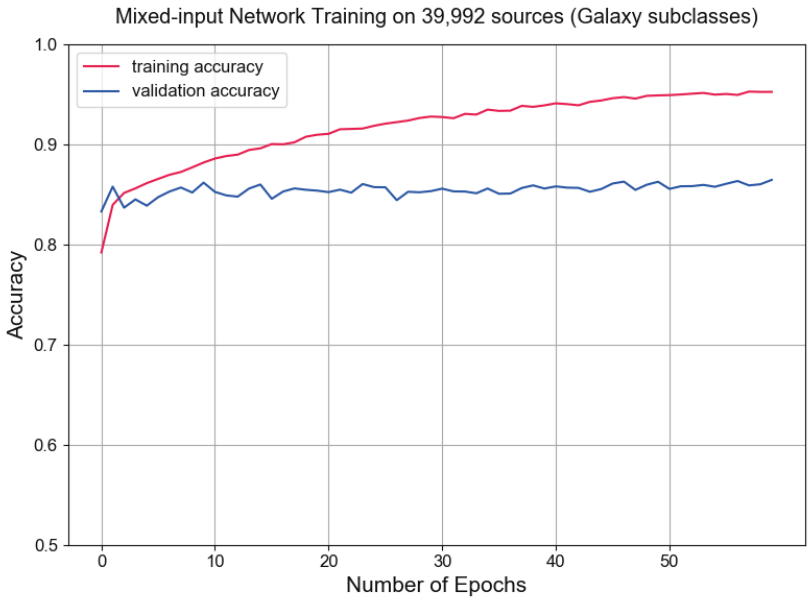



Figure 23: **Mixed-input Network**: Training and validation accuracies, trained for 60 epochs classifying galaxy subclasses

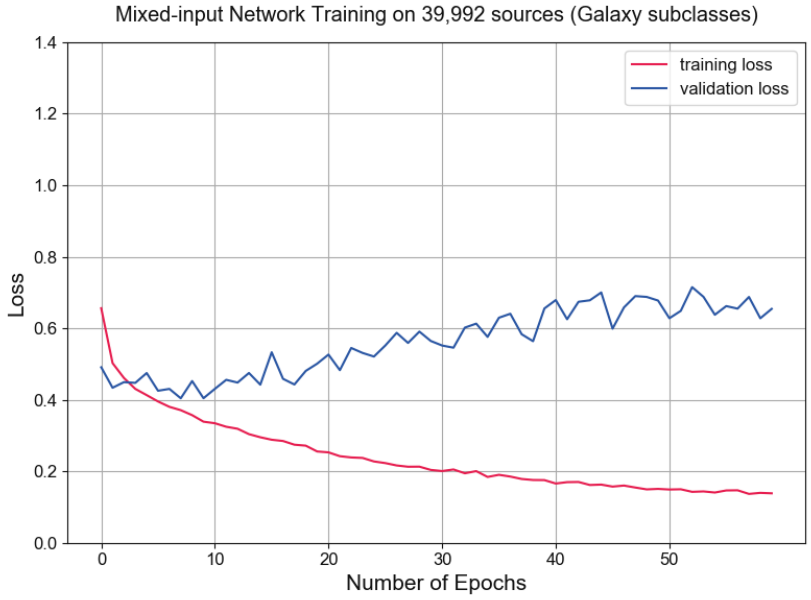



Figure 24: **Mixed-input Network**: Training and validation losses, trained for 60 epochs classifying galaxy subclasses

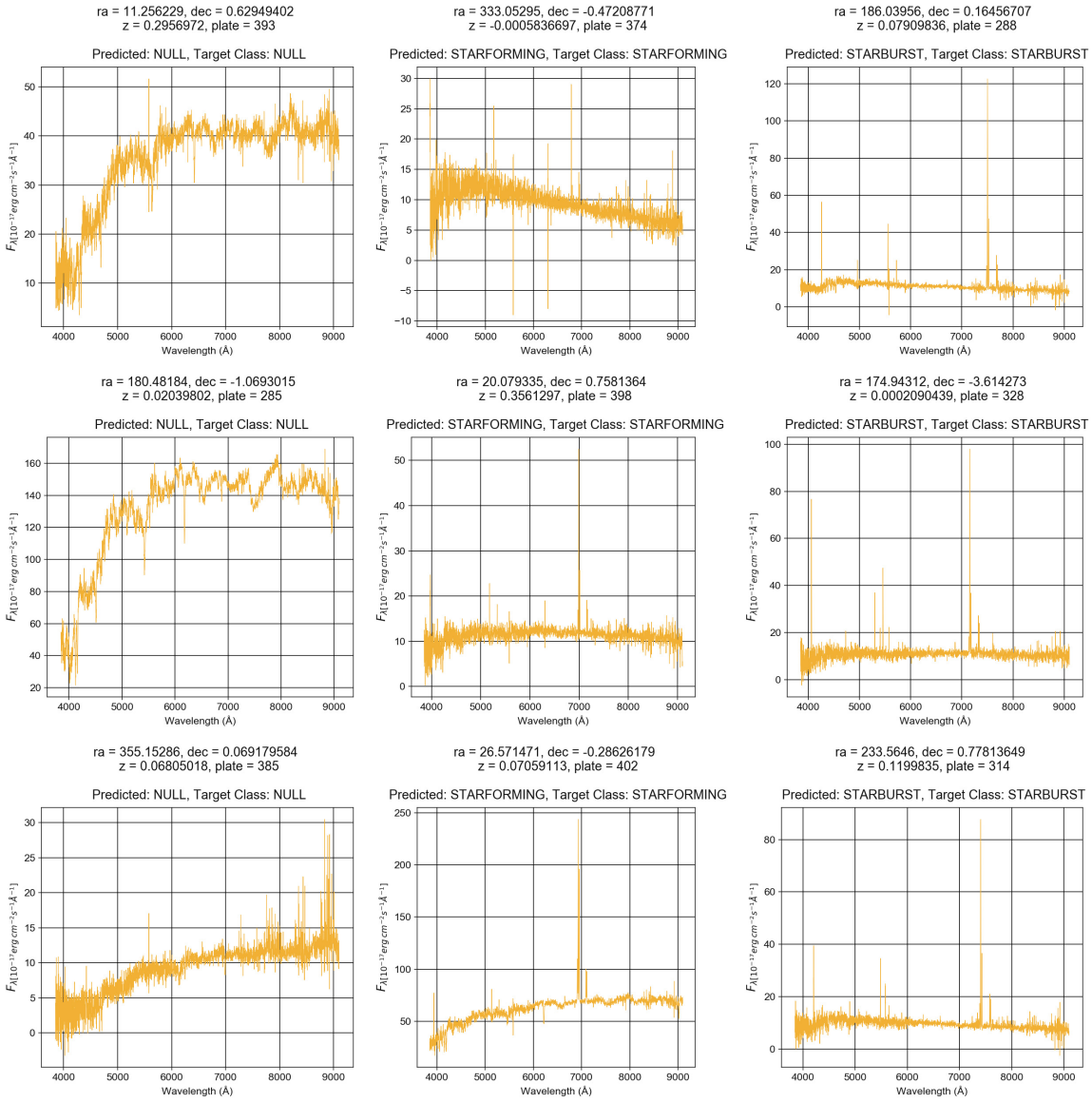



Figure 25: **Mixed-input Network - Galaxy classification**: 9 typical correct predictions by the model

### 7.3.3 Classifying Stars

The mixed-input network was trained for 60 epochs on 7,842 sources and tested on 1960. It reached a **training accuracy: 0.828** and a **test accuracy: 0.783** on

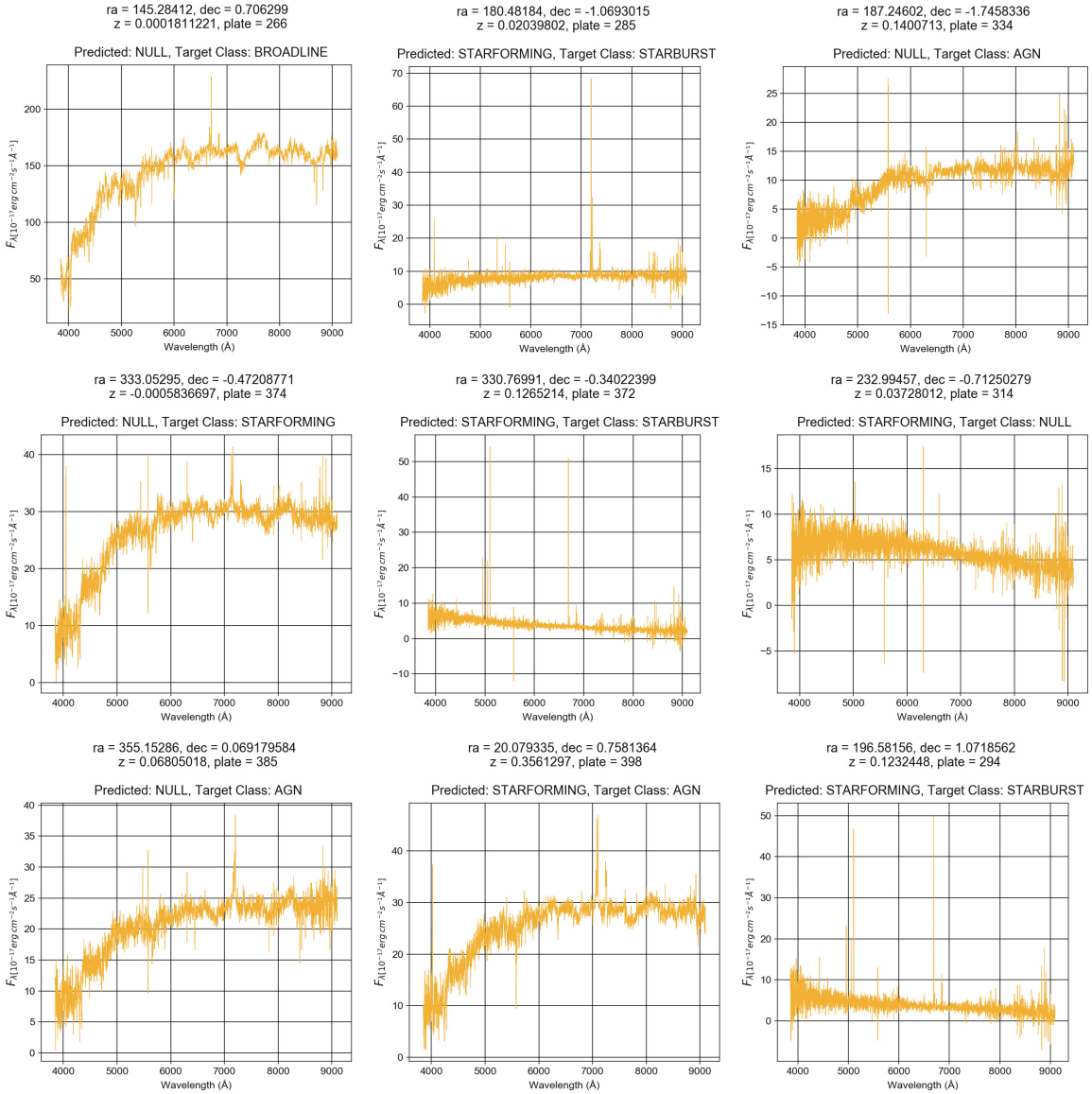



Figure 26: **Mixed-input Network - Galaxy classification**: 9 typical incorrect predictions by the model

15 classes. Figure 28 and 29 shows the training / validation accuracies and losses over 60 epochs of training. Table 8 shows the confusion matrix, Figure 30 presents 9 correct predictions of the model and Figure 31 shows 9 incorrect predictions.

| Confusion Matrix | | Target Class | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | A | B | CV | C | C WD | C lines | F | G | K | L | M | O | T2 | WD | WD M |
| Predicted Class | A | 355 | 0 | 0 | 0 | 0 | 0 | 72 | 1 | 2 | 0 | 1 | 0 | 0 | 6 | 0 |
| | B | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 16 | 0 |
| | CV | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 5 | 0 |
| | C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| | C WD | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| | C lines | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | |
| | F | 5 | 0 | 0 | 0 | 0 | 0 | 365 | 77 | 36 | 0 | 2 | 0 | 0 | 0 | 0 |
| | G | 0 | 0 | 0 | 0 | 0 | 0 | 49 | 92 | 13 | 0 | 0 | 0 | 0 | 0 | 0 |
| | K | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 281 | 0 | 0 | 0 | 0 | 0 | 0 |
| | L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 0 | 0 |
| | M | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 39 | 0 | 380 | 0 | 0 | 0 | 0 |
| | O | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 |
| | T2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | WD | 12 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 13 | 0 | 13 | 0 | 0 | 97 | 0 |
| | WD M | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

Table 9: **Mixed-input network**: Confusion Matrix of the predictions from star subclass classification. **C** refers to Carbon and **WD** refers to White Dwarf

These predictions were hand-picked out of a pool of 100 plots with the aim of presenting versatile examples.

There are several issues I had encountered regarding star classification. The most pressing of all is that I could only acquire 9,802 stars during the downloading phase. In these 9,802 sources, SDSS lists the following 41 different subclasses: A0, A0p, B6, B9, CV, Carbon, CarbonWD, Carbon_lines, F2, F5, F9, G0, G2, K1, K3, K5, K7, L0, L1, L2, L3, L4, L5.5, L9, M0, M0V, M1, M2, M2V, M3, M4, M5, M6, M7, M8, M9, O, OB, T2, WD, WDmagnetic. In machine learning classification to have only 9,802 examples with 41 target classes is considered inadequate. To address this issue and to improve performance I have combined letters into one class to reduced the number of classes and to increase classification performance. The grouped classes are: A, B, CV, Carbon, CarbonWD, Carbon_lines, F, G, K, L, M, O, T2, WD, WDmagnetic. As a future opportunity it would interesting to retrain this model with the original number of classes but where the number of sources is closer to 100,000.

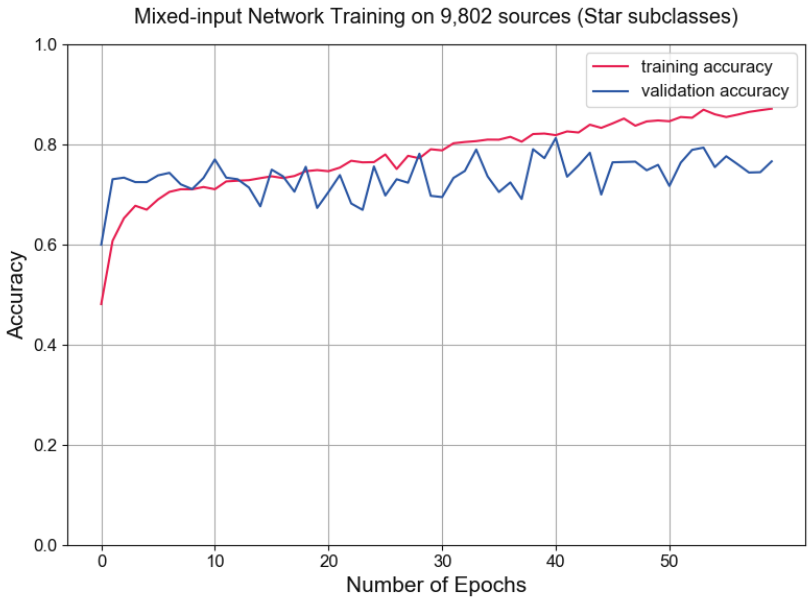



Figure 27: **Mixed-input Network**: STAR classification using 15 classes. Training and validation losses after each epoch, trained for 60 epochs

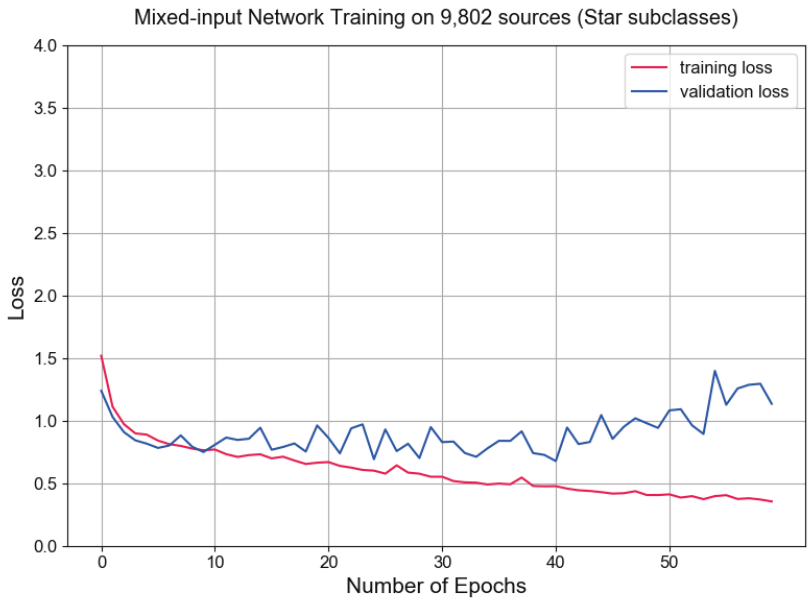



Figure 28: **Mixed-input Network**: STAR classification using 15 classes. Training and validation losses after each epoch, trained for 60 epochs

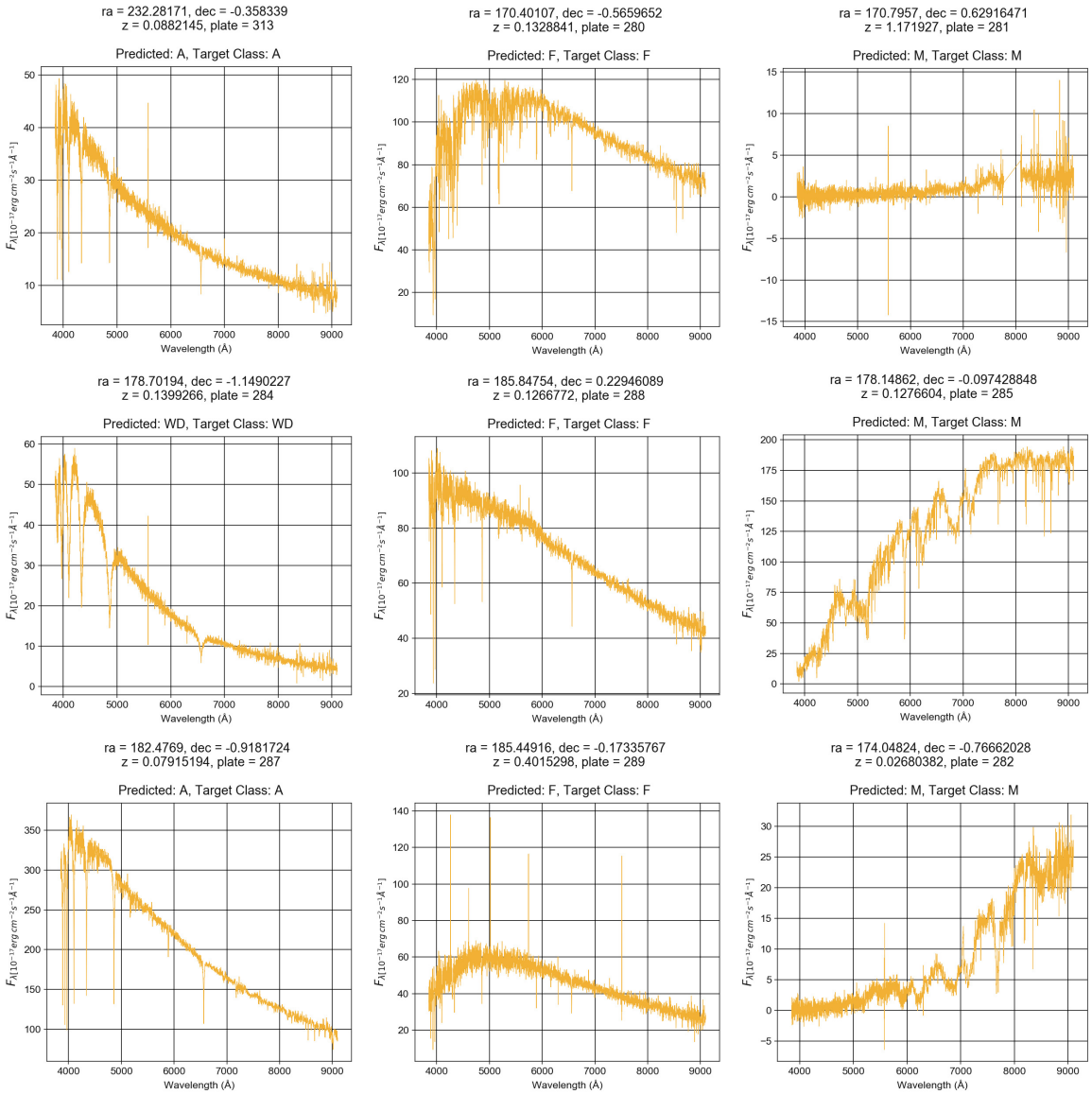



Figure 29: **Mixed-input Network - STAR classification** using 15 classes: Correct predictions of the model

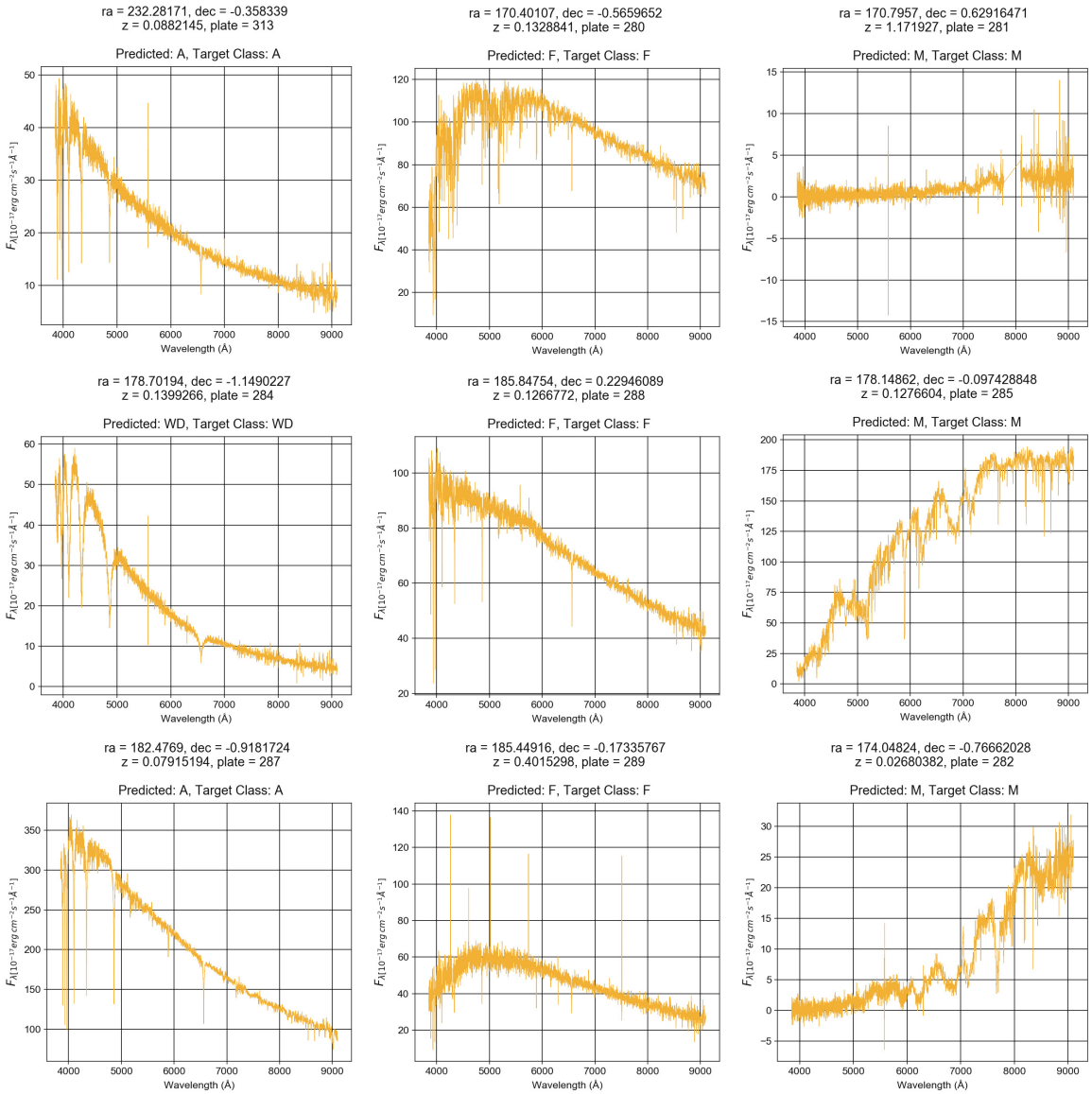



Figure 30: **Mixed-input Network - STAR classification** using 15 classes: Incorrect predictions of the model

# 8 Discussion

Classifying astronomical spectra with deep learning algorithms is a process requiring many steps and hurdles overcome. This project focued on querying, downloading, preprocessing and classifying astronomical objects based on their spectra using Python and deep neural networks of different architectures. I would like to spend a few lines reflecting on each of them.

## 8.1 Querying

Querying may have been the most straightforward out of the four. Using the `SDSS` class from astroquery.sdss one can query any table from SDSS. Using SDSS' schema browser I could find all data about the sources that may be relevant for classification as well as downloading spectra.

One issue that did not get enough attention early in the project was that when querying all sources without conditions, 83% of returned objects are galaxies and the most of the remaining objects are quasars. This introduced a large imbalance in the dataset that was hard to correct for at later stages. For the different types of classifications this project aimed at would have been optimal to have a uniform distribution of galaxies, quasars and stars and to have a uniform distribution among the subclasses as well. This can be solved with clever conditional querying that could be interesting for the continuation of this project.

## 8.2 Downloading

One of the most time consuming parts of the project was to download enough sources. To have an easier to reproduce project I included the querying as well as the downloading in a Python project which can be found on GitHub. However I'm not certain the downloading solution presented in this thesis is the fastest possible one. First of all, the method used here looped through coordinates from the result of the query and fetched a spectrum within 1 arcsecond radius. For a yet unknown reason this produced many duplicates that had to be deleted. Coupling this with the fact that it takes around 1-3 seconds to download each spectrum

and the occasional SDSS server errors had put an upper limit on what I could download in my available time.

## 8.3 Preprocessing

Be it storing and preprocessing data may not be the most interesting part of any machine learning project, it can often take up the most time of the development if one works with messy or hard-to-collect data. This project had gone through many different phases of data storing and processing to reach a point of being a relatively streamlined process in which the processing and handling of data flows efficiently from downloading to classification. One of the hurdles to overcome was the exploding file sizes that not only slowed saving and reading files, but slowed down other preprocessing steps and at points limited my available storage space. This was caused by multiple factors including saving to *pickle* instead of *HDF5*, storing fluxes in cells as arrays and leaving classes and subclasses as byte objects provided by SDSS as such.

Fixing these and creating a more or less chronological pipeline of data processing had greatly reduced the time needed to collect and store more sources as well as to run different experiments and try out different architectures.

## 8.4 Classification

All issues and inefficiencies described in the previous few sections compound in their effect on classification, which is the central focus of this project. This compounding is more prevalent at some of the classification tasks mentioned than in others. Namely galaxy and star subclass classification had been effected by data imbalances and in the case of galaxies: missing subclasses that had not been addressed. However I believe this is only partially the reason for less than optimal classification accuracies.

It has been traditionally difficult to draw hard cut boundaries between classes of astronomical objects. Nature doesn't always cluster objects in ways that can be easily distinguished by humans just looking at their light source. For example the

line between a quasar and an AGN host galaxy can be blurry. Another component was that signal-to-noise ratio varied across different sources some having very noise fluxes, which makes the object hard to classify regardless of what neural network architecture is used.

# 9 Conclusions

As I hope I made clear from the discussion, getting reasonable classification results in this field is not about grabbing data and pushing it through a neural network built with Keras. There are many obstacles to overcome at each step of the pipeline from downloading to classifying. Some of these obstacles come from the underlying physics (e.g. negative flux values and other artifacts), which one has to take into consideration when designing algorithms.

SDSS uses 33 spectral templates to classify objects (*SDSS Spectral Templates* n.d.), which is done by looping over and checking against possible templates to find the best fit for every classification. In the past template matching had worked relatively well, however this technique has been showing its age. Comparing spectra to each template to find the best fit is not performance efficient and can be inaccurate for spectra with low signal-to-noise ratio or ones that have missing values.

In contrast a neural network with the adequate architecture trained on large well processed datasets can offer a fast and accurate alternative to template matching. Performing a classification with a trained NN is basically a set of matrix multiplications, which had gotten extremely fast due to years of software and hardware optimizations and the advance of GPU technology.

Performance of classification is getting increasingly crucial as enormous data is being captures with a growing rate. This will compound as new telescopes are soon to be producing larger and larger spectroscopic surveys such as **4MOST** (*4MOST Website* n.d.), **SDSS-V**(*SDSS-V Website* n.d.), **WEAVE** (*WEAVE Website* n.d.) and **DESI** (*DESI Website* n.d.).

In this thesis I tried to show that even training on a small subset of SDSS' large collection of sources can yield good results in differentiating between galax-

ies, quasars and stars if one makes use of spectro-photometric features as well as spectra. While I laid out several issues I encountered during subclass classification of galaxies and stars, I see no real limitations in getting their performances to approach the one shown in mainclass classification once implementing the improvements suggested in previous chapters.

Throughout the project I used supervised learning because of an underlying assumption that the classes SDSS provides are to be trusted. Although I discovered that these classes are most often reliable, this is an assumption one does not have to make. Another approach to this problem is to turn to unsupervised learning algorithms and let them discover physical features in spectra that differentiates sources from each other most effectively. One very promising option I entertained is autoencoder neural networks also called encoder-decoder networks.

## 9.1 Autoencoders

If we think the properties of all objects in our sample are driven by a few physical parameters, we may derive an algorithm that captures this information. Autoencoders do this by mapping the large dimensional space of the input to a lower dimensional space by learning the representation that recovers the original input. This is done by first having layers of decreasing size until this bottleneck is reached, then expanding back to the size of the input with layers of increasing size in a symmetrical structure. In our case, it implements 1D convolutional layers and max pooling, similar to chapter 6. It has 3 main parts:

- **encoder**: a set of convolutional layers where each subsequent layer has fewer neurons than previous ones. First of these layers is the input of the network
- **code** or **latent space** also called **bottleneck**: the learnt lower dimensional representation of the data. This can be interesting to study, since if the network learned the representation well, it means that most of the relevant information about a source can be described with these set of numbers that must somehow relate to physical properties of the object.

- **decoder**: a set of convolutional layers where each subsequent layer is larger than previous ones. The aim of the decoder is to accurately reconstruct the input data from the code. Its last layer has the same dimensionality as the input layer.

When learning the autoencoder tries to reconstruct the input running through the bottleneck by minimizing the difference between input and output. If successful the reconstructed spectrum should look much like the original that was fed into the network. More formally an autoencoder is the combination of 2 functions

$$\begin{aligned} \phi &: X \to F \\ \psi &: F \to X \end{aligned} \tag{18}$$

where $\phi$ is the function that maps input $X$ to latent space $F$, and $\psi$ is the function that maps latent space $F$ to a input $X$. We want to minimize the difference between the input $X$ and the reconstructed input $\hat{X} = (\psi \circ \phi)X$, we can say

$$L\,(\,X, \hat{X}\,) = ||X - (\psi \circ \phi)X||^2 \tag{19}$$

Training the network is minimizing this loss function

$$\phi, \psi = \arg\min L \tag{20}$$

First I selected quasars and trained an autoencoder on 25,990 sources and tested it on 6,498. The autoencoder architecture used here has 896 neurons as its input layer, and 14 in its latent space. It would be interesting to see how sources form clusters in this 14 dimensional space and if the clases given by astronomers resemble the clusters created by the network. A pairwise plot of quasars in this 14 dimensional space is presented in Figure 34. It shows initial work in identifying possible clusters. For Figure 32 I picked outputs of the autoencoder that resemble the original input relatively well, and in Figure 33 there are some clear artifacts showing.

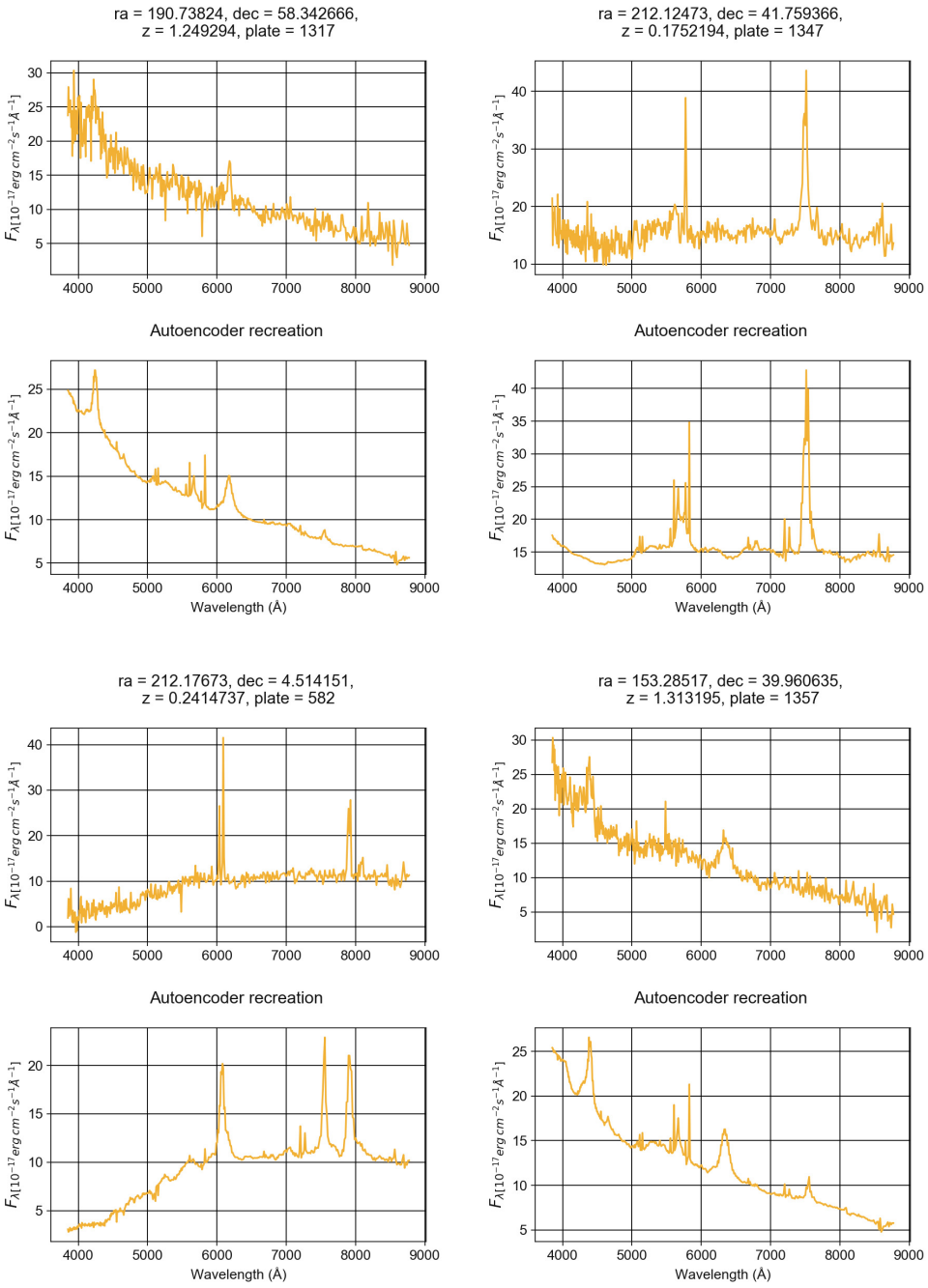



Figure 31: **Autoencoder**: Quasar spectra and their recreation using an autoencoder trained on 25,990 quasars

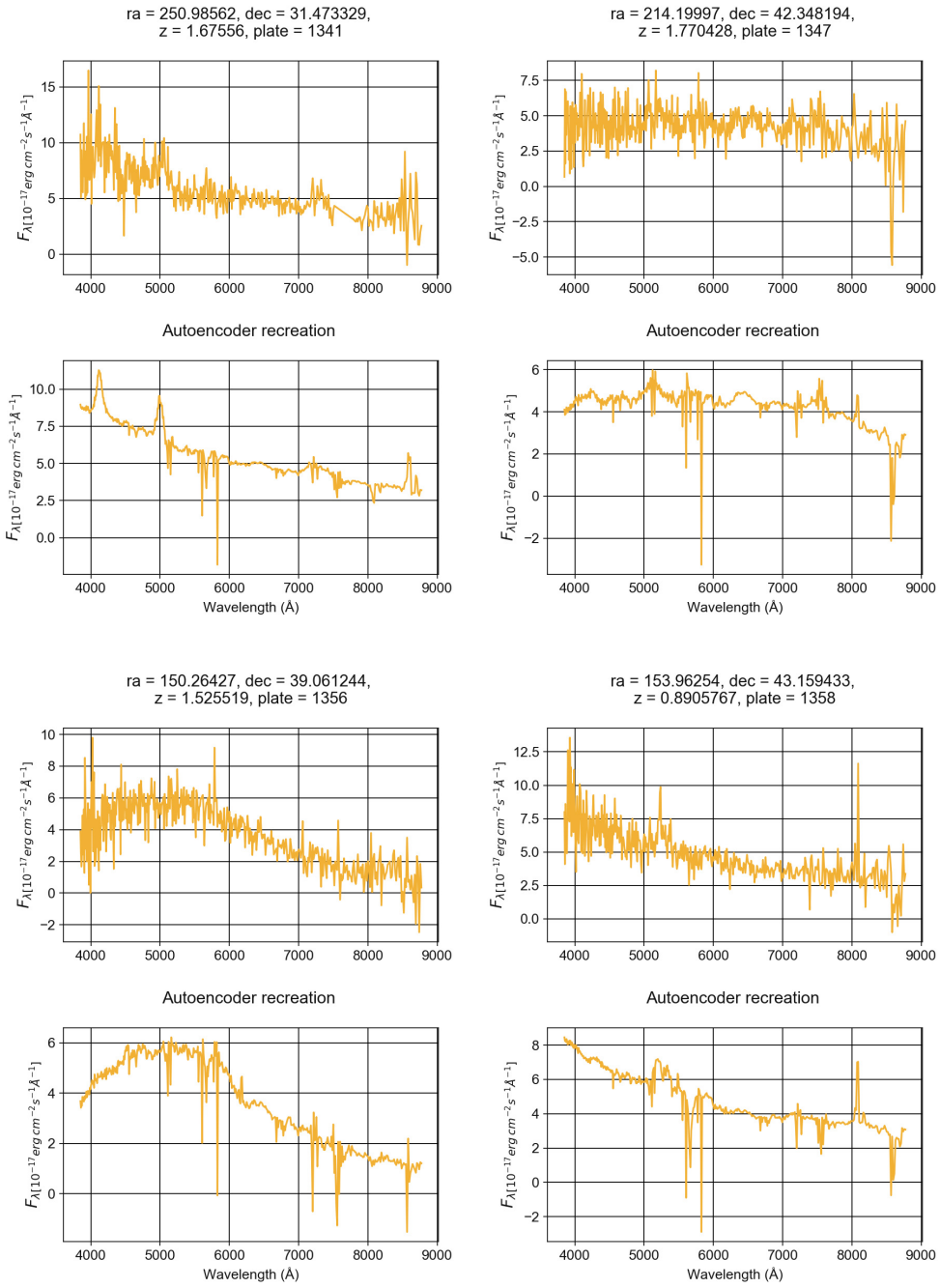



Figure 32: **Autoencoder**: Quasar spectra and their recreation using an autoencoder trained on 25,990 quasars

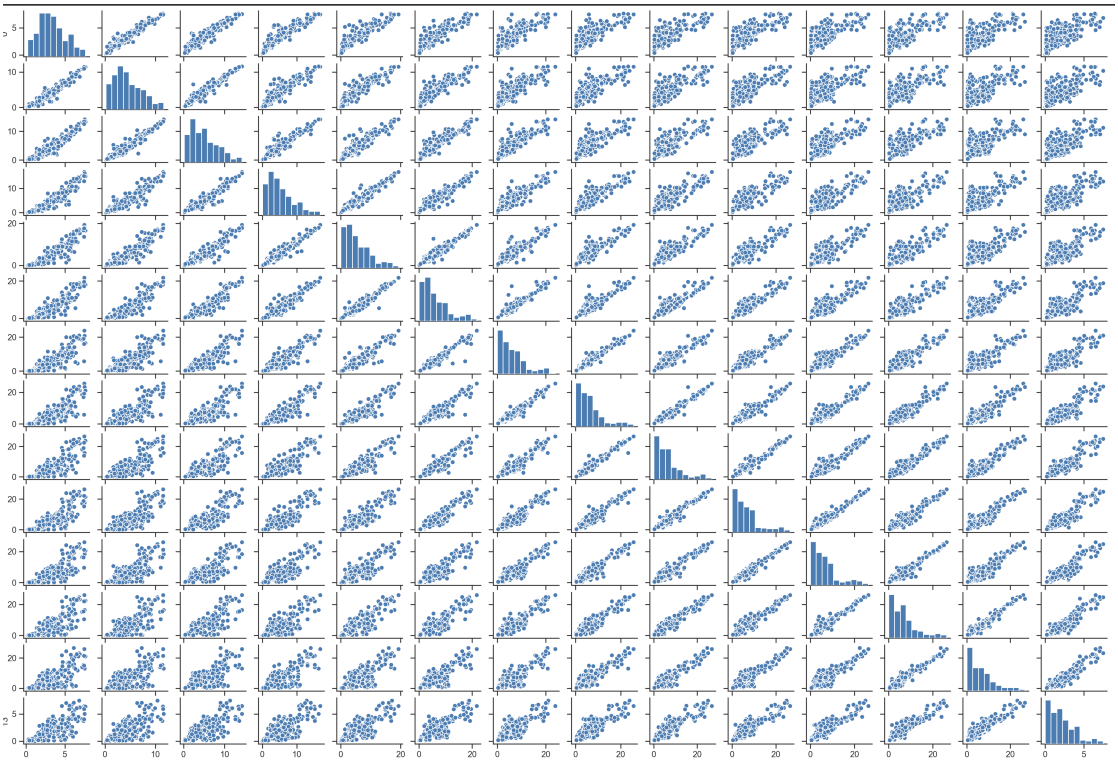


Figure 33: **Autoencoder**: Pairwise plot produced from sources in the 14 dimensional latent space using only quasars

For this next experiment I have increased the resolution of the input spectrum to **1792 dimensions**, while keeping the bottleneck layer at **14**. This model was trained for 50 epochs on 76,891 spectra from all three main classes. Figure 35 shows recreations of galaxies, Figure 36 shows quasars and Figure 37 shows stars. And finally Figure 38 presents another attempt at showing clustering in this 14 dimensional space. The following snippet of code was used to build the atuoencoder model. For the lower resolution spectra it was slightly different (Max Pooling exclusively with size 2). The GitHub link and the full code can be found in Appendix 10.5.

```
input_layer = Input(shape=(self.X_train.shape[1], 1))

# ENCODER #
x = Conv1D(filters=256, kernel_size=7, activation='relu', padding='same')(input_layer)
x = MaxPooling1D(4)(x)

x = Conv1D(filters=128, kernel_size=5, activation='relu', padding='same')(x)
x = MaxPooling1D(4)(x)

x = Conv1D(filters=64, kernel_size=5, activation='relu', padding='same')(x)
x = MaxPooling1D(2)(x)

x = Conv1D(filters=32, kernel_size=3, activation='relu', padding='same')(x)
x = MaxPooling1D(2)(x)

x = Conv1D(filters=32, kernel_size=3, activation='relu',padding='same')(x)
x = MaxPooling1D(2)(x)

x = Conv1D(filters=1, kernel_size=3, activation='relu', padding='same')(x)
encoded = MaxPooling1D(2, padding='same')(x)

# DECODER #

x = Conv1D(filters=1, kernel_size=3, activation='relu', padding='same')(encoded)
x = UpSampling1D(2)(x)

x = Conv1D(filters=32, kernel_size=3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)

x = Conv1D(filters=32, kernel_size=3, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)

x = Conv1D(filters=64, kernel_size=5, activation='relu', padding='same')(x)
x = UpSampling1D(2)(x)

x = Conv1D(filters=128, kernel_size=5, activation='relu', padding='same')(x)
x = UpSampling1D(4)(x)

x = Conv1D(filters=256, kernel_size=7, activation='relu', padding='same')(x)
x = UpSampling1D(4)(x)

decoded = Conv1D(1, 1, activation='tanh', padding='same')(x)

autoencoder = Model(input_layer, decoded)
autoencoder.compile(loss='mse', optimizer='adam')
```

By looking at the recreations produced by the autoencoder (Figure 35, 36 37) it becomes aparent what the shortcomings of the model are. In some occasions it adds absorption lines that were not present in the original source, for example the bottom left plot in Figure 35. At other times it mistakes narrow emission lines with broad ones (most likely an artifact of seeing broad emission lines of quasars). Overall even with these obvious errors it seems to be able to capture relevant information in the 14 dimensional latent space from which it can recreate the input. It would be interesting to experiment with different network architectures and resolutions as well as changing the dimensionality of the latent space.

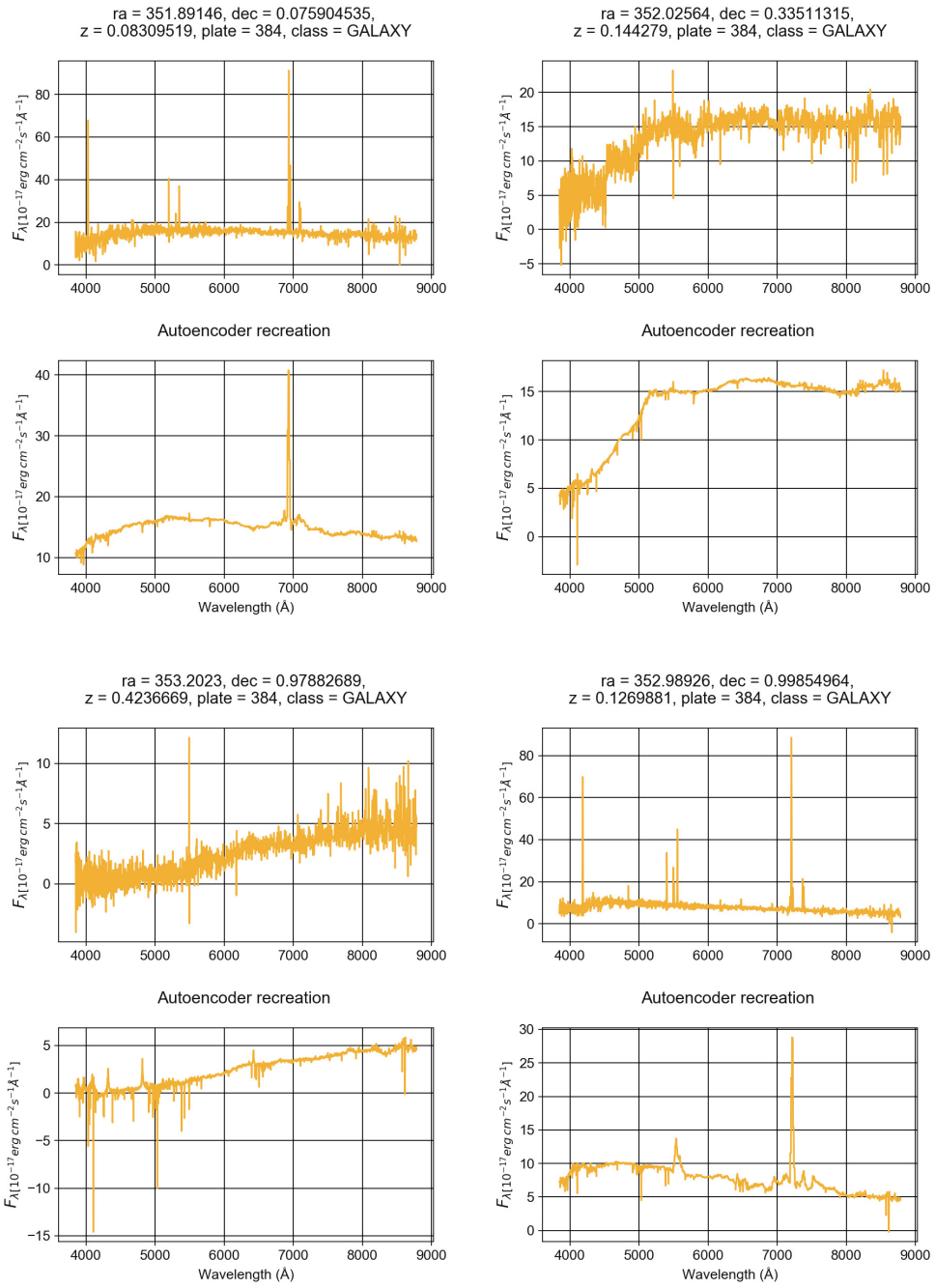



Figure 34: **Autoencoder**: Galaxy spectra and their recreation using an autoencoder trained on 76,891 sources

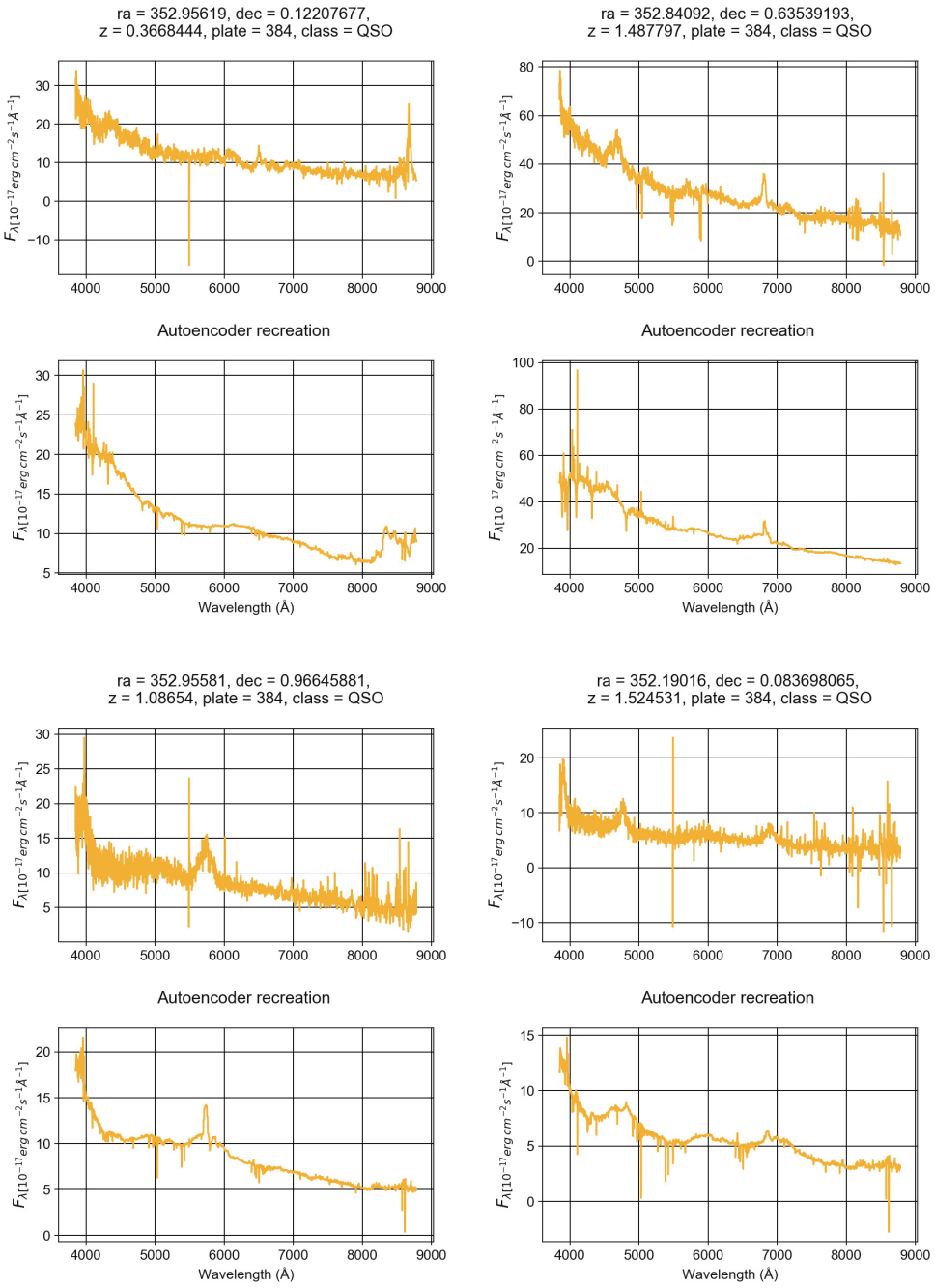



Figure 35: **Autoencoder**: Quasar spectra and their recreation using an autoencoder trained on 76,891 sources

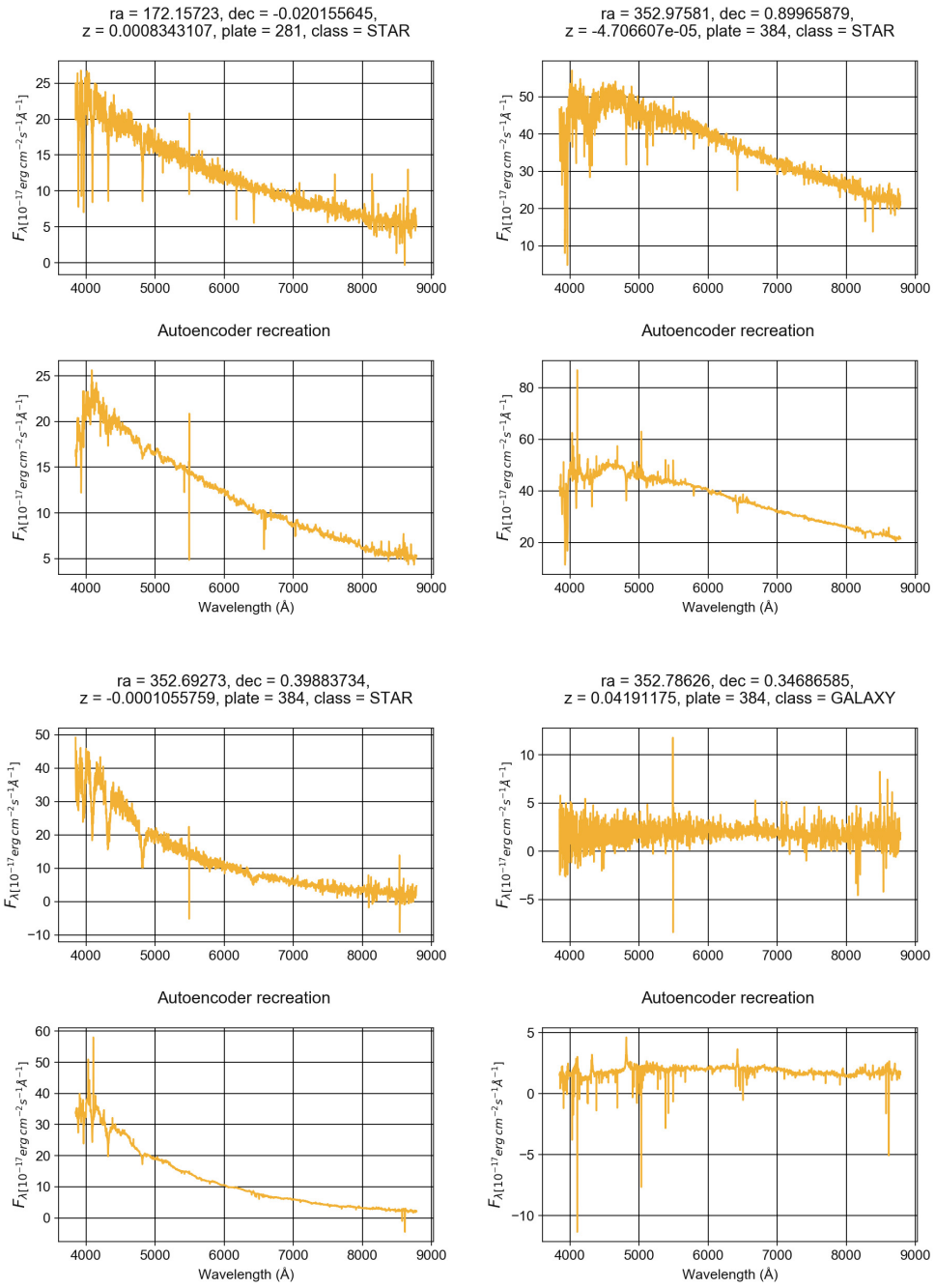



Figure 36: **Autoencoder**: Star spectra and their recreation using an autoencoder trained on 76,891 sourcess

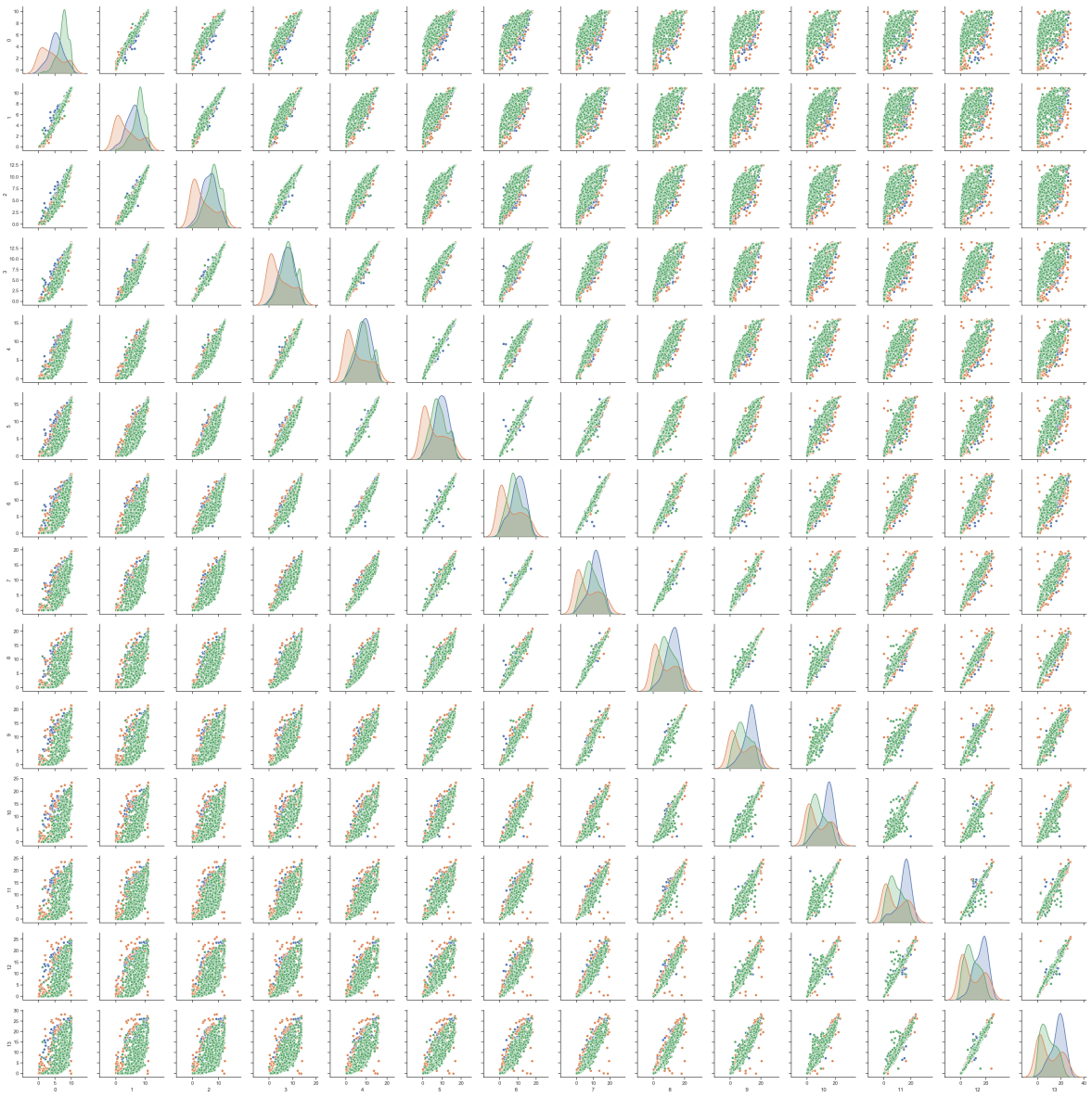



Figure 37: **Autoencoder**: Pairwise plot produced from sources in the 14 dimensional latent space using all sources (green - galaxy, blue - qso, yellow - star)

# 10 Appendix

Any code presented in the appendix may be subject to inconsistencies. I have changed and rewritten everything many times and have not had time to finalize and refactor. It should all make sense and the code is functional, but use of variable names or documentation might be inconsistent. There may be imports of functions from local files that are not provided here, but everything can be found at the link `https://github.com/csepreghy/spectral_analysis`.

## 10.1 Source Code on GitHub

`https://github.com/csepreghy/spectral_analysis`

## 10.2 Code for Querying SDSS

https://github.com/csepreghy/spectral_analysis/blob/master/spectral_analysis/downloading/sdss_direct_query.py

```
from astroquery.sdss import SDSS
import pandas as pd
import time as time

def truncate(n):
  return int(n * 1000) / 1000

def get_coordinates_from_query(save_metatable=False, save_coordinates=False, source_type=None):
    """
    get_coordinates_from_query()

    Downloads and saves into CSV a list of coordinates based on the SQL query written
    in the function

    Parameters
    ----------
    save_metatable : boolean
        When True, save the resulting DataFrame containing meta data into a pickle
        When False, don't save
    save_coordinates : boolean
        When True, save the coordinates to a CSV
        When False, don't save

    """
```

```
start = time.clock()

if source_type == 'QSO':
    query = "select \
                spec.z, spec.ra, spec.dec, spec.specObjID, spec.bestObjID, spec.fluxObjID, \
                spec.targetObjID, spec.plate, spec.class, spec.subClass, spec.zErr, \
                spho.petroMag_u, spho.petroMag_g, spho.petroMag_r, spho.petroMag_i, \
                spho.petroMag_z, spho.petroMagErr_u, spho.petroMagErr_g, spho.petroMagErr_r, \
                spho.petroMagErr_i, spho.petroMagErr_z \
                from SpecObjAll AS spec \
                JOIN SpecPhotoAll AS spho ON spec.specObjID = spho.specObjID \
                where \
                spec.zWarning = 0 AND spec.class = 'QSO'"

elif source_type == 'STAR':
    query = "select \
                spec.z, spec.ra, spec.dec, spec.specObjID, spec.bestObjID, spec.fluxObjID, \
                spec.targetObjID, spec.plate, spec.class, spec.subClass, spec.zErr, \
                spho.petroMag_u, spho.petroMag_g, spho.petroMag_r, spho.petroMag_i, \
                spho.petroMag_z, spho.petroMagErr_u, spho.petroMagErr_g, spho.petroMagErr_r, \
                spho.petroMagErr_i, spho.petroMagErr_z \
                from SpecObjAll AS spec \
                JOIN SpecPhotoAll AS spho ON spec.specObjID = spho.specObjID \
                where \
                spec.zWarning = 0 AND spec.class = 'STAR'"

else:
    query = "SELECT\
                spec.z, spec.ra, spec.dec, spec.specObjID, spec.bestObjID, spec.fluxObjID, \
                spec.targetObjID, spec.plate, spec.class, spec.subClass, spec.zErr, \
                spho.petroMag_u, spho.petroMag_g, spho.petroMag_r, spho.petroMag_i, \
                spho.petroMag_z, spho.petroMagErr_u, spho.petroMagErr_g, spho.petroMagErr_r, \
                spho.petroMagErr_i, spho.petroMagErr_z, \
                em.Flux_Hb_4861, em.Flux_Hb_4861_Err, em.Amplitude_Hb_4861, \
                em.Amplitude_Hb_4861_Err, em.Flux_OIII_4958, em.Flux_OIII_4958_Err, \
                em.Amplitude_OIII_4958, em.Amplitude_OIII_4958_Err, em.Flux_OIII_5006, \
                em.Flux_OIII_5006_Err, em.Amplitude_OIII_5006, em.Amplitude_OIII_5006_Err, \
                em.Flux_Ha_6562, em.Flux_Ha_6562_Err, em.Amplitude_Ha_6562, \
                em.Amplitude_Ha_6562_Err, em.Flux_NII_6547, em.Flux_NII_6547_Err, \
                em.Amplitude_NII_6547, em.Amplitude_NII_6547_Err, em.Flux_NII_6583, \
                em.Flux_NII_6583_Err, em.Amplitude_NII_6583, em.Amplitude_NII_6583_Err \
                FROM SpecObjAll AS spec \
                JOIN SpecPhotoAll AS spho ON spec.specObjID = spho.specObjID \
                JOIN emissionLinesPort AS em ON em.specObjID = spec.specObjID \
                WHERE \
                spec.zWarning = 0 AND spec.class = 'AGN'"

res = SDSS.query_sql(query, timeout=600)
df = res.to_pandas()
```

```
    print('df', df)
    print(df.columns)

    if save_metatable:
        df.to_pickle('data/sdss/qso_meta_table_emissionlines.pkl')

    end = time.clock()
    tt = end - start
    print("time consuming:", truncate(tt), 's')

def main():
        """
        main()

        Runs a test coordinate download.
        """
        print('sdss_direct_query.py -- __main__')

        get_coordinates_from_query(save_metatable=True, save_coordinates=False)

if __name__ == '__main__':
        main()
```

## 10.3 Code for Downloading SDSS Spectra

GitHub link

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import time as time
import pickle

from astropy.table import Table
from astropy import coordinates as coords
import astropy.units as u
from astroquery.sdss import SDSS

def download_spectra(coord_list_url, from_sp, to_sp, save=False):
    """
    download_spectra()

    Downloads SDSS spectra in a specified range based on a list of coordinates

    Parameters
    ----------
```

```
    coord_list_url: string
        The path for the CSV file that contains the list of coordinates
        that was downloaded using SQL, which provides coordinates for spectra
        to download. Contains 500,000 rows

    from_sp : int
        The index from which to download spectra. This enables us to download in
        batches and save the spectral data in batches
    to_sp : int
        The index which specifies the upper limit until which to download spectra.
    save : boolean
        When True, save the resulting DataFrame into a pickle
        When False, don't save

    Returns
    -------
    df: pandas.DataFrame
        The DataFrame that contains all downloaded spectral data.

        columns:        'flux_list',
                    'wavelength',
                    'z',
                    'ra',
                    'dec',
                    'objid'
    """
    t_start = time.clock()

    coord_list = pd.read_csv(filepath_or_buffer=coord_list_url)
    print(f'coord_list = {coord_list}')

    ra_list = coord_list["ra"].tolist()
    dec_list = coord_list["dec"].tolist()

    ra = ra_list[from_sp:to_sp]
    dec = dec_list[from_sp:to_sp]

    n_errors = 0

    df = {}
    df['flux_list'] = []
    df['wavelength'] = []
    df['z'] = []
    df['ra'] = []
    df['dec'] = []
    df['objid'] = []

    n_coordinates = len(ra)
    number_none = 0
```

```
    for i in range(n_coordinates):
        try:
            pos = coords.SkyCoord((ra[i]) * u.deg, (dec[i]) * u.deg, frame='icrs')
            xid = SDSS.query_region(pos, spectro=True) # radius=5 * u.arcsec)

            if xid == None:
                number_none = number_none + 1
                print('xid is None at:', i)
                continue

            elif xid != None and len(xid) > 1: xid = Table(xid[0])

            sp = SDSS.get_spectra(matches=xid)

            df['flux_list'].append(sp[0][1].data['flux'])
            df['wavelength'].append(10. ** sp[0][1].data['loglam'])
            df['z'].append(xid['z'])
            df['ra'].append(xid['ra'])
            df['dec'].append(xid['dec'])
            df['objid'].append(xid['objid'])

            print(f'Downloaded: {i}')

        except:
            print('Failed to download at:', i)
            n_errors = n_errors + 1

    df = pd.DataFrame(df)
    print('df.head()', df.head())
    if save:
        df.to_pickle('data/sdss/spectra/spectra_' + str(from_sp) + '-' + str(to_sp) + '.pkl')

    t_end = time.clock()

    t_delta = t_end - t_start
    n_downloads = len(ra) - 1
    print("time for " + str(n_downloads) + " stellar objects:", t_delta)

    print('DF After Downloading:')
    print(df.columns)
    print(df)
    print(f'Length of df = {len(df)}')

    return df

def main():
        """
        main()
```

```
        Runs a test batch download to test whether the download() works properly.
        """

        download_spectra(coord_list_url = "data/coord_list.csv",
                                            from_sp = 110001,
                                          to_sp = 120000,
                                        save=True)

if __name__ == '__main__':
          main()
```

## 10.4 Code for the CNN with hyperparameter optimization

GitHub link

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import History, EarlyStopping
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv1D, MaxPooling1D, Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.utils import to_categorical

from kerastuner.tuners import RandomSearch
from kerastuner.engine.hyperparameters import HyperParameters

import time

LOG_DIR = f"{int(time.time())}"

class CNN:
    def __init__(self, df_fluxes, max_trials, epochs, batch_size):
        self.input_length = len(df_fluxes.columns) - 1
        self.max_trials = max_trials
        self.epochs = epochs
        self.batch_size = batch_size

    def _prepare_data(self, df_source_info, df_fluxes):
        columns = []
```

```
        df_source_info['class'] = pd.Categorical(df_source_info['class'])
        df_dummies = pd.get_dummies(df_source_info['class'], prefix='category')
        df_dummies.columns = ['category_GALAXY', 'category_QSO', 'category_STAR']
        df_source_info = pd.concat([df_source_info, df_dummies], axis=1)

        for column in df_source_info.columns:
            if column not in ['class', 'dec', 'ra', 'plate', 'wavelength', 'objid', 'subClass']:
                columns.append(column)

        X = np.delete(df_fluxes.values, 0, axis=1)
        y = []

        print(f'df_source_info = {df_source_info}')

        for _, spectrum in df_source_info[columns].iterrows():
            category_GALAXY = spectrum["category_GALAXY"]
            category_QSO = spectrum["category_QSO"]
            category_STAR = spectrum["category_STAR"]

            y_row = [category_GALAXY, category_QSO, category_STAR]

            y.append(y_row)

        return X, y


    def _fit(self, X_train, y_train, X_test, y_test, X_val, y_val):
        tuner = RandomSearch(self._build_model,
                             objective='val_accuracy',
                             max_trials=self.max_trials,
                             executions_per_trial=1,
                             directory='logs/keras-tuner/',
                             project_name='cnn')

        tuner.search_space_summary()

        tuner.search(x=X_train,
                     y=y_train,
                     epochs=self.epochs,
                     batch_size=self.batch_size,
                     verbose=0,
                     validation_data=(X_val, y_val),
                     callbacks=[EarlyStopping('val_accuracy', patience=4)])
        print('kakkanat\n\n\n\n\n\n')
        print(tuner.results_summary())
        model = tuner.get_best_models(num_models=1)[0]
        print(model.summary())

        # Evaluate Best Model #
```

```
    _, train_acc = model.evaluate(X_train, y_train, verbose=0)
    _, test_acc = model.evaluate(X_test, y_test, verbose=0)
    print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

def _build_model(self, hp):
    HP = {
        'n_conv_layers': hp.Int('n_conv_layers', 2, 6),
        'conv_layer_1_filters': hp.Choice('conv_layer_1_filters',
                                          values=[64, 128, 256, 512],
                                          default=256),
        'conv_layer_1_kernel_size': hp.Choice('conv_layer_1_kernel_size', values=[3, 5, 7]),
        'n_dense_layers': hp.Int('n_dense_layers', 2, 6),
        'learning_rate': hp.Choice('learning_rate', values=[1e-3, 1e-4])
    }

    for i in range(HP['n_conv_layers'] - 1):
        i = i + 2
        if i < 4:
            HP[f'conv_layer_{i}_filters'] = hp.Choice(f'conv_layer_{i}_filters',
                                                     values=[64, 128, 256, 512],
                                                     default=256)
        else:
            HP[f'conv_layer_{i}_filters'] = hp.Choice(f'conv_layer_{i}_filters',
                                                     values=[64, 128, 256],
                                                     default=256)
        HP[f'conv_layer_{i}_kernel_size'] = hp.Choice(f'conv_layer_{i}_kernel_size',
                                                     values=[3, 5, 7])
    for i in range(HP['n_dense_layers'] - 1):
        i = i + 1
        HP[f'dense_layer_{i}_nodes'] = hp.Choice(f'dense_layer_{i}_nodes',
                                                values=[64, 128, 256, 512],
                                                default=256)
    model = Sequential()

    model.add(Conv1D(filters=HP['conv_layer_1_filters'],
                     kernel_size=HP['conv_layer_1_kernel_size'],
                     activation='relu',
                     input_shape=(self.input_length, 1)))
    model.add(Dropout(0.1))

    for i in range(HP['n_conv_layers'] - 1):
        i = i + 2
        model.add(Conv1D(filters=HP[f'conv_layer_{i}_filters'],
                         kernel_size=HP[f'conv_layer_{i}_kernel_size'],
                         activation='relu'))
        model.add(Dropout(0.1))
        if i < 5: model.add(MaxPooling1D(pool_size=hp.Int('max_pool_size', 1, 4)))

    model.add(Flatten())
```

```
        for i in range(HP['n_dense_layers'] - 1):
            i = i + 1
            model.add(Dense(HP[f'dense_layer_{i}_nodes']))
            model.add(Dropout(0.5))

        model.add(Dense(3, activation='softmax'))
        model.compile(loss='categorical_crossentropy',
                      optimizer=Adam(HP['learning_rate']),
                      metrics=['accuracy'])

        return model

    def run(self, df_source_info, df_fluxes):
        X, y = self._prepare_data(df_source_info, df_fluxes)

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2)

        scaler = StandardScaler()

        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
        X_val = scaler.transform(X_val)

        X_train = np.expand_dims(X_train, axis=2)
        X_test = np.expand_dims(X_test, axis=2)
        X_val = np.expand_dims(X_val, axis=2)

        y_train = np.array(y_train)
        y_test = np.array(y_test)
        y_val = np.array(y_val)

        self._fit(X_train, y_train, X_test, y_test, X_val, y_val)

def main():
    df_fluxes = pd.read_hdf('data/sdss/preprocessed/balanced_spectral_lines.h5',
                            key='fluxes')
    df_source_info = pd.read_hdf('data/sdss/preprocessed/balanced_spectral_lines.h5',
                                 key='source_info')

    cnn = CNN(df_fluxes, max_trials=40, epochs=12, batch_size=32)
    cnn.run(df_source_info, df_fluxes)

if __name__ == "__main__":
    main()
```

## 10.5 Code of the Autoencoder Model

GitHub link

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import time

from tensorflow.keras.layers import (Input,
                                     Dense,
                                     Flatten,
                                     Conv1D,
                                     MaxPooling1D,
                                     UpSampling1D,
                                     BatchNormalization,
                                     Reshape)
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import TensorBoard, History, EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam, Nadam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping

from kerastuner.engine.hyperparameters import HyperParameters
from kerastuner.tuners import RandomSearch

from sklearn.preprocessing import MinMaxScaler, StandardScaler

import seaborn as sns

from spectral_analysis.classifiers.neural_network.helper_functions import train_test_split
from spectral_analysis.plotify import Plotify

class AutoEncoder():
    def __init__(self, df_source_info, df_fluxes, df_wavelengths, load_model, weights_path=''):
        self.load_model = load_model
        self.weights_path = weights_path
        X = self._prepare_data(df_source_info, df_fluxes, df_wavelengths)
        indeces = list(range(len(X)))
        X_train, X_test, self.i_train, self.i_test = train_test_split(X, 0.2, indeces=indeces)
        X_train, X_val, self.i_train, self.i_val = train_test_split(X_train, 0.2, indeces=indeces)

        self.scaler = StandardScaler()
        X_train = self.scaler.fit_transform(X_train)
        X_test = self.scaler.transform(X_test)
        X_val = self.scaler.transform(X_val)

        self.X_train = np.expand_dims(X_train, axis=2)
        self.X_test = np.expand_dims(X_test, axis=2)
        self.X_val = np.expand_dims(X_val, axis=2)
```

```
def _prepare_data(self, df_source_info, df_fluxes, df_wavelengths):
    # self.df_source_info = df_source_info.loc[df_source_info['class'] == 'QSO']
    self.df_source_info = df_source_info
    self.objids = self.df_source_info['objid'].to_numpy()
    fluxes = df_fluxes.loc[df_fluxes['objid'].isin(self.objids)]

    X = np.delete(fluxes.values, 0, axis=1)
    X = X[:, 0::2]
    print(f'X.shape = {X.shape}')
    X = X[:, np.mod(np.arange(X[0].size),25)!=0]
    X = X[:,:1792]
    print(f'X.shape = {X.shape}')


    wavelengths = df_wavelengths.to_numpy()
    wavelengths = wavelengths[::2]
    self.wavelengths = wavelengths[0:1792]
    # plot_spectrum(X[0], wavelengths)
    return X

def build_model(self):
    # ================================================================================ #
    # ==================================== ENCODER ==================================== #
    # ================================================================================ #

    input_layer = Input(shape=(self.X_train.shape[1], 1))

    # encoder
    x = Conv1D(filters=256,
               kernel_size=7,
               activation='relu',
               padding='same')(input_layer)
    x = MaxPooling1D(4)(x)

    x = Conv1D(filters=128,
               kernel_size=5,
               activation='relu',
               padding='same')(x)

    x = MaxPooling1D(4)(x)
    x = Conv1D(filters=64,
               kernel_size=5,
               activation='relu',
               padding='same')(x)
    x = MaxPooling1D(2)(x)

    x = Conv1D(filters=32,
               kernel_size=3,
```

```
            activation='relu',
            padding='same')(x)
x = MaxPooling1D(2)(x)

x = Conv1D(filters=32,
            kernel_size=3,
            activation='relu',
            padding='same')(x)
x = MaxPooling1D(2)(x)

x = Conv1D(filters=1,
            kernel_size=3,
            activation='relu',
            padding='same')(x)

encoded = MaxPooling1D(2, padding='same')(x)

# ============================================================================= #
# ================================== DECODER ================================== #
# ============================================================================= #

x = Conv1D(filters=1,
            kernel_size=3,
            activation='relu',
            padding='same')(encoded)

x = UpSampling1D(2)(x)

x = Conv1D(filters=32,
            kernel_size=3,
            activation='relu',
            padding='same')(x)

x = UpSampling1D(2)(x)

x = Conv1D(filters=32,
            kernel_size=3,
            activation='relu',
            padding='same')(x)

x = UpSampling1D(2)(x)

x = Conv1D(filters=64,
            kernel_size=5,
            activation='relu',
            padding='same')(x)

x = UpSampling1D(2)(x)
```

```
        x = Conv1D(filters=128,
                   kernel_size=5,
                   activation='relu',
                   padding='same')(x)

        x = UpSampling1D(4)(x)

        x = Conv1D(filters=256,
                   kernel_size=7,
                   activation='relu',
                   padding='same')(x)
        x = UpSampling1D(4)(x)

        decoded = Conv1D(1, 1, activation='tanh', padding='same')(x)

        self.autoencoder = Model(input_layer, decoded)
        self.autoencoder.summary()
        self.autoencoder.compile(loss='mse', optimizer='adam')

        return self.autoencoder

    def train_model(self, epochs, batch_size=32):
        model = self.build_model()

        if self.load_model == False:
            modelcheckpoint = ModelCheckpoint(filepath='logs/1-14_autoencoder.epoch{epoch:02d}.h5',
                                              monitor='val_loss',
                                              save_best_only=True)

            history = model.fit(x=self.X_train,
                                y=self.X_train,
                                epochs=epochs,
                                batch_size=32,
                                validation_data=(self.X_val, self.X_val),
                                callbacks=[EarlyStopping('val_loss', patience=8), modelcheckpoint])

            self.evaluate_model(model)

        else:
            model.load_weights(self.weights_path)
            print(f'model = {model}')
            # self.evaluate_model(model)
            self.get_bottleneck_values(model)

        return model

    def get_bottleneck_values(self, model):
        bottleneck = model.get_layer('conv1d_5')
```

```
        extractor = Model(inputs=model.inputs, outputs=[bottleneck.output])
        features = extractor(self.X_test)
        features = np.squeeze(features, axis=2)

        df_source_info_test = pd.DataFrame({'class':
                                            self.df_source_info.iloc[self.i_test]['class'].values})

        print(f'df_source_info_test = {df_source_info_test}')

        df = pd.DataFrame(features)
        df = df.join(df_source_info_test)

        print(f'df = {df}')

        sns.set(style="ticks", color_codes=True)
        sns.pairplot(df, hue='class')
        plt.savefig('plots/autoencoder_pairplot', dpi=100)

    def evaluate_model(self, model):
        preds = model.predict(self.X_test)

        print(self.X_test.shape)
        self.X_test = np.squeeze(self.X_test, axis=2)
        preds = np.squeeze(preds, axis=2)
        print(self.X_test.shape)

        self.X_test = self.scaler.inverse_transform(self.X_test)
        preds = self.scaler.inverse_transform(preds)

        for i in range(100):
            qso_ra = self.df_source_info.iloc[self.i_test[i]]['ra']
            qso_dec = self.df_source_info.iloc[self.i_test[i]]['dec']
            qso_plate = self.df_source_info.iloc[self.i_test[i]]['plate']
            qso_z = self.df_source_info.iloc[self.i_test[i]]['z']
            qso_class = self.df_source_info.iloc[self.i_test[i]]['class']

            plotify = Plotify(theme='ugly')
            _, axs = plotify.get_figax(nrows=2, figsize=(5.8, 8))
            axs[0].plot(self.wavelengths, self.X_test[i], color=plotify.c_orange)
            axs[1].plot(self.wavelengths, preds[i], color=plotify.c_orange)
            axs[0].set_title(f'ra = {qso_ra}, dec = {qso_dec}, \n \
                             z = {qso_z}, plate = {qso_plate}, class = {qso_class} \n', \
                             fontsize=14)
            axs[1].set_title(f'Autoencoder recreation \n')
            axs[0].set_ylabel(r'$F_{\lambda[10^{-17} erg \: cm^{-2}s^{-1} Å^{-1}]}$', fontsize=14)
            axs[1].set_ylabel(r'$F_{\lambda[10^{-17} erg \: cm^{-2}s^{-1} Å^{-1}]}$', fontsize=14)
            axs[1].set_xlabel('Wavelength (Å)')

            plt.subplots_adjust(hspace=0.4)
```

```
            plt.savefig(f'plots/autoencoder/__all_sources/_autoencoder_{i}', dpi=160)

        return preds

def main():
    df_fluxes = pd.read_hdf('data/sdss/preprocessed/balanced.h5', key='fluxes')
    df_source_info = pd.read_hdf('data/sdss/preprocessed/balanced.h5', key='source_info')
    df_wavelengths = pd.read_hdf('data/sdss/preprocessed/balanced.h5', key='wavelengths')

    ae = AutoEncoder(df_source_info,
                     df_fluxes,
                     df_wavelengths,
                     load_model=False,
                     weights_path='logs/colab-logs/_all_sources1-14_autoencoder.epoch30.h5')
    ae.train_model(epochs=12, batch_size=64)


if __name__ == "__main__":
    main()
```

# References


[] *4MOST Website*. URL: https://www.4most.eu/cms/. (accessed: 01.07.2020).

[] *DESI Website*. URL: https://www.desi.lbl.gov/. (accessed: 01.07.2020).

[] *SDSS Spectral Templates*. URL: http://classic.sdss.org/dr5/algorithms/spectemplates/. (accessed: 01.07.2020).

[] *SDSS-V Website*. URL: https://www.sdss.org/future/. (accessed: 01.07.2020).

[] *WEAVE Website*. URL: http://casu.ast.cam.ac.uk/surveys-projects/weave. (accessed: 01.07.2020).

[Bai+18] Yu Bai et al. 'Machine Learning Applied to Star–Galaxy–QSO Classification and Stellar Effective Temperature Regression'. In: *The Astronomical Journal* 157.1 (Dec. 2018), p. 9. DOI: 10.3847/1538-3881/aaf009. URL: https://doi.org/10.3847/1538-3881/aaf009.

[Bai01] C. Bailer-Jones. 'Automated stellar classification for large surveys: a review of methods and results'. In: (Mar. 2001).

[Che13] Fuqiang Chen. 'Spectral Classification Using Restricted Boltzmann Machine'. In: *CoRR* abs/1305.0665 (2013). arXiv: 1305.0665. URL: http://arxiv.org/abs/1305.0665.

[Con+16] Christopher J. Conselice et al. 'THE EVOLUTION OF GALAXY NUMBER DENSITY ATz 8 AND ITS IMPLICATIONS'. In: *The Astrophysical Journal* 830.2 (Oct. 2016), p. 83. DOI: 10.3847/0004-637x/830/2/83. URL: https://doi.org/10.3847%2F0004-637x%2F830%2F2%2F83.

[DHS11] John Duchi, Elad Hazan and Yoram Singer. 'Adaptive subgradient methods for online learning and stochastic optimization.' In: *Journal of machine learning research* 12.7 (2011).

[DIs+16] A. D'Isanto et al. 'An analysis of feature relevance in the classification of astronomical transients with machine learning methods'. In: *Monthly Notices of the Royal Astronomical Society* 457.3 (Feb. 2016), pp. 3119–3132. ISSN: 0035-8711. DOI: 10.1093/mnras/stw157. eprint: https://academic.oup.com/mnras/article-pdf/457/3/3119/8001547/stw157.pdf. URL: https://doi.org/10.1093/mnras/stw157.

[EBS08] C. Elting, C. A. L. Bailer-Jones and K. W. Smith. 'Photometric Classification of Stars, Galaxies and Quasars in the Sloan Digital Sky Survey DR6 Using Support Vector Machines'. In: *AIP Conference Proceedings* 1082.1 (2008), pp. 9–14. DOI: 10.1063/1.3059095. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.3059095. URL: https://aip.scitation.org/doi/abs/10.1063/1.3059095.

[Gun+06] James E. Gunn et al. 'The 2.5 m Telescope of the Sloan Digital Sky Survey'. In: *The Astronomical Journal* 131.4 (Apr. 2006), pp. 2332–2359. DOI: 10.1086/500975. URL: https://doi.org/10.1086/500975.

[KB14] Diederik P Kingma and Jimmy Ba. 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980* (2014).

[Mit+97a] Tom M Mitchell et al. *Machine learning*. 1997.

[Mit+97b] Tom M Mitchell et al. 'Machine learning'. In: McGraw-hill New York, 1997. Chap. 4.

[MP43] Warren S McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MS63] Thomas A. Matthews and Allan R. Sandage. 'Optical Identification of 3c 48, 3c 196, and 3c 286 with Stellar Objects.' In: *The Astrophysical Journal* 138 (July 1963), p. 30. DOI: 10.1086/147615. URL: https://doi.org/10.1086/147615.

[OA14] I. Ordovás-Pascual and J. Sánchez Almeida. 'A fast version of thekmeans classification algorithm for astronomical applications'. In: *Astronomy & Astrophysics* 565 (May 2014), A53. DOI: 10.1051/0004-6361/201423806. URL: https://doi.org/10.1051/0004-6361/201423806.

[Rud16] Sebastian Ruder. 'An overview of gradient descent optimization algorithms'. In: *arXiv preprint arXiv:1609.04747* (2016).

[SC94] L. Sodré and H. Cuevas. 'Spectral classification of galaxies'. In: *Vistas in Astronomy* 38 (Jan. 1994), pp. 287–291. DOI: 10.1016/0083-6656(94)90039-6. URL: https://doi.org/10.1016/0083-6656(94)90039-6.

[Shi99] Gregory A. Shields. 'A Brief History of Active Galactic Nuclei'. In: *Publications of the Astronomical Society of the Pacific* 111.760 (June 1999), pp. 661–678. DOI: 10.1086/316378. URL: https://doi.org/10.1086/316378.

[SS19] Siddharth Sharma and Ruchi Sharma. 'Classification of Astronomical Objects Using Various Machine Learning Techniques'. In: *Lecture Notes in Electrical Engineering*. Springer Singapore, Dec. 2019, pp. 275–283. DOI: 10.1007/978-981-15-0372-6_21. URL: https://doi.org/10.1007/978-981-15-0372-6_21.

[Unw+06] Stephen Unwin et al. 'Taking the Measure of the Universe: Precision Astrometry with SIM Planetquest (Preprint)'. In: (Oct. 2006), p. 49.

[Viq+18] Mohammed Viquar et al. 'Machine Learning in Astronomy: A Case Study in Quasar-Star Classification'. In: (Apr. 2018).

[WGL16] Ke Wang, Ping Guo and A-Li Luo. 'A new automated spectral feature extraction method and its application in spectral classification and defective spectra recovery'. In: *Monthly Notices of the Royal Astronomical Society* 465.4 (Dec. 2016), pp. 4311–4324. DOI: 10.1093/mnras/stw2894. URL: https://doi.org/10.1093/mnras/stw2894.

[Zha+12] Yanxia Zhang et al. 'Classification of Quasars and Stars by Supervised and Unsupervised Methods'. In: *Proceedings of the International Astronomical Union* 8.S288 (Aug. 2012), pp. 333–334. DOI: 10.1017/s1743921312017176. URL: https://doi.org/10.1017/s1743921312017176.

[ZZW95] Dennis Zaritsky, Ann I. Zabludoff and Jeffrey A. Willick. 'Spectral Classification of Galaxies Along the Hubble Sequence'. In: *The Astronomical Journal* 110 (Oct. 1995), p. 1602. DOI: 10.1086/117634. URL: https://doi.org/10.1086/117634.