Classification of crystal structure based on XRD patterns using deep learning

By

Zhaotong Du

Submitted in Partial Fulfillment of the

Requirements of the Degree

Master of Science

Supervised by Professor Niaz Abdolrahim

Materials Science Program

Arts, Sciences and Engineering

Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester

Rochester, New York

2022

Dedication

To my families and friends for their love and support.

Table of Contents

Biographical Sketch

The author, Zhaotong Du, was born in Beijing, China. He graduated with a Bachelor of Science degree in Mechanical Engineering from Beijing Institute of Technology. He came to the University of Rochester in 2020. He pursued a Master's thesis track under the instruction of Professor Niaz Abdolrahim.

Acknowledgements

The completion of this thesis would not be possible without the help and support from my advisor, colleagues, families, and friends. I would like to thank my advisor, Professor Niaz Abdolrahim, for her understanding, guidance, mentorship and discussion during my research. I would like to thank Professor Chenliang Xu for his guidance on computer science during the research. Thank you to Jerardo Salgado and Sam Lerman for their co-working, accompany, discussion and advice. Thank you to all members of the Advanced Computational Mechanics and Materials Laboratory. Finally, thank you to the University of Rochester, the department of Mechanical Engineering and Sarah Ansini for your two years of administrative support.

Abstract

Deformation behaviors of materials under extreme condition have attracted interest from material scientists. In order to reveal microscopic material structure changes, X-ray diffraction method are widely used. Traditional method of decoding a XRD pattern relies on software of programmed algorithms that based on first principle and Rietveld refinement, where correct peak indexing and expertise are vital. However, in the last decade, machining learning and artificial intelligence technology is developing rapidly with the raise of computational capabilities. One of the most common methods is deep learning, which is based on artificial neural networks. Classification task of deep learning can learn and assign a label to inputs from the problem domain.

This thesis applies deep learning to identify material structures, where crystal symmetries are assigned as labels and full XRD patterns as input features. First, we simulate ideal XRD patterns from large number of known structures with known symmetries from ICSD database. Then we train deep neural network models to learn from all calculated XRD patterns and their symmetry labels. The model results demonstrate that deep learning can learn to decode XRD patterns to the structure symmetries without any material science insights or programmed algorithms. Moreover, we apply the models on real world experimental XRD patterns from RRUFF database, and achieved 77% on 7-way crystal system classification, and 63.5% on 230-way space group classification.

Contributors and Funding Sources

This thesis was supported by a dissertation committee consisting of Professor Niaz Abdolrahim (advisor) and Professor Ryan Rygg of the Department of Mechanical Engineering, as well as Professor Chenliang Xu of the Department of Computer Science.

List of tables

List of figures

Chapter 1 Introduction

1.1 Motivation

In situ X-ray diffraction experiments[1-3] have been used to study the deformation behaviors of materials under extreme conditions. Microscopically, the deformation behaviors are changes and transformations of atom arrangements. Crystalline materials follow certain symmetric atoms arrangements within periodic unit cells. To describe the symmetries, crystallography termed 7 crystal systems to group unit cell parameters, and 230 space groups for sets of symmetries of atoms' spatial positions. Powder X-ray diffraction (XRD) has been the principal method for determining the arrangement of atoms. Powder XRD patterns encode 3D spatial electron-density distribution to 1D array patterns of various peaks. To decode XRD patterns secretes, traditional methods of analyzing those XRDs are based on first principal method and Rietveld refinement method that requires correct peak indexing of material science experts and time efforts[4]. It is even harder to identify low-symmetry atom arrangements with powder XRD patterns, since peaks can be overlapped and hard to separate and index. However, as computer science develops, one cutting edge method to rapidly identify the symmetries is to apply deep learning to powder XRD patterns and to classify their symmetries automatically without human interference[5-13].

1.2 Background

1.2.1 Crystallography

Crystal stands for ideal solid materials that have periodic structures, where a unit cell can propagate along one, two or three dimensions. To fully describe a three-dimensional unit cell, a total of three non-coplanar vectors are required. The unit cell thus can be completely described by a total of six scalar quantities:

$$a, b, c, \alpha, \beta, \gamma$$

And as shown schematically in Figure 1.1.

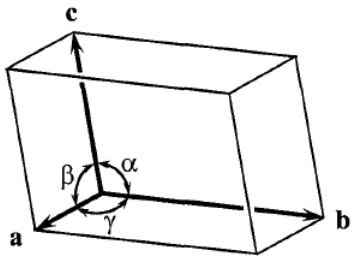

Figure 1.1 Unit cell in three-dimensions.

Usually, unit cell contains more than one molecule or a group of atoms as shown in Figure 1.2. As they have spatial distribution, these atoms can be converted into each other by simple geometrical transformations, which are called symmetry operations. There are four simple symmetry operations, rotation, inversion, reflection and translation. Based on valid combinations of those symmetry operations, finite groups can be assembled, and mathematical theory of groups are applied to crystal symmetry groups, termed group theory. Here we keep those well-developed theories short and only bring up the results of different groups of symmetries. Here shows an example of

the arrangement of atoms in unit cell, where different colors represent different atoms, and some atoms are following symmetries.

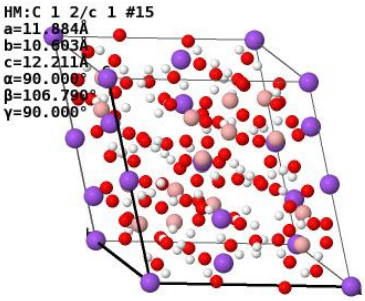

Figure 1.2 Crystal structure of a unit cell of Borax (sodium tetraborate decahydrate). Visualized by ICSD WEB[17,18].

All possible three-dimensional crystallographic point groups can be divided into 7 crystal systems based on specific symmetry: Triclinic, Monoclinic, Orthorhombic, Trigonal, Tetragonal, Hexagonal, and Cubic. Similarly, crystallographic space groups can also be defined by combining symmetry elements plus allowed translations. Limited by allowed rotations and translations, there is a total of 230 three-dimensional space groups, and they are arranged according to seven crystal systems, as shown in Table 1-1.

7 Crystal systems	230 space groups	Unit cell parameters
Triclinic	1-2	$a \neq b \neq c; \alpha \neq \beta \neq \gamma \neq 90°$
Monoclinic	3-15	$a \neq b \neq c; \alpha = \gamma = 90°, \beta \neq 90°$
Orthorhombic	16-74	$a \neq b \neq c; \alpha = \beta = \gamma = 90°$
Tetragonal	75-142	$a = b \neq c; \alpha = \beta = \gamma = 90°$
Trigonal	143-167	$a \neq b \neq c; \alpha = \beta = 90°, \gamma = 120°$
Hexagonal	168-194	$a \neq b \neq c; \alpha = \beta = 90°, \gamma = 120°$

Cubic	195-230	$a = b = c; \alpha = \beta = \gamma = 90°$

Table 1-1 Crystal systems and space groups

Crystallographic plane is mathematical abstraction for understanding diffraction of the next section. Crystallographic planes are defined as a family of planes that intersect all lattice points. All planes in the family are parallel to each other and are equally spaced. Three integers are used to describe the family planes, and are called Miller indices:

$$h, k, l$$

Miller indices indicate that the planes divided unit cell edges a, b, and c, to h, k, and l equal parts. For example, Figure 1.3 shows planes that cut the unit cell at a: to 1 part, b: to 1 part, and c: not cutting, which gives (hkl) as (110). For a determined unit cell, for a set of (hkl) indices, we can find single answer for planar distance.

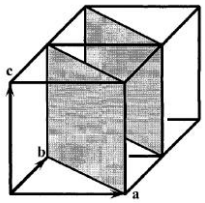

Figure 1.3 Family of (110) crystallographic planes[19].

1.2.2 X-ray diffraction

X-rays are an electromagnetic radiation consists of electric and magnetic vectors, and have wavelengths from ~0.1 to 100 Angstroms. The most commonly used in

crystallography is between ~0.5 and 2.5 angstroms, as this is the order of magnitude of interatomic distances. X-rays are generated from x-ray tubes, and are received by detectors.

Electrons surrounding atoms can be considered as charged particles, and can interact with electromagnetic waves. The oscillating electric field from X-ray exerts forces on electrons, forcing the electrons to oscillate with the same frequency. Thus, the oscillating electrons emits electromagnetic radiation in all directions. And this type of scattering, where incident wave and scattering wave shares the same frequency and wavelength, is called coherent scattering.

Braggs discovered that diffraction from a crystal can be explained and visualized by using mirror reflection of a series of crystallographic planes. Those planes, described by Miller indices, are parallel and equally spaced. Two parallel propagated in-phases waves can result in a new wave with doubled amplitude, which is called constructive interference. Braggs' law describes relationship between the interplanar distance, the wavelength of incident waves and the incident angle as shown in Figure 1.4.

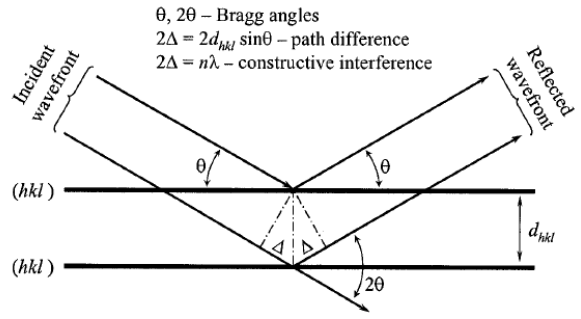

Figure 1.4 Illustration of Braggs' law[19].

Powder XRD is widely used for the identification of unknown crystalline materials. Each powder diffraction pattern can be characterized by their unique distribution of positions and intensities of Bragg peaks, where peak positions are defined by the unit cell dimensions and intensities are defined by the electron-density distribution (atom distribution) in the unit cell. An example of powder XRD pattern of borax is shown in Figure 1.5.

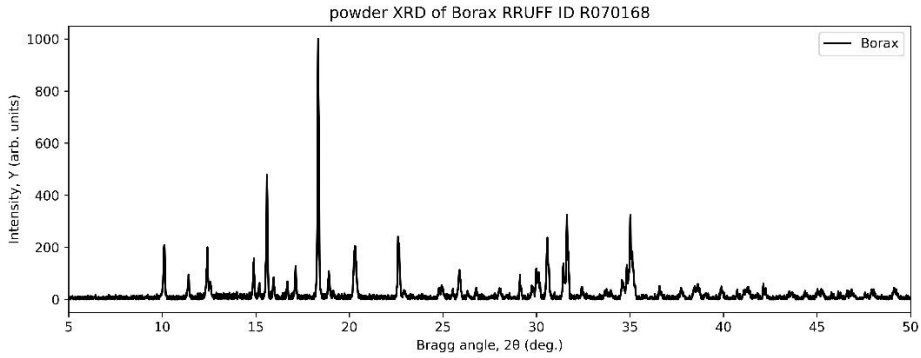

Figure 1.5 An example of powder XRD pattern of Borax from RRUFF database[21].

The instruments to obtain above patterns can be simplified as figure shown below. The change of incident angle is achieved by synchronized rotation of both source and detector arms. A schematic of simple diffractometer is shown in Figure 1.6.

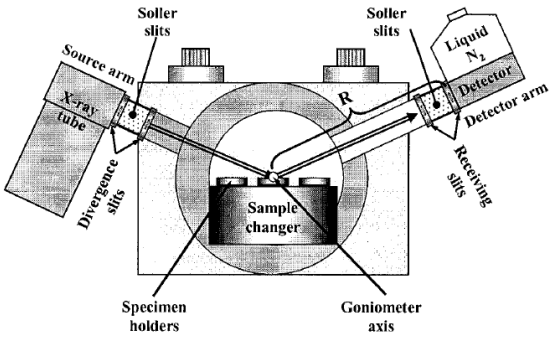

Figure 1.6 Schematic of a powder diffractometer[19].

1.2.3 Deep learning

Deep learning is a machine learning algorithm that use multiple layers to progressively extract higher-level features from the inputs. The most common field of deep learning, for example image processing, can only identify edges with lower layers, but can identify digits, letters or faces with higher layers[20]. Deep learning is a powerful set of techniques for learning in neural networks and it has proved to be a promising and effective tool that outperforms traditional rule-based methods in many areas, such as image classification, pattern recognition, speech recognition and natural language processing[4].

Deep neural network (DNN) is an artificial neural network with multiple layers between the input and output layers. Figure 1.7 shows an example of DNN architecture. The leftmost layer in this network is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons. The middle layers are called a hidden layer.

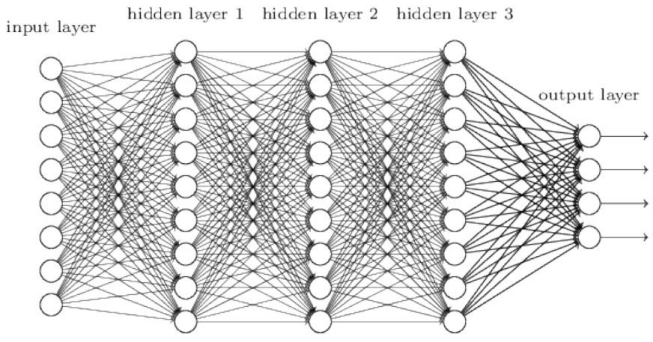

Figure 1.7 An example of architecture of deep neural network (DNN)[20].

DNN architecture doesn't take into account the spatial information. It treats input

features which are far apart and close together on exactly the same footing. Convolutional neural network (CNN) can learn spatial features. CNN has three additional algorithms.

1) Local receptive field: Each neuron will be connected to a small region. This region is the local receptive field. Each connection learns a weight. Then the local receptive field is slide across the entire layer as shown in Figure 1.8.

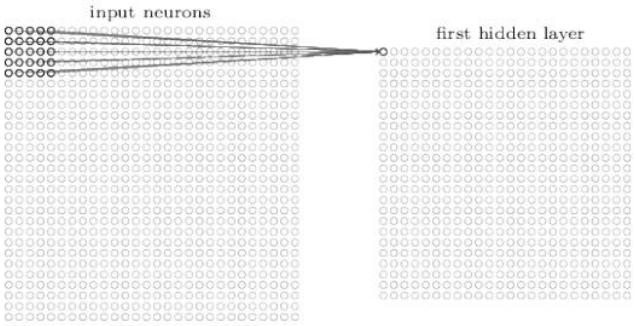

Figure 1.8 An example of convolutional neural network (CNN) local receptive field[20].

2) Shared weights and biases: Same weights and bias are used for each of the hidden neurons, which means all neurons in one hidden layer detect exactly the same features, at different locations in the inputs. For example, if the local receptive field can pick up a vertical edge, it can also identify edges at other places in the inputs. Thus, CNNs are well adapted to the translation invariance. The shared weights and bias are also called kernel or filter.
3) Pooling layers: Pooling layers are used immediately after convolutional layers. Pooling layers simplify the information in the output from convolutional layer. Commonly used pooling layers are max pooling layers, where the maximum value of the field is the output. Pooling layers of 2 times 2 can reduce the dimensions of inputs to one quarter (half in one dimension) as shown in Figure 1.9.

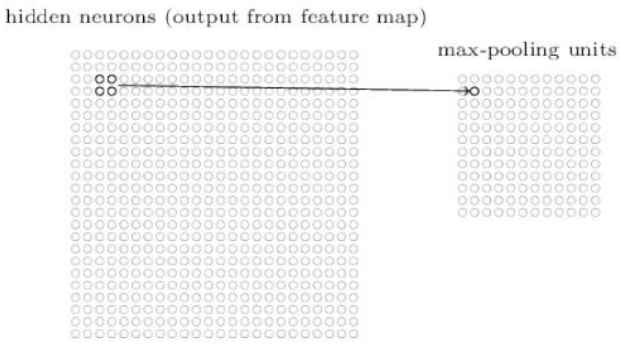

Figure 1.9 An example of max pooling layer in CNN[20].

Combining all above additional algorithms, a complete CNN is formed shown in below. Inputs are provided to local receptive field, and end up with 3 hidden feature neurons. Then, a max pooling is applied to reduce the complexity. Finally, fully connected layer connects every neuron from the max pooling layer, to the output layers via previously introduced DNN architecture. The major difference between DNN and CNN, is that CNN is made of many simple kernels or filters, that behaves different weights and biases.

1.2.4 Research Status

Park et al introduced three CNN models trained for classifying crystal systems, extinction groups and space groups[4]. The CNN architecture consists 3 pairs of convolutional and average pooling layer, and following by 2 fully connected layers. ReLu activation function is used, and coupled with a dropout of 0.3 in all convolutional and pooling layers and fully connected layers. They trained their model with 150,000 XRD patterns calculated from 150,000 cleaned ICSD structure solution data, using a

combination of fix parameters and random selected parameters. The XRD patterns are calculated from 10° to 110°(both are included) with a step of 0.01°, which leads to input layer length of 10001. A total of 3 models with different length of fully connected layers are trained for 3 different classification tasks. The training of the CNNs is based on 80% of total dataset. The rest 20% are used to test the performance. Testing accuracy reached 81%, 83% and 95% for space groups, extinction groups and crystal systems classifications. They tested their model on two novel materials and correctly predicted the crystal systems, but failed to predict their space groups.

Ziletti et al reported an image classification based on 2D diffraction image[5]. They defined their unique way of representing the structure using a 2D diffraction fingerprint. They demonstrated a classification probability change between BCC and simple cubic. They obtained an accuracy of 100% on both training and testing set. After introducing random defects by vacancies, the model remains 100% for up to 40% vacancies, which shows strong robustness for this classification task.

Vecsei et al introduced a comparison between DNN and CNN trained by over 100,000 patterns from ICSD[6]. Their DNN reaches 54% on experimental pattern from RRUFF. During their data preparation, they added several patterns of noises to simulate the experimental behavior. They also found out that DNN models performs better that CNN models. After introducing a scheme to reject large uncertainty prediction, they finally reach 82% on space group classifications at cost of rejecting half of predictions. For

their CNN architecture, they use average pooling and large kernel size. For DNN architecture, they use dropouts. Their results show that CNN has better testing accuracy than DNN, while DNN has better RRUFF accuracy than CNN, for both crystal systems and space groups.

Oviedo et al reported a CNN to classify a specific set of space groups[7]. They labeled ICSD entries and predicted experimental patterns of 7 space groups. They applied data augmentation: peak scaling, peak elimination and peak shifting, and also mixing simulated patterns and experimental patterns. The simulated training dataset consists of 164 compounds extracted from ICSD. Eventually, 2000 patterns are augmented from the simulated dataset, and 2000 patterns are augmented from experimentally measured spectra. Three different approaches are applied to split the training and testing dataset. Case 1 is to only use simulated XRD patterns for both training and testing. Case 2 is to only use simulated patterns for training, and only use experimental patterns for testing. Case 3 is to use all simulated patterns plus 80% experimental patterns for training, and the rest 20% experimental for testing. They found that case 1 has the highest accuracy since no experimental data is involved, which means free of experimental errors. Case 2 is trained on simulated data and tested on experimental data, which across two domains, thus has the lowest accuracy. Case 3 has significant higher accuracy that case 2. Their results showed that using all simulated and 80% of experimental patterns as training, which is case 3, gives 89.3% accuracy on the experimental testing data for space group classification. They also applied class activation maps to interpret their

results and concluded 3 main cases for failed classification. The first cause is the mixture of phases in the sample. The second cause is the lack of XRD patterns in the training data: certain space group class only has less than 5 patterns. The third case is missing peaks or too few peaks in the XRD, which lack enough discriminate features.

Lee et al reported CNNs trained by 1,785,405 synthetic patterns to classify quaternary composition phases[10]. They restricted their data to 170 ICSD entries of a quaternary composition pool. They reached 100% on simulated dataset and 98% on real experimental dataset. They also converted a numerical prediction task to step-faction task on identifying phase fraction and reach 98% on simulated dataset and 86% on real experimental dataset. Lee et al also worked on a different quaternary compositional system of Li-La-Zr-O that involves 218 ICSD compounds[11]. For phase classification, they reach 96% and 91% for simulated data and real-world data, respectively. For phase-fraction regression, they reach mean square error of 0.0018 and 0.0024 for simulated data and real-world data, respectively.

Li et al argued on the reliability and capacity of identifying symmetries using only XRDs patterns[12]. They alternatively proposed a new descriptor set that consists of Magpie descriptors. Despite for the fact that for any new material, XRDs are the only explicit inputs we could have, they consider total atom number, covalent radius, ground state value and many other hidden or indirect parameters as inputs. The accuracy of their model reaches 81% and 73% for crystal system and space group, respectively.

Dong et al presented a CNN to classify specific material phases, and tested against both simulated and experimental dataset[13]. For their classification goal, instead of symmetry, they target on scale factors, lattice parameters and crystallite sizes. The input features can be changes along with output predictions, as long as the deep learning can identify and fit based on input statistics.

1.3 Conclusion

In this chapter, we firstly introduced the background of the structure of crystals in terms of three-dimensional periodic arrays of atoms, and basic concepts of symmetry. We then introduced fundamental of X-ray diffusion. We also discussed commonly used deep learning models. In this work, our primary objective is to develop a pipeline to apply deep learning to synthetic powder XRD patterns and to predict the symmetry of experimental XRD patterns.

The main body of this thesis is divided to six chapters.

- In chapter one, we provide a general introduction of this work, including motivation, thesis outline and background.
- In chapter two, we introduce the data generation pipeline to calculate the XRD of crystal structures from ICSD database.
- In chapter three, we introduce the data formatting pipeline to provide formatted training data for the deep learning process.
- In chapter four, we introduce the deep learning pipeline, models configurations

and model results.

- In chapter five, we introduce XRD patterns from experimental database RRUFF to validate the capacity of our models.
- In chapter six, we propose potential directions for future works.

Chapter 2 Data generation pipeline

We need large synthetic XRD data for our training process. The approach is: use the crystal structures from crystal database and calculate the ideal XRD pattern of each crystal structures using diffraction equations.

2.1 Accessing ICSD database

Crystal database usually stores large amount of crystal structure solutions. Cystal structure solutions are conventionally stored in Crystallographic Information File (CIF) format, which was initially adopted by the IUCr in 1990[15].

We purchased the license of Inorganic Crystal Structure Database (ICSD), which is the world's largest database for inorganic crystal structures. Multiple previous reports used ICSD as their crystal structure solution source[6-7,10].

The first step would be downloading the entire database. The downloading requires an additional agreement between the data user and ICSD.

The downloading uses an Application Programming Interface (API) approach, where a user name and password is needed to broadcast to database's Uniform Resource Locator (URL). Officially, ICSD has provided the tutorial to use the API. Luckily, Leverhulme Research Centre for Functional Materials Design (LRCFMD) has shared a python code

with the community under GPL-3.0 License. This code contains the function to iterate each and every entry of ICSD database and download them[16].

2.2 Understanding CIF files

The XRD calculation simply requires a complete crystal structure unit cell. By a complete crystal structure unit cell, it means both unit cell parameters and atoms parameters. The cell parameters stand for unit cell length a, b, c and angles alpha, beta, gamma. The atoms parameters stand for all atoms positions and atoms types.

CIF files contains above information. Prior to the calculation, we first need to extract the information. For all crystal structures downloaded from ICSD, the data is formatted in CIF files.

2.2.1 File lines and the loop

The most common way to read a file in Python is simply to open the file and read all the lines. VAR “cif_file_dir” stores the directory and file name of the file. Notice that an important ARG “encoding” must be UTF-8, otherwise could encounter errors of European alphabet. All lines of the file are stored as a list of strings in VAR “cif_content_lines”.

```
with open(cif_file_dir, "r", encoding="UTF-8") as cif_content:
    cif_content_lines = cif_content.readlines()
```

```
line_count = 0
for line in cif_content_lines:
    line_count += 1
```

```
if ';' in line.strip():
    if ";" == line.strip()[0]:
        if not comment_judge:
            comment_judge = True
        else:
            comment_judge = False
```

2.2.2 Basic index and chemical formula block

The first block tells some fundamental information. We have extracted the following information: the index of entry in ICSD archive, the chemical formula. The index of entry is the most important information for the pipeline. This index will be the index to search for entry in pipeline or outside the pipeline. The chemical formula would tell us the crystal we are looking at. Also, the chemical name can be extracted, but the name is too long and too complicated to be recorded and used. Also, there are entries that don't have chemical name in CIF files. We temporary didn't extract the chemical name.

```
_database_code_ICSD 1
_audit_creation_date 1986-07-25
_audit_update_record 2019-08-01
_chemical_name_common 'Dichromium(III) undecaoxo-tetratellurate(IV)'
_chemical_formula_structural 'Cr2 Te4 O11'
_chemical_formula_sum 'Cr2 O11 Te4'
```

```
_exptl_crystal_density_diffrn 5.17
_citation_title 'Cr2 Te4 O11: une structure a anions complexes (Cr2 O10)(14-)'
```

The python code to extract chemical formula is shown below. For ICSD index, which is ICSD database code, we search for the line that include string “database_code_ICSD”. After we locate the line, the ICSD index is stored immediate after the string. To only extract the index, we simply read the line and split the line by space and take the second item. For chemical formula, we use Regular Expression (RE) to split the line in two parts and take the second string. And replace the quotation and strip to remove additional space mark or new line mark.

```
elif "_database_code_ICSD" in line and not comment_judge:
    icsd_coll_code = line.split()[1]
elif '_chemical_formula_sum' in line and not comment_judge:
    chem_form = re.split(r"['\sB]", line, 1)[1].replace("'", "").strip()
```

2.2.3 Cell parameters and space group block

The cell parameters and space group are stored in one block, as shown below. We extracted six cell parameters: length a, length b, length c, angle alpha, angle beta and angle gamma. These parameters are critical to reconstruct the unit cell. Also, by comparing the value of these parameters, one can already tell the crystal system: while lengths are different and two 90 degrees, we are looking at a monoclinic.

The space group under Hermann–Mauguin (HM) notation can be found, followed by

the space group number. The space group number already serves the purpose of telling the space group thus we don't have to extract HM symbol. Notice that later we will see that the HM symbol tells the subgroup of 230 space group, however here we don't need the subgroups.

```
loop_
_citation_author_citation_id
_citation_author_name
primary 'Meunier, G.'
primary 'Frit, B.'
primary 'Galy, J.'
_cell_length_a 7.016(3)
_cell_length_b 7.545(3)
_cell_length_c 9.728(3)
_cell_angle_alpha 90.
_cell_angle_beta 99.69(5)
_cell_angle_gamma 90.
_cell_volume 507.61
_cell_formula_units_Z 2
_space_group_name_H-M_alt 'P 1 21/c 1'
_space_group_IT_number 14
```

For 6 cell parameters, we simply need to find the line and extract the value. Notice that there are brackets that provides an additional decimal. The influence of this decimal hasn't been tested or proved. We simply keep this additional decimal.

Below shows the code to extract cell lengths. Here we use cell length a for an example.

The first if statement aims to check the format. Correct formatted line should be split into 2 strings if separated by space. The first string should be the key word, the second string should be the value. If the line is split into more or less than 2 strings, the script would return warnings like “Failed: _cell_length_a wrong format”. The warning is clear enough to show the reason for the pipeline failure. Thus, if the format is correct, we take the second string, which is the length value, and put it to a remove bracket function “rmv_brkt”. This function converts the string to float while removes brackets and keeps the additional decimal.

```
elif '_cell_length_a' in line and not comment_judge:
    if len(line.split()) == 2:
        cell_a = rmv_brkt(line.split()[1])
    else:
        return 'Failed: _cell_length_a wrong format'
```

Below shows the code to extract cell angles. Here we use cell angle alpha for an example. Still, the first if statement aims to check the format. While the format is correct, the angle value is converted via “rmv_brkt”. However, for some CIF files, due to some error, the value is unreliable. For example, there are situations where 90.0 degrees have the dot missing and shows 900 degrees. In this case, we could fix the problem, but to keep the input consistent and clear, here we use if statement to check if the angle value is within 180 degrees range. If the value is wrong, we return a warning and we format the wrong value along with the warning message. If both the format and the value is reliable, we convert the degree angles to radian angles and pass them to trigonometric

functions of NumPy library.

```
elif '_cell_angle_alpha' in line and not comment_judge:
    if len(line.split()) == 2:
        cell_alpha = rmv_brkt(line.split()[1])
        if cell_alpha > 180:
            return 'Failed: _cell_length_alpha wrong value {}'.format(cell_alpha)
        cell_alpha = rmv_brkt(line.split()[1]) * np.pi / 180.
    else:
        return 'Failed: _cell_length_alpha wrong format'
```

2.2.4 Atom types, positions and occupancies block

The atom types, positions and occupancies are stored in one block. We go through them one by one:

1) "_atom_site_label" stands for the index of each atom, which is atom type plus index: "Te1".
2) "_atom_site_type_symbol" stands for the oxidation type of the atoms: "Te4+".
3) "_atom_site_symmetry_multiplicity" stands for the total amount of equivalent position of this atom. The rest positions can be calculated via symmetry operation equations in Section 2.2.5. For this case, there are 4 symmetry operation equations thus the symmetry multiplicity is 4. Notice that "O6" has a symmetry multiplicity of 2, since the atom is at a specific position that has overlapped symmetry that cancels the other 2 of them.
4) "_atom_site_Wyckoff_symbol" is not used in this pipeline. The meaning should be

found from IUCr.

5) "_atom_site_fract_x" stands for the fractional coordinate of this atom in the direction of cell length a: "`0.1397(1)`". Followed by y and z.
6) "_atom_site_B_iso_or_equiv" is not used in this pipeline. The meaning should be found from IUCr.
7) "_atom_site_occupancy" stands for the mathematical equivalent occupancy factor of this atom at that position. For some cases, unit cell is mathematically calculated that two types of atoms are occupying a same atom site, where their occupancies are summed to be 1.

```
loop_
_atom_site_label
_atom_site_type_symbol
_atom_site_symmetry_multiplicity
_atom_site_Wyckoff_symbol
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
_atom_site_B_iso_or_equiv
_atom_site_occupancy
Te1 Te4+ 4 e 0.1397(1) 0.8599(1) 0.1762(1) . 1.
Te2 Te4+ 4 e 0.6723(1) 0.8618(2) 0.4158(1) . 1.
Cr1 Cr3+ 4 e 0.3192(3) 0.5019(3) 0.3842(2) . 1.
O1 O2- 4 e 0.158(1) 0.649(1) 0.4866(7) . 1.
O2 O2- 4 e 0.563(1) 0.641(2) 0.4517(8) . 1.
O3 O2- 4 e 0.468(1) 0.375(1) 0.2623(8) . 1.
O4 O2- 4 e 0.127(1) 0.314(1) 0.3429(9) . 1.
```

```
O5 O2- 4 e 0.205(1) 0.644(1) 0.2121(7) . 1.
O6 O2- 2 d 0.5 0 0.5 . 1.
```

The code to find above labels is complicated. Here the explanations will be provided one by one. First, we need to find the beginning of this block. This block starts with STR “loop_”. VAR “atom_loop_line_judge” is set to be true as initial condition. Here if we find STR “loop_”, we save the VAR “line_count” to VAR “atom_loop_line”. This line count stores the line index of STR “loop_”. We need to line count to do subtraction to find the columns of all the labels.

```
elif 'loop_' in line and atom_loop_line_judge and not comment_judge:
    atom_loop_line = line_count
```

For the atom site label, once the string is found, VAR “line_count” is saved as VAR “atom_label_line”. VAR “atom_start_line_judge”, which is initially set to false, is now set to true, indicates the atom block is confirmed to start with STR “_atom_site_label”. VAR “atom_loop_line_judge” is then set to false, which will prevent the rest STR “loop_” repeated considered as the beginning of atom info block.

```
elif '_atom_site_label' == line.strip() and not comment_judge:
    atom_label_line = line_count
    atom_start_line_judge = True
    atom_loop_line_judge = False
```

For the atom type symbol, we know it contained both atom type and oxidation state(ionization). If the STR “_atom_site_type_symbol” is found, we save the line

count to VAR “ion_label_line”. And for VAR “ion_label_line_judge”, which initially to be false assuming that no ionization information is provided, now set to true that we could use ion scattering factor instead of neutral atom scattering factor. Details of ion and neutral atom scattering factors are in Appendix A.

```
elif '_atom_site_type_symbol' == line.strip() and not comment_judge:
    ion_label_line = line_count
    ion_label_line_judge = True
```

For the atom coordinates labels along 3 directions. We simply save their line count to the variables.

```
elif '_atom_site_fract_x' == line.strip() and not comment_judge:
    atom_x_line = line_count
elif '_atom_site_fract_y' == line.strip() and not comment_judge:
    atom_y_line = line_count
elif '_atom_site_fract_z' == line.strip() and not comment_judge:
    atom_z_line = line_count
```

For the atom occupancy, we save its line count to VAR “`atom_ocp_line`”. Then we set VAR “atom_ocp_line_judge” to be true indicating that occupancy factor is provided explicitly, which initial set to be false assuming no occupancy factor is provided.

```
elif '_atom_site_occupancy' == line.strip() and not comment_judge:
    atom_ocp_line = line_count
    atom_ocp_line_judge = True
```

Above are just locating the labels, which provide the identity of columns. Then we need

to locate the value list. The rows of the list indicate how many atom sites are provided. The columns of the list indicate the atom information labels by labels.

To find the beginning of the list, we start from the end of labels. As shown above, all labels start with STR "_atom". If VAR "atom_start_line_judge" is true and STR "_atom" not in line, we consider this to be the first line of the list. This means after we extracted STR "_atom_site_label", we start to find the next line who doesn't start with STR "_atom". After the first line of list is located, we save the line count into VAR "atom_start_line". We then set VAR "atom_start_line_judge" to be false, which prevent the next lines repeatedly considered as the first line of list. Then we set "atom_end_line_judge" to be true indicating that we could start to locate the end of the list, which initially set to be false assuming we haven't located the beginning yet thus no need to find the end. Notice VAR "atom_end_blank_judge" is a special judgment for the end of list, since in some cases the end is blank with no specific string.

```
elif ('_atom' not in line) and atom_start_line_judge and not comment_judge:
    atom_start_line = line_count
    atom_start_line_judge = False
    atom_end_line_judge = True
    atom_end_blank_judge = True
```

To find the end of the list, above judgements have ensured that the beginning is already found. The end of the list has 3 different types. The end could be the start of another block, where STR "loop_" can be found. Or, the end could be starting with STR "#End".

Both scenarios are contained in this if statement. Under these scenarios the end line should be the line before current line count. Thus, for VAR “atom_end_line”, the line count should minus 1.

```
elif atom_end_line_judge and (('loop_' in line) or ('#End' in line)) and not comment_judge:
    atom_end_line = line_count - 1
    atom_start_line_judge = False
    atom_end_line_judge = False
    atom_end_blank_judge = False
```

The third scenario for the end of the list is total blank, which means the file ends at the atom block. In this case there is no line to judge if the file has ended. If VAR “atom_end_blank_judge” is true, which means above if statement is not satisfied, below if statement will be satisfied along the rows of the list to the end of the list, which indicate that the file end with nothing. Thus current line count indicate the end of the list.

```
if atom_end_blank_judge and not comment_judge:
    atom_end_line = line_count
```

With the line count of each atom labels, and the start and end of atom list, we could extract any information we want use a tuple. For the entire lines of the CIF file, the first dimension will indicate the line count which is the nth row of the list, and the second dimension will indicate the label we want which is the nth column of the list.

2.2.5 Symmetry operations block

Symmetry operations block contains all the symmetry equations that this cell follows. Below shows an example of symmetry operations. Above atom block do provide some atom indexes and positions. However, CIF files doesn't include all atom parameters due to symmetry. Which means, for a pair of symmetry position, only the atom at the initial position is provided. We need to use the initial position along with the symmetry equations to find out the paired atom information. As mentioned in Section 2.2.3, each atom has a multiplicity of 4, which means that each atom should go through below symmetry operations to find the rest 3 positions. Notice that there is always a symmetry operation of "x, y, z" stand for the original position itself.

```
loop_
_space_group_symop_id
_space_group_symop_operation_xyz
1 'x, -y+1/2, z+1/2'
2 '-x, -y, -z'
3 '-x, y+1/2, -z+1/2'
4 'x, y, z'
```

The application of combining atom list and symmetry operations is discussed in Section 2.2.5. We need to first locate the line of this block. To locate this block, we need to search for STR "_space_group_symop_operation_xyz" or "symmetry_equiv_pos_as_xyz". If either of these 2 strings are found, we could start extracting symmetry operations from the next line. Thus VAR "symm_start_line" should be current line count plus 1. Also, the founding of the start of symmetry

equations indicate that we could search for the end. VAR “symm_end_line_judge” is then set to true, which is initially false.

```
if '_space_group_symop_operation_xyz' in line or '_symmetry_equiv_pos_as_xyz' in line
and not comment_judge:
    symm_start_line = line_count + 1
    symm_end_line_judge = True
```

The end of symmetry equations can be found followed by the next block. Thus, we could find STR “loop_” while VAR “symm_end_line_judge” is true. Of course, the end line count should be minus 1. And then closing the VAR “symm_end_line_judge”.

```
if 'loop_' in line.strip() and symm_end_line_judge and not comment_judge:
    symm_end_line = line_count - 1
    symm_end_line_judge = False
```

2.3 Calculating XRD pattern

2.3.1 Positions of peaks

In a XRD pattern, the diffraction peaks appear at fixed angles once the periodic lattices are fixed. There are two major factors that determines Bragg angle: 1) interplanar distance, 2) wavelength. The Braggs' law explicitly shows the relation between the diffraction angle(Bragg angle), θ_{hkl}, interplanar distance, d_{hkl}, and the wavelength, λ, as:

$$sin\theta_{hkl} = \frac{\lambda}{2 \cdot d_{hkl}}$$

Equation 2-1

As we already have the wavelength for X-ray, we need to calculate planar distance d_{hkl} from the unit cell. The interplanar distance is a function of the unit cell parameters and Miller indices, h, k and l, which allows us to describe every set of crystallography planes. The general equation to calculate planar distance is:

$$\frac{1}{d_{hkl}^2} = \left[\frac{h^2}{a^2 \sin^2 \alpha} + \frac{2kl}{bc}(cos\beta cos\gamma - cos\alpha) + \frac{k^2}{b^2 \sin^2 \beta} + \frac{2hl}{ac}(cos\alpha \cdot cos\gamma - cos\beta) + \frac{l^2}{c^2 \sin^2 \gamma} + \frac{2hk}{ab}(cos\alpha \cdot cos\beta - cos\gamma)\right] / (1 - \cos^2 \alpha - \cos^2 \beta - \cos^2 \gamma + 2 \cdot cos\alpha \cdot cos\beta \cdot cos\gamma)$$

Equation 2-2

From Section 2.2.3, we already know the unit cell parameters. We still need Miller indices. The Miller indices are considered as an array has a shape of $n \times 3$, where n represent different sets of hkl combinations.

The code applies the equation for calculating planar distance d_{hkl}. The Miller indices were passed by array “hkl_info”. And after the calculation, we obtain an array “hkl_d” that has a shape of $n \times 1$.

```
hkl_h = hkl_info[:, 0]
hkl_k = hkl_info[:, 1]
hkl_l = hkl_info[:, 2]
a = cell_a
b = cell_b
c = cell_c
```

```
alpha = cell_alpha
beta    = cell_beta
gamma = cell_gamma
v = 0
v = (a*b*c) * ( 1 + 2*np.cos(alpha)*np.cos(beta)*np.cos(gamma) - np.cos(alpha)**2 -
np.cos(beta)**2 - np.cos(gamma)**2 )**(1/2)
hkl_d = np.zeros((hkl_info.shape[0], 1))
hkl_d = (1/v) * (hkl_h**2*b**2*c**2*np.sin(alpha)**2 + hkl_k**2*a**2*c**2*np.sin(beta)**2 +
hkl_l**2*a**2*b**2*np.sin(gamma)**2                                                    +
2*hkl_h*hkl_k*a*b*c**2*(np.cos(alpha)*np.cos(beta)-np.cos(gamma))                      +
2*hkl_k*hkl_l*a**2*b*c*(np.cos(beta)*np.cos(gamma)-np.cos(alpha))                      +
2*hkl_h*hkl_l*a*b**2*c*(np.cos(alpha)*np.cos(gamma)-np.cos(beta)))**(1/2)
hkl_d = 1/hkl_d
```

Once the planar distance array is obtained, we can directly apply Bragg's law shown in Equation 2-1. For a planar distance array of shape of $n \times 1$, the output Bragg angle array will also have a shape of $n \times 1$.

Notice that for the code block shown below, we explicitly distinguished degree angles and arc angels.

```
wavelength     = 1.5418
two_theta      = np.zeros((hkl_info.shape[0], 1))
two_theta_pi = np.zeros((hkl_info.shape[0], 1))
for i in range (0, hkl_info.shape[0]):
    if wavelength / 2 / hkl_d[i] < 1:
        theta_cal = np.arcsin(wavelength / 2 / hkl_d[i])
        theta_err = 0
```

```
            theta_obs = theta_cal + theta_err
            two_theta[i] = 2*theta_obs/np.pi*180
            two_theta_pi[i] = 2*theta_obs
two_theta = np.around(two_theta, decimals=2)
```

2.3.2 Peak intensities

The diffraction pattern has multiple Bragg angles, which lead to multiple Bragg peaks, and each has its peak intensities. The peak intensities are affected by three groups of factors: 1) structural factors, 2) specimen factors, and 3) instrumental factors. In our pipeline, we mainly focused on structural factors, which depend on the atomic structure of the crystal. The overall equation for intensities is:

$$I_{hkl} = K \times P_{hkl} \times L_{\theta} \times P_{\theta} \times A_{\theta} \times T_{hkl} \times E_{hkl} \times |F_{hkl}|^2$$

Equation 2-3

Where:

- K is the scale factor as a multiplier to normalize.

- p_{hkl} is the multiplicity factor that accounts for the symmetrical equivalent points. Here we neglect due to our way of generating (hkl)s has covered all symmetrical equivalent points.

- L_{θ} is the Lorentz multiplier, defined by the geometry of diffraction.

- P_{θ} is the polarization factor, represents the polarization of scatted EM wave.

- A_{θ} is absorption multiplier. Here we neglect.

- T_{hkl} is the perfect orientation factor. Here we neglect.

- E_{hkl} is the extinction multiplier. Here we neglect.

- F_{hkl} is the structure factor, which is determined by the positions and types of atoms in the unit cell. The structure factor is the most important factor that determined the peak intensities at different Bragg angles.

The Lorentz multiplier and polarization factor are merged as Lorentz-polarization factor, which is given as:

$$LP = \frac{1 + \cos^2 2\theta}{cos\theta \cdot \sin^2 \theta}$$

Equation 2-4

The LP factor varied with Bragg angle, and has the same amount as the sets of interplanar distance, as well as the sets of Bragg peak positions.

```
lp = np.zeros((hkl_2theta.shape[0], 1))
for i in range (0, hkl_2theta.shape[0]):
    lp[i] = (1 + np.cos(two_theta_pi[i])**2) / (np.cos(two_theta_pi[i]/2)
*np.sin(two_theta_pi[i]/2) ** 2)
```

The structure factor is determined by the distribution of atoms in the unit cell. The equation for structure factor is given as:

$$F_{hkl} = \sum_{j=1}^{n} g_j \cdot t_j \cdot f_j \cdot \exp\left(2\pi i(h \cdot x_j + k \cdot y_j + l \cdot z_j)\right)$$

Equation 2-5

Where:

- n is the total number of atoms in the unit cell.

- g_j is the population and occupation factor of the j^{th} atom.

- t_j is the temperature factor. Here we neglect.

- f_j is the atomic scattering factor of that j^{th} atom type.

- x, y, z is the fractional coordinates of the j^{th} atom.

The occupation factor can be obtained from cif files described in Section 2.2.4. The population factor represents the position of the atom. If the atom is at the corner of the cell, the population factor should be 1/8 as the neighboring unit cell are sharing the same location. There were four locations: interior, surface, edge, corner.

```
pos_pop = np.zeros((cell_info.shape[0], 1))
i = 0
for i in range (0, cell_info.shape[0]):
    j = 1
    count = 0
    for j in range (2, 5):
        if cell_info[i, j] == 1 or cell_info[i, j] == 0:
            count += 1
    if count == 0:
        pos_pop[i, 0] = 1
    elif count == 1:
        pos_pop[i, 0] = 1/2
    elif count == 2:
        pos_pop[i, 0] = 1/4
    elif count == 3:
        pos_pop[i, 0] = 1/8
```

The atomic scattering factor describes the ability of atoms to scatter radiation. For practical computational purposes, the normal atomic scattering factors are represented as follows:

$$f_j(sin\theta/\lambda) = \sum_{i=1}^{4} a_i \exp(-b_i \cdot \sin^2 \theta/\lambda^2) + c$$

Equation 2-6

The scattering factors of chemical elements and ions are represented by nine coefficients, which can be referenced from the International Tables for Crystallography, Vol. C. For the purpose of convenience, the table used by the pipeline can be found in Appendix A.

Since the scattering factors are affected by both Bragg angles and atom types, the scattering factor array is in shape $n \times m$, where n is the amount of set of $hkls$, and m is the total number of atoms in the unit cell.

```
with open("ion scattering table.txt", "r") as scat_table:
    scat_table_lines = scat_table.readlines()
atom_scat = np.zeros((hkl_info.shape[0], cell_info.shape[0]))
i = 0
for i in range (0, cell_info.shape[0]):
    col = np.zeros((hkl_info.shape[0], 1))
    abc_ion = np.zeros((9, 1))
    abc_ion[0:9, 0] = np.array(scat_table_lines[int(cell_info[i, 0]) - 1].split()[3:12])
    j = 0
```

```
    for j in range (0, 7, 2):
        col[:, 0] = col[:, 0] + abc_ion[j, 0] * np.exp(- abc_ion[j+1, 0] * s[:, 0]**2)
    col[:, 0] = col[:, 0] + abc_ion[8]
    col[:, 0] = col[:, 0] * cell_info[i, 5]
    atom_scat[:, i] = col[:, 0]
```

Notice that during the calculation of scattering factor, the occupancy factors obtained from Section 2.2.4 are applied to each atoms in the unit cell.

The calculation involved with atom positions and $hkls$ can be considered as a vector multiplication shown below:

$$h \cdot x_j + k \cdot y_j + l \cdot z_j = (h\ k\ l) \times \begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix} = \boldsymbol{h} \cdot \boldsymbol{x}_j$$

Equation 2-7

With all the position of atoms stored, we can directly multiply two matrix, one $hkls$ matrix and one atom position matrix. The output is in shape $n \times 1$, where n is the amount of set of $hkls$.

```
hkl_pos = np.matmul(hkl_info[:, [0, 1, 2]], cell_info[:, [2, 3, 4]].T)
```

The structure factor F_{hkl} can be calculated following Equation 2.5, where the exponential function that include imaginary part were calculated first, then matrix multiplication, and absolute value, and squared.

```
expArray = np.exp(2 * np.pi * 1j * hkl_pos)
struc = np.zeros((hkl_info.shape[0], 1))
```

```
struc[:, 0] = np.square(np.absolute(np.sum(np.matmul(np.multiply(atom_scat, expArray),
pos_pop), axis=1))).T
```

Then the intensities are purely multiplication of LP factor and structure factor following Equation 2.3. Along with corresponding Bragg angle, we obtained the xy matrix, where x stand for the Bragg angles, and y stand for the intensities.

```
inte = np.zeros((hkl_info.shape[0], 1))
inte[:, 0] = np.multiply(lp[:, 0], struc[:, 0])
x_y = np.zeros((hkl_info.shape[0], 2))
x_y[:, 0] = two_theta[:, 0]
x_y[:, 1] = inte[:, 0]
```

2.3.3 Peak shape functions

Observed peak shapes of diffraction pattern can be described by peak shape functions. The peak shape functions are affected by instrumental boarding, wavelength dispersion, specimen function. There are four commonly used peak shape function: Gauss, Lorentz, Pseudo-Voigt and Pearson-VII. In our pipeline, for the purpose of simplicity, we only adopt Gauss peak shape function. The Gauss peak shape function can be described as:

$$y(x) = G(x) = \frac{C_G^{\frac{1}{2}}}{\sqrt{\pi}H} \cdot \exp(-C_G \cdot x^2)$$

Equation 2-8

Where:

- H are the full widths at half maximum(FWHM), and can be described by three free variables, U, V, W, and with θ as $H = (U \cdot \tan^2\theta + V \cdot tan\theta + W)^{\frac{1}{2}}$. This known as Caglioti formula.
- $C_G = 4ln2$
- $x = (2\theta_i - 2\theta_k)/H_k$, where $2\theta_i$ is the Bragg angle of the i^{th} point of the powder diffraction pattern, and $2\theta_k$ is the ideal Bragg angle of the k^{th} Bragg reflection.

With each individual peak shape function drawn, the intensity Y_i of the i^{th} point of the pattern, would be the sum of several individual peak shapes overlapped together. Say there are m peak shape overlapped at the i^{th} point, the expression would be:

$$Y_i = \sum_{k=1}^{m} I_k[1.5 \cdot y_k(x_k)]$$

Equation 2-9

In the pipeline, the Gauss peak shape function parameters shown in Equation 2-8 were assigned priorly, followed by Equation 2-9, which was assigned to multiple process to speed up the calculation.

```
U = 0.05
V = -0.06
W = 0.07
H = np.zeros((xy_merge.shape[0], 1))
H[:, 0] = (U * (np.tan(xy_merge[:, 0]*(np.pi/180)/2))**2 + V * np.tan(xy_merge[:, 0] *
(np.pi/180)/2) + W)**(1/2)
```

Below shows how multiple Equation 2-9 can be parallel processed. The iteration is on

“x_val” through the entire pattern, and was automatically dispatched to three processes. For each process, the input “x_val” is a list of values, and was fed to function “y_multi”.

```
pool = mp.Pool(3)
pattern = pool.starmap(y_multi, [(x_val, step, xy_merge, H) for x_val in range (0, total_points)])
pool.close()
```

Below shows the python script that stores the function called “y_multi”. Where the peak shape around “x_val” is evaluated and accounted for. To speed up the process, we manual limit the threshold to include the peak shape. Only the peaks within 5 degrees will be calculated as contribution to the intensity at “x_val”.

```
def y_multi(x_val, step, xy_merge, H):
    y_val = 0
    for xy_idx in range (0, xy_merge.shape[0]):
        angle = xy_merge[xy_idx, 0]
        inten = xy_merge[xy_idx, 1]
        if angle > (x_val * step - 5) and angle < (x_val * step + 5):
            y_val = y_val + inten * (gaus((x_val * step - angle), H[xy_idx, 0])*1.5)
    return y_val
```

The “pool” method automatically appends the returned intensities “y_val” to list, which complete the intensities array. Eventually, we obtain the complete pattern with two theta angles as x-axis and diffraction intensities as y-axis.

```
pattern2 = np.zeros((total_points,2))
pattern2[:, 1] = np.asarray(pattern)
pattern2[:, 0] = np.arange(0,180,step)
```

Normalize the y-axis to 0 and 1, and limit both x-axis and y-axis to three decimals.

```
pattern2[:,0] = pattern2[:,0].round(decimals=3)
pattern2[:,1] = (pattern2[:,1] / np.max(pattern2[:,1])).round(decimals=3)
```

Finally, it is optional to plot the pattern and save as a figure to local.

```
plt.figure(figsize=(15,4))
plt.xlim([x_min, x_max])
plt.plot(pattern2[:, 0], pattern2[:,1], label=cif_file)
figName = f"archive_xrd/archive_figure/{ cif_file }.jpeg"
plt.legend()
plt.savefig(figName)
```

2.4 Writing to text files

For every structure we obtained and successfully reconstructed from CIF files, a text file was built to store some key information along with the output pattern. In which, prior to the XY data, there are three header lines that stores the chemical formular and the symmetry. Both crystal structure and space group are stored in the output text files.

```
new_filename = "{}.txt".format(re.split(r"[.]", cif_file)[0])
        with open("{}/{}".format(out_dir, new_filename), "w") as new_file:
            new_file.write(chem_form   + "\n")
            new_file.write("crystal_structure " + str(int(crystal_sys)) + "\n")
            new_file.write("space_group " + str(int(space_group)) + "\n")
            new_file.write("\n".join(str(item).replace("[",  "").replace("]",  "")  for  item  in
pattern2.tolist()))
```

2.5 Conclusion

In conclusion, the pipeline is able to achieve reading raw CIF file to reconstruct unit cell to calculate synthetic XRD pattern. There are totally 204,654 CIF files have been retrieved from ICSD database, and the pipeline can calculate all recognized CIF files in around 72 hours. The output text files are relatively large and might take up to 50 GB if under 0.01 degrees step, or only 5 GB if under 0.1 degrees step. Clearly as the resolution increased 10 times more, the data points increased in 10 times more, which increase the computation cost for both calculating them and transfer them.

Chapter 3 Data cleaning and formatting pipeline

Although ICSD database provide us all the information of structures, the structures can be similar or even identical, as either they are measured multiple times or different groups measured the same structure. These duplicated or similar structure will be computed as duplicated or similar XRD patterns that are telling the same thing. This overlapped information is always treated as useless information, and should be identified and removed.

3.1 Identifying crystal structure similarities

To identify similar crystal structures, we need to summarize a table of rows of structure parameters. This is defined as intermediate database, as the original ICSD database is providing all information, and it is only the format has been changed to better compare. The similarities can be described in various parameters:

1) Unit cell parameters
2) Structure symmetries (space group)
3) Atom types within the unit cell

Where, the unit cell parameters, $a, b, c, \alpha, \beta, \gamma$, are used to describe the shape of the unit cell, which also corresponds to the crystal system. The space group represent the relative atom positions within the unit cell. The atom types represent once the positions are the same, the atom types occupying the positions are also the same.

3.1.1 Intermediate database

Therefore, the intermedia database has several columns for each structure. The 1st column stands for the structure ID, which is inherited from ICSD collection code. The 2nd column stands for the space group index. The 3rd to 8th column stores the unit cell parameters. The 9th to the end stores the atomic numbers involved in the unit cell. The function “cifToInter” simply repeats section 2.2 to extract those required parameters.

Similarity comparison 1 space group & cell & atom.ipynb

```
full_dir = "{}/{}".format(cwdir, path)
cif_return = cifToInter(full_dir, file, ion_idx_enable)
calCount += 1
if type(cif_return) != str:
    print(file, str(int(cif_return[0])), cif_return[1], cif_return[2], cif_return[3],
          cif_return[4], cif_return[5], cif_return[6],
          " ".join(str(value) for value in cif_return[7].astype(int).tolist()),
          file = logFile)
```

Eventually, the intermediate database looks like this(first five rows of the database).

archive_cif-archive_icsd_inter_ld2ad1_ionIdxFalse.txt

```
icsd_000001.cif 14 7.02 7.55 9.73 90.0 99.7 90.0 8 24 52
icsd_000002.cif 62 10.51 6.13 4.82 90.0 90.0 90.0 8 14 25 27
icsd_000003.cif 165 7.18 7.18 7.35 90.0 90.0 120.0 9 57
icsd_000004.cif 165 7.13 7.13 7.29 90.0 90.0 120.0 9 58
icsd_000005.cif 33 7.62 7.9 7.38 90.0 90.0 90.0 1 8 11 15
……
```

3.1.2 Similarity comparison and clusters

After we have the intermedia database, we can compare rows by rows to identify similar pairs. Since we are looking for similar structures, we would expect all parameters to be the same, includes space groups, cell parameters, and atom types. The comparison starts from the 2nd column. Also, an ignore list has been added to skip those pairs that has already been identified.

Similarity comparison 3 intermediate format comparison.ipynb

```
for i in range (0, size):
        if i not in ignoreList:
            for j in range (i+1, size):
                if cifInfoLines[i].split()[1:] == cifInfoLines[j].split()[1:]:
                    print(cifInfoLines[i].split()[0], cifInfoLines[j].split()[0], file = f)
                    ignoreList.append(j)
```

The results would be all pairs that are compared and match. Here shows the first 5 groups identified.

icsd_compared_groups_Id2ad1_ionIdxFalse.txt

```
icsd_000003.cif icsd_060232.cif
icsd_000003.cif icsd_060233.cif
icsd_000003.cif icsd_252346.cif
icsd_000004.cif icsd_056773.cif
icsd_000007.cif icsd_015045.cif
……
```

For one structure, there might be multiple match structures, and they should form

similar clusters. By merging all paired groups, we obtained the clusters.

Similarity comparison 4 identify cluster from compared groups.ipynb

```
for i in range(1, len(contentLines)):
        if contentLines[i].split()[0] == contentLines[i-1].split()[0]:
            file.write(" "+contentLines[i].split()[1].rstrip('\n'))
        else:
            file.write('\n'+contentLines[i].rstrip('\n'))
```

Here shows first few rows of the cluster file.

icsd_cluster_ld2ad1_ionIdxFalse.txt

```
icsd_000003.cif icsd_060232.cif icsd_060233.cif icsd_252346.cif
icsd_000004.cif icsd_056773.cif
icsd_000007.cif icsd_015045.cif icsd_023418.cif
icsd_000025.cif icsd_073972.cif icsd_073973.cif icsd_605321.cif
icsd_000049.cif icsd_000206.cif
```

Once the clusters are calculated, we can simply only pick one structure out of each cluster, and of course added with those that are not with the clusters.

Similarity comparison 5 similarity postprocess.ipynb

```
nosimList = []
for entry in xrdList:
    if entry.replace(".txt", ".cif") not in simList:
        nosimList.append(entry)
oneSimList = []
with open(clusterFile, "r") as file:
    clusterList = file.readlines()
for clusterLine in clusterList:
```

```
    oneSimList.append(clusterLine.strip().split()[0])
finalList = nosimList
for entry in oneSimList:
     finalList.append(entry)
```

The result final list would be a list that contained the cleaned name of structures as shown below. Notice the 1st structure of each cluster are append to the end of the file.

icsd_cleaned_list_171k.txt

```
icsd_000001.txt
icsd_000002.txt
icsd_000005.txt
icsd_000006.txt
icsd_000008.txt
icsd_000009.txt
……
```

3.2 Converting format for deep learning

The formatting issue mainly means that when providing data to the deep leaning process, the files should be easy to access and manage, and can be easily imported to the training pipeline.

3.2.1 Cleaned list

From Section 3.1 we obtained a text file that stores the ids of cleaned entries. To merge the information from the cleaned list to format, we simply iterate though the ids within the cleaned list. All features, labels and ids are following the cleaned list, which means

the output files only contain cleaned data.

[Step 2]XRD 1D pipeline for ICSD format converter for training.ipynb

```
with open(entryFile, "r") as file:
entryList = file.readlines()
for entry in entryList:
    fileName = entry.strip("[\n']").replace(".cif", ".txt")
    fileDir = xrdDir + "/" + fileName
```

3.2.2 Features

The first file is the features file. The features file should store all patterns as rows and intensities as columns, and saved as CSV file. The XY data is extracted from all text files of XRD, and the XY data starts from the fourth line. Since the XY data from XRD text files are normalized to 0 to 1, here we use a trickly approach to multiple 1000 to normalize the pattern between 0 to 1000, and convert all values to integers. Integers have saved file space and also limit the intensity resolution to 1000.

[Step 2]XRD 1D pipeline for ICSD format converter for training.ipynb

```
with open(featuresDir, "a") as features:
    featuresVector = np.zeros((1, 8500))
    i = 0
    for i in range (0, 8500):
        featuresVector[0, i] = float(xrdLines[i+3+500].split()[1]) * 1000
    featuresVector = featuresVector.astype(int)
    np.savetxt(features, featuresVector, fmt="%d", delimiter=",")
```

3.2.3 Labels

The second and third files are the labels files. For this pipeline, we require both crystal system label, and space group label. To simplify, we call crystal system label, the 7-way label. And the space group label, the 230-way label. The 7-way label is on the second line of XRD files. The 230-way label is on the third line of XRD files.

[Step 2]XRD 1D pipeline for ICSD format converter for training v03.ipynb

```
with open(labels7Dir, "a") as labels7:
    labels7Array = np.zeros((1, 7))
    labels7Array[0, int(xrdLines[1].split()[1]) - 1] = 1
    labels7Array = labels7Array.astype(int)
    np.savetxt(labels7, labels7Array, fmt="%d", delimiter=",")
with open(labels230Dir, "a") as labels230:
    labels230Array = np.zeros((1, 230))
    labels230Array[0, int(xrdLines[2].split()[1]) - 1] = 1
    labels230Array = labels230Array.astype(int)
    np.savetxt(labels230, labels230Array, fmt="%d", delimiter=",")
```

3.2.4 Ids

The file name of XRD files are the ids, which is also the ICSD collection code. The Ids file simply stores the ids of each row of your features file and labels files.

[Step 2]XRD 1D pipeline for ICSD format converter for training v03.ipynb

```
with open(idsDir, "a") as ids:
    ids.write(fileName.replace(".txt", "").strip())
    ids.write("\n")
```

3.3 Conclusion

In conclusion, after the data cleaning and formatting, we obtained four CSV files as the output: 1) features.csv, 2) labels7.csv, 3) labels230.csv, 4) ids.csv. For different XRD settings and similarity setting, we can obtain different output. All output folder has the same 4 file names, which is convenient for us to organize different dataset and import them to the deep learning step.

Chapter 4 Deep learning pipeline

Once we have prepared the dataset, we can start applying deep learning to the dataset. We use Pytorch as the deep learning library. Pytorch library is an open-source machine learning framework, primarily developed by Meta AI. Pytorch is very commonly used for academic purpose for application such as computer vision and language processing. Also, commercial companies are adopting to Pytorch such as Tesla and Uber. In out deep learning pipeline, all math algorithm and background processes are carried out by Pytorch, and for the simplicity of focusing on materials science topic, we consider deep learning algorithms as Blackbox, and will focus on analyzing data in this dissertation.

4.1 Configuring dataset

The dataset mentioned in this chapter is the term, stands for the dataset of XRDs generated by the pipeline of Chapter 2, and cleaned, formatted by Chapter 3. The raw dataset from ICSD provide 204,654 entries. After the data cleaning, wo obtained a dataset contains 171,006 entries. This dataset is called the “171k” dataset. However, “171k” only represent a single feature of the dataset: the total amount of data points in the dataset. Within each data point, which is one XRD pattern of a structure, the XRD could still have varies parameters, such as the two-theta range, the two-theta resolution, the peak shape functions, or whether the pattern has noise or not. Configuring dataset means that with a fixed data cleaning process, with the 171k data points assigned, we still need to config the parameters of XRDs to have multiple different 171k dataset.

4.1.1 Dataset

Based on peak shape function and noise, we have four basic datasets based on cleaned 171k entries list. We have two peak shapes apply to our pipeline, both Gauss functions, but follows different Caglioti formula, which leads to different FWHM.

1) Peak shape 1: $u = 0.05, v = -0.06, w = 0.07$
2) Peak shape 2: $u = 0.05, v = -0.01, w = 0.01$

Notice that for an experimental XRD pattern, the exact peak shape functions are determined by the equipment. Thus, for a calculated pattern, or simulated, or synthetic pattern, there is no "correct" peak shapes. In the next Chapter, we will explain the different performance caused by different training data. Besides peak shape, there are also two types of noise.

1) No noise
2) Uniform distributed noise: $0.2\%\sim2\%$

Where, no noise suggest that the patterns are perfect, that those intensities who are not at Bragg angles are direct zeros. A uniform distributed noise suggests that for each intensity, the background noise is a random value between 0.2% and 2%.

Eventually, we combine two peak shapes and two noises to obtain four groups of datasets. Also., instead of having single peak shape or noise amplitude throughout the

171k dataset, we mix peak shapes and noise, to randomly choose one group of parameters for each pattern, while remaining the total datapoints still 171,006.

1) Peak shape 1 + no noise
2) Peak shape 1 + noise(2%)
3) Peak shape 2 + no noise
4) Peak shape 2 + noise(2%)
5) Mix

The dataset is loaded to Pytorch using Dataset class. After import Pytorch to the script, we need to define a new class that inherit Dataset and provide the "_getitem_" the data point itself and indexes of that data point. The indexes are used to query the data and make the train test split. In out code, the "XRDData" class inherit the Dataset class, and both train and test data are split using this class.

```
class XRDData(Dataset):
    def __init__(self, train=True, train_test_split=0.9):
        self.feature_file = featuresFolder + "features.csv"
        self.label_file = featuresFolder + f"labels{num_classes}.csv"
        with open(self.feature_file, "r") as f:
            self.feature_lines = f.readlines()
        with open(self.label_file, "r") as f:
            self.label_lines = f.readlines()
        self.num_datapoints = len(self.label_lines)
        print(f"datapoint: {self.num_datapoints}")
        self.train_test_split = train_test_split
        self.size = train_size = round(self.num_datapoints * self.train_test_split)
```

```
        self.train = train
        if not self.train:
            self.size = self.num_datapoints - train_size
        self.train_inds = np.random.choice(np.arange(self.num_datapoints), size=train_size, replace=False)
        self.test_inds = np.array([x for x in np.arange(self.num_datapoints) if x not in self.train_inds])

    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        if self.train:
            idx = self.train_inds[idx]
        else:
            idx = self.test_inds[idx]
        line = self.feature_lines[idx]
        x = list(map(float, line.strip().split(",")))
        x = torch.FloatTensor(x)
        line = self.label_lines[idx]
        y = list(map(float, line.strip().split(",")))
        y = torch.FloatTensor(y)
        y = torch.argmax(y)
        return x, y
```

This Dataset class is a Pytorch class that will be called in DataLoader. This is the standard way that Pytorch handle dataset.

4.1.2 DataLoader

The DataLoader is used to load Dataset class, and it calls the "__getitem__" to index and iterate all data. We use the DataLoader to load the dataset to memory while split the training and testing data. Here we additionally define four new variables for Dataset and DataLoader.

```
train_test_split = 0.9
train_dataset = XRDData(train=True, train_test_split=train_test_split)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_dataset = XRDData(train=False, train_test_split=train_test_split)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
```

Where, the "train_dataset" is the Dataset class that is configured to be training data, and the "train_loader" is the DataLoader that loads the training data. Similarly, the "test_dataset" repeats the Dataset class, but configured to be training data. And the "test_loader" is the DataLoader that loads the testing data. All batch size is 128.

After both training and testing data is loader, we need to iterate the DataLoader and feed the data to the model and perform deep learning. Prior to that, we still need to config the deep learning parameters.

4.1.3 Optimizer and loss function

The estimating equation and loss function are defined before the training. Here, we pick Stochastic gradient descent(SGD) for optimizing the object function. And for loss

function, we pick the cross-entropy loss between inputs and targets. Those are standard and common functions for a deep learning model. And since those detailed math is part of Computer Science and are complicated and well developed, so we will treat them as Blackbox and just represent the settings.

```
optim = SGD(model.parameters(), lr=lr)
cost = nn.CrossEntropyLoss()
```

4.2 Applying DNN model

We apply the same architecture on the five different datasets, and to see the performance. The settings of the datasets are following Section 4.1.1.

1) Peak shape 1 + no noise
2) Peak shape 1 + noise(2%)
3) Peak shape 2 + no noise
4) Peak shape 2 + noise(2%)
5) Mix

4.2.1 Architecture

Deep Neural Networks models have fully connected dense layers. The initial layer has the length of the data point dimensions. And the output length is the amount of output labels.

In our case, since we are working on a two-theta between 5 to 90 degrees and with a resolution of 0.01 degrees, we have 8,500 dimensions in an array, which include 5.00

degrees and exclude 90.00 degrees. For 7-way classification, which is the crystal system classification, we apply commonly used ReLu activation function, and applied a dropout of 0.3. The architecture is shown below.

```
model = nn.Sequential(
    nn.Linear(8500, 2048),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(2048, 512),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(512, 128),
    nn.ReLU(),
    nn.Dropout(0.3),

    nn.Linear(128, num_classes),
    )
```

Where, the DNN models have four fully connected layers. The length of each layer is determined by several trials and finally tuned.

For 230-way classification, which is the space group classification, we followed the same architecture as 7-way. Instead of 7 neurons in the output layer, we have 230 neurons.

4.2.2 Training testing accuracy and loss

The training and testing accuracy and loss during training for five dataset is plotted as below. To figure out effect of peak shape functions, we apply 7-way model on all 5 datasets.

1) Peak shape 1 with no noise

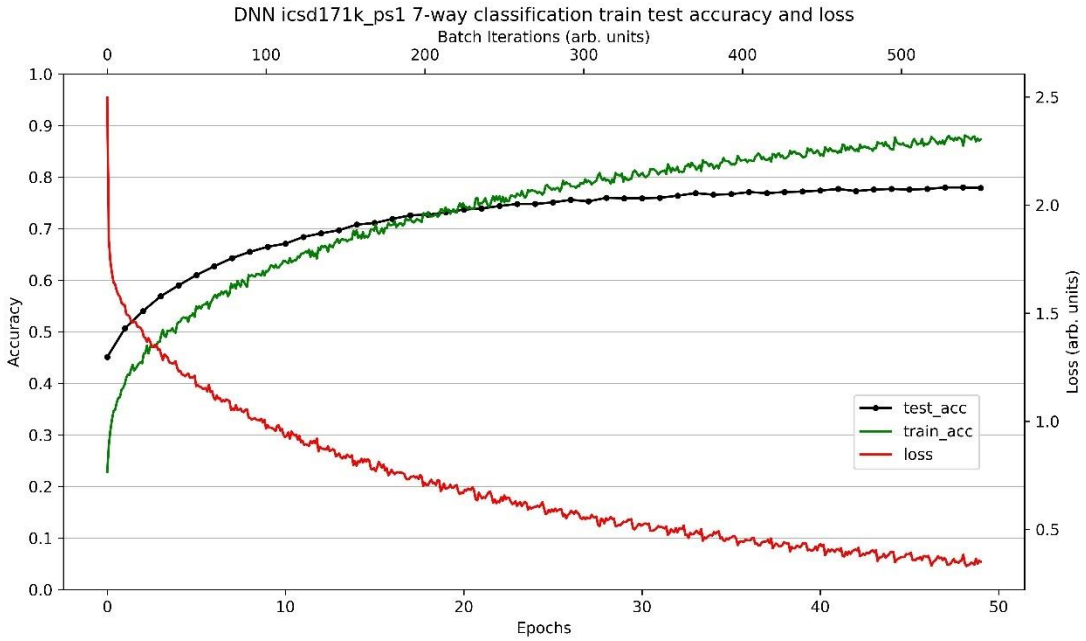

Figure 4.1 Training and testing accuracy and loss of 7-way DNN trained on dataset peak shape 1 with no noise.

2) Peak shape 1 with noise 2%(20 out of 1000)

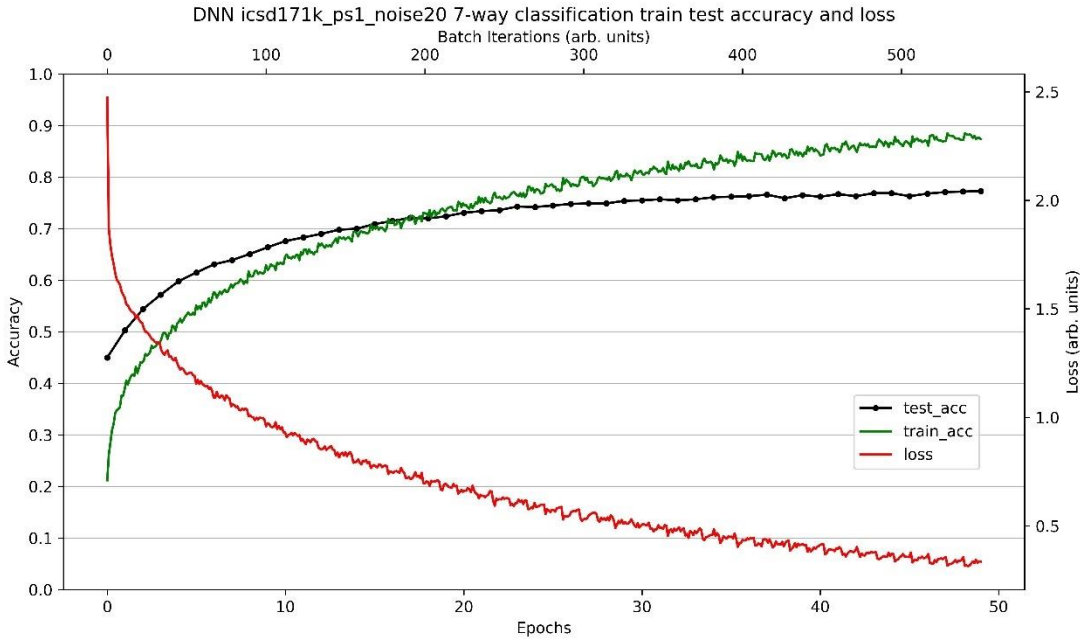

Figure 4.2 Training and testing accuracy and loss of 7-way DNN trained on dataset peak shape 1 with noise.

3) Peak shape 2 with no noise

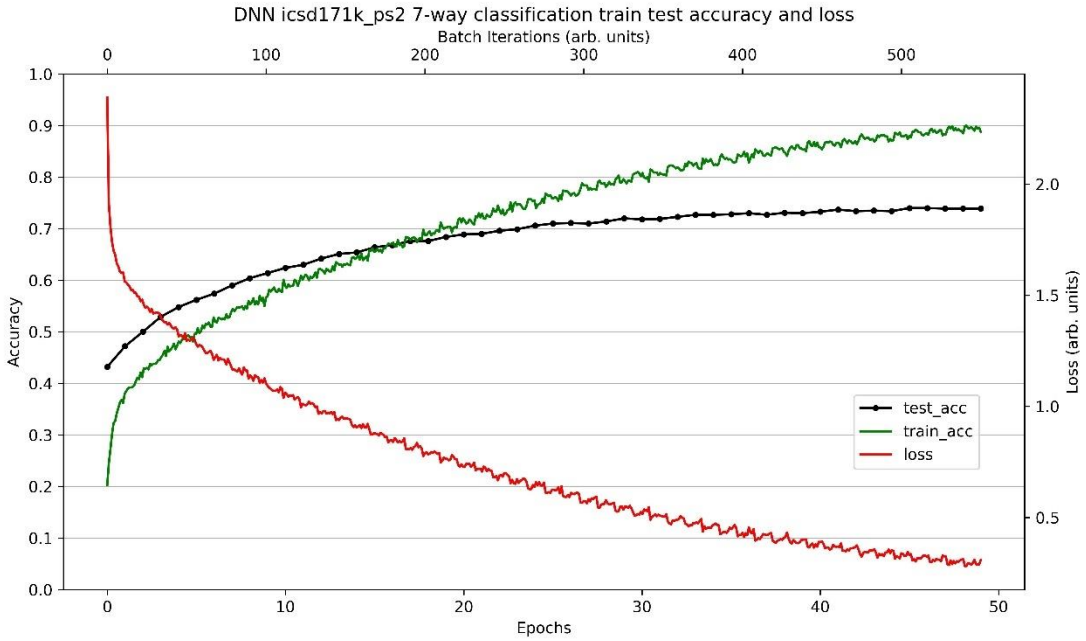

Figure 4.3 Training and testing accuracy and loss of 7-way DNN trained on dataset peak shape 2.

4) Peak shape 2 with noise 2%(20 out of 1000)

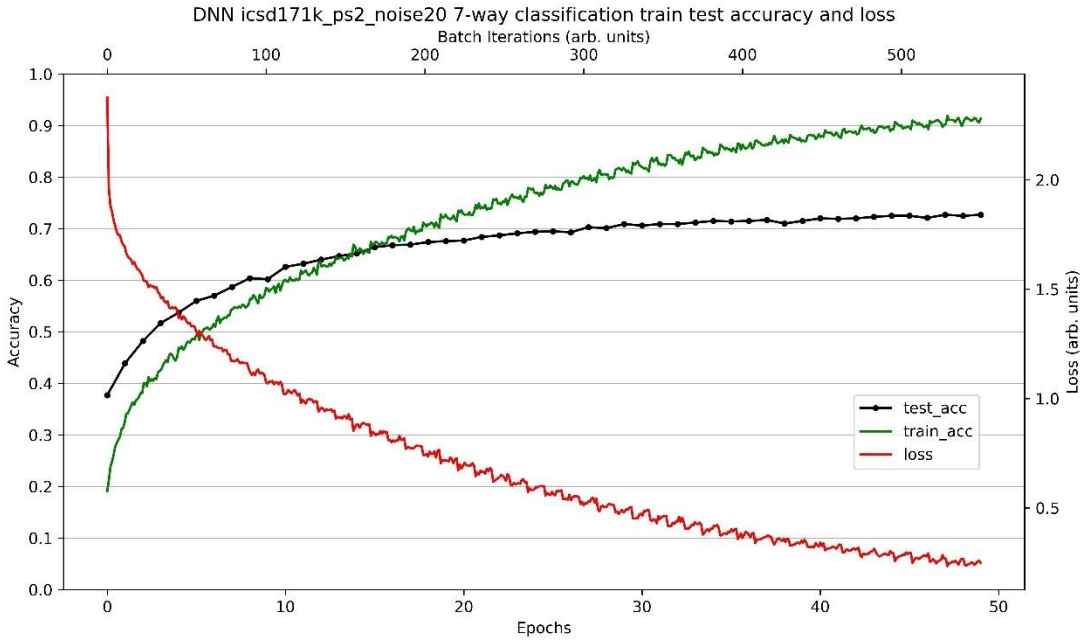

Figure 4.4 Training and testing accuracy and loss of 7-way DNN trained on dataset peak shape 2 with noise.

5) Mix

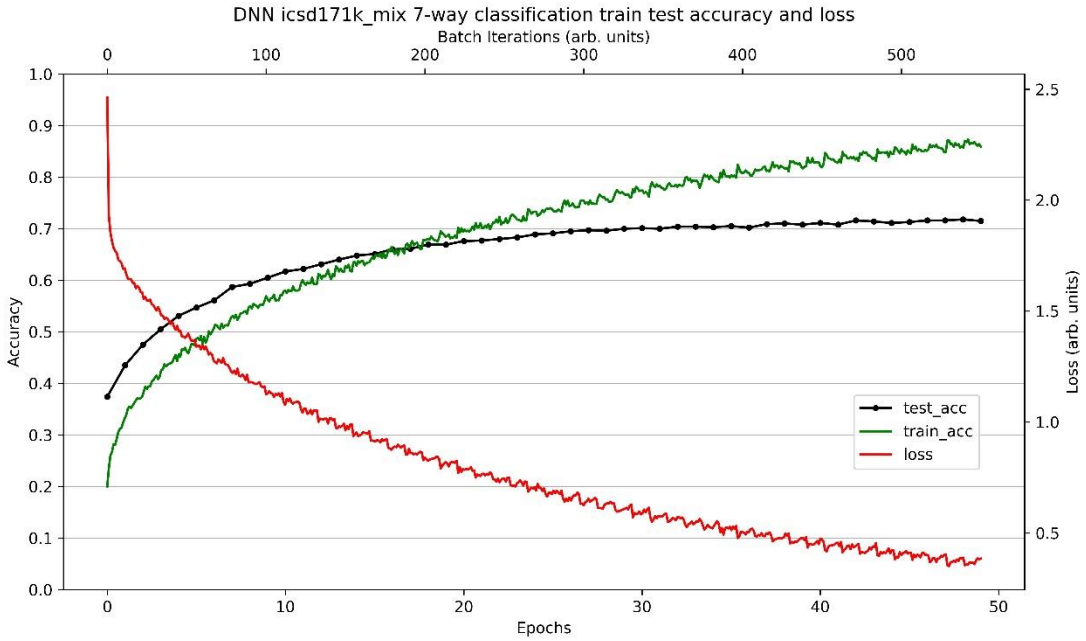

Figure 4.5 Training and testing accuracy and loss of 7-way DNN trained on dataset mix.

The testing accuracies of 7-way DNN models of 5 datasets are summarized below.

DNN 7-way	PS1	PS1+noise20	PS2	PS2+noise20	Mix
Testing accuracy	77.9%	77.3%	73.9%	72.7%	71.5%

Table 4-1Testing accuracies of 7-way DNN models of 5 datasets.

We apply 230-way model only on "Mix" dataset. The reason to only accept "Mix" dataset will be detailed discuss in Chapter 5. Below shows the training and testing accuracy and loss for 230-way DNN model. The testing accuracy for 230-way is 50.6%.

DNN dataset "Mix"	7-way	230-way
Testing accuracy	71.5%	50.6%

Table 4-2 Testing accuracy of 7-way and 230-way DNN models of dataset "Mix".

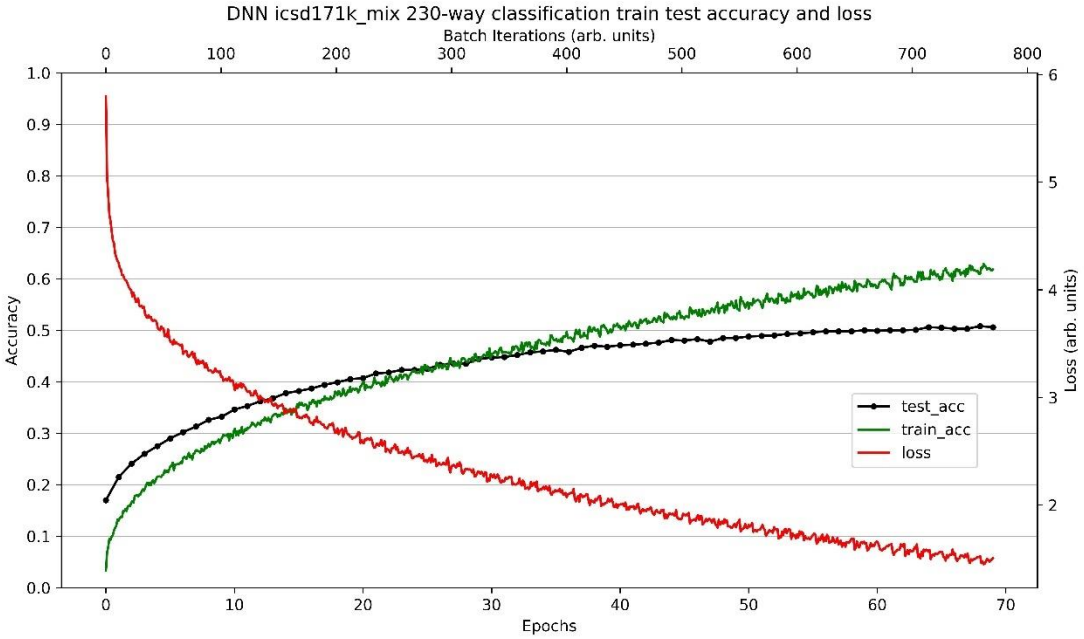

Figure 4.6 Training and testing accuracy and loss of 230-way DNN trained on dataset mix.

4.2.3 Confusion matrix

The confusion matrix is plotted only based on the last dataset “mix”. Confusion matrix shows the prediction preference and the degree of deviation. Especially, confusion matrix can show us the preference of certain predictions under certain labels.

For 7-way classification, the confusion matrix of DNN of dataset “mix” is shown below. The confusion matrix shows the number of entries in the testing set. The darker the color means the larger the amount. The absolute value plot shows the datapoints distribution of different class.

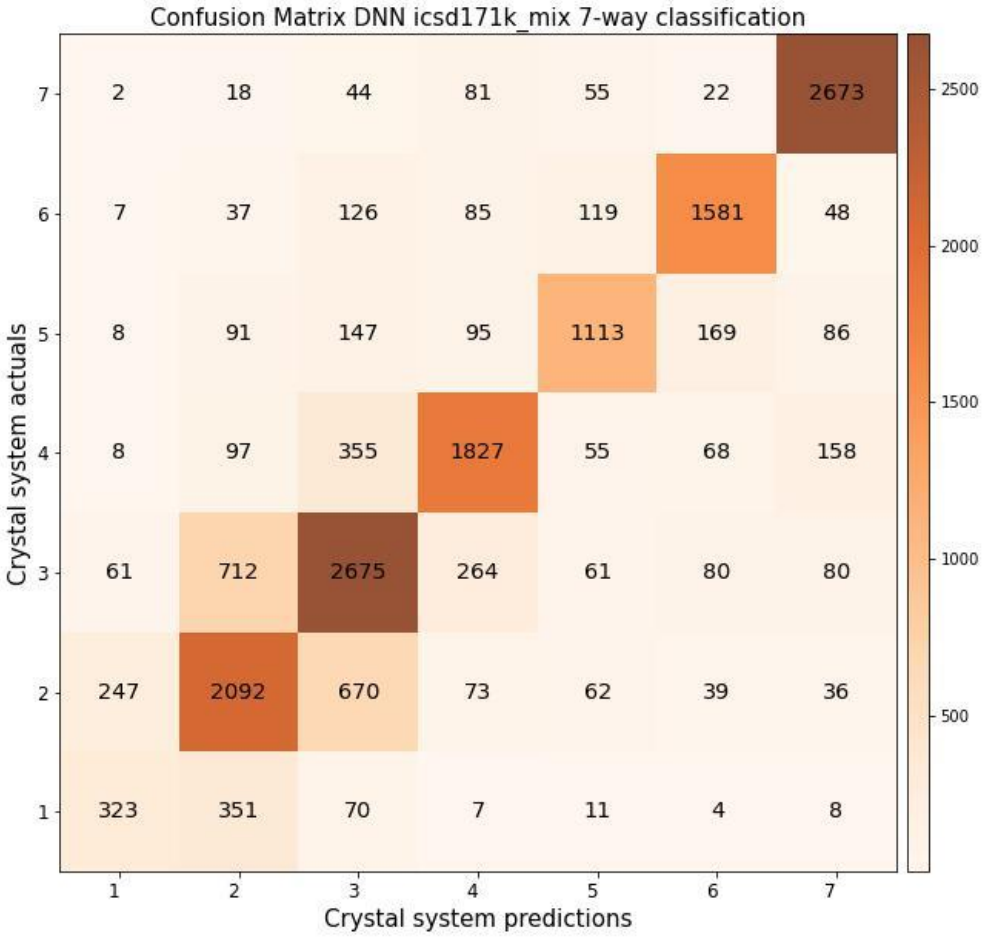

Figure 4.7 The confusion matrix on testing set of 7-way DNN trained on dataset mix.

The percentage plot shows the precent distribution within each class. Compare with the absolute value plot, percent plot shows that the model of 7-way has high distribution along the diagonal, which indicate the predictions are matching with the labels.

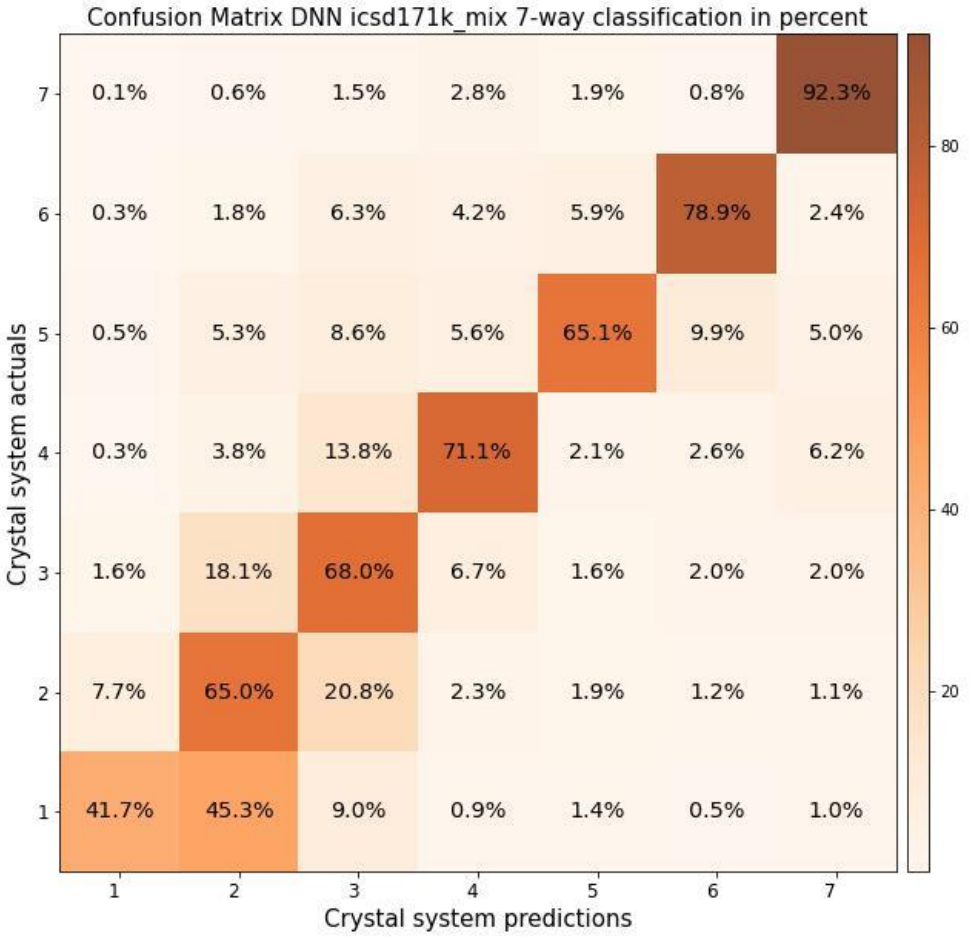

Figure 4.8 The percentage confusion matrix on testing set of 7-way DNN trained on dataset mix.

Noticeable, for the 1st crystal system, triclinic, the model is having large bias that prefers to predict triclinic as 2nd crystal system monoclinic. Also, for the actuals of 2nd and 3rd crystal system, the model is having slight bias to confuse between them.

For 230-way classification, the confusion matrix of DNN of dataset “mix” is shown below. Compare with the 7-way confusion matrix, due to the large number of classes, we only use color darkness to show the counts. Clearly, for space group, if we look at the absolute value, the distribution is totally unclear. Hence, a percent plot is necessary.

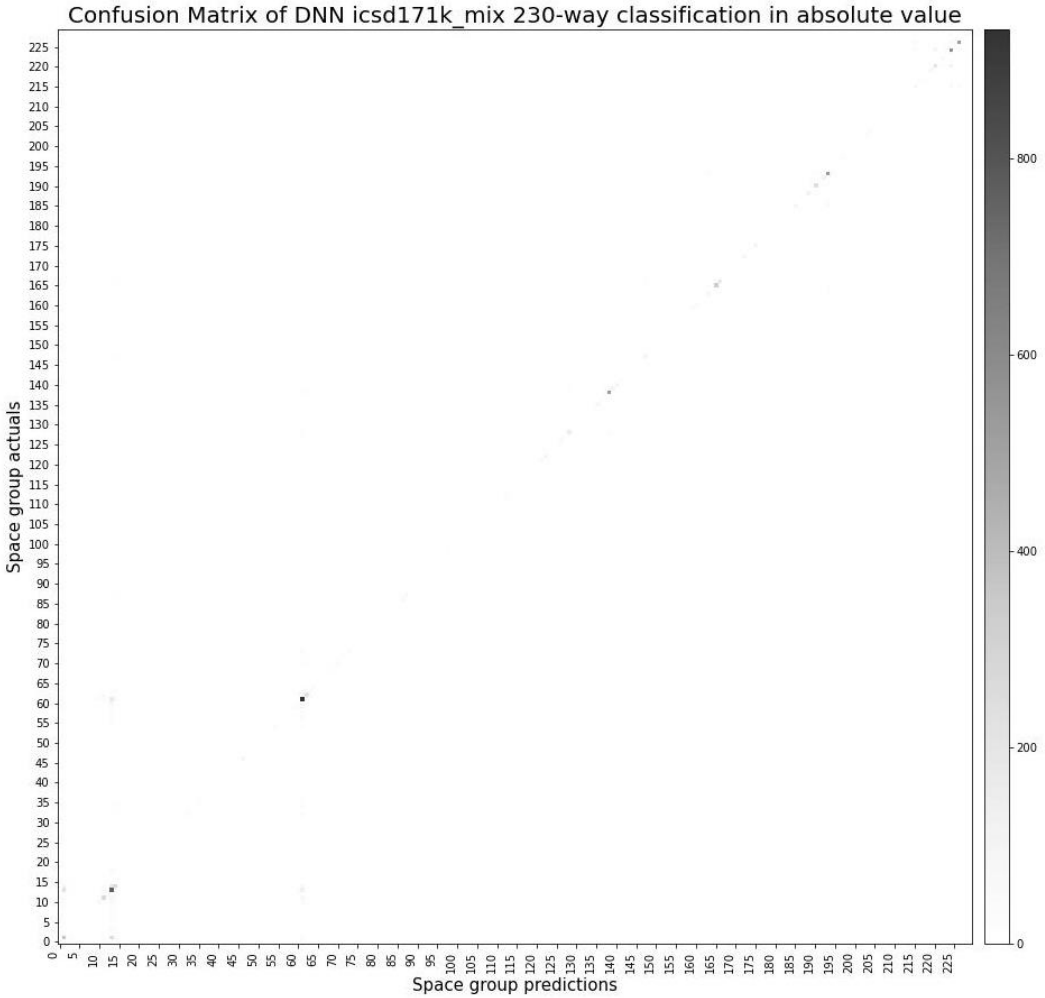

Figure 4.9 The confusion matrix on testing set of 230-way DNN trained on dataset mix.

The percent plot of 230-way confusion matrix is shown below.

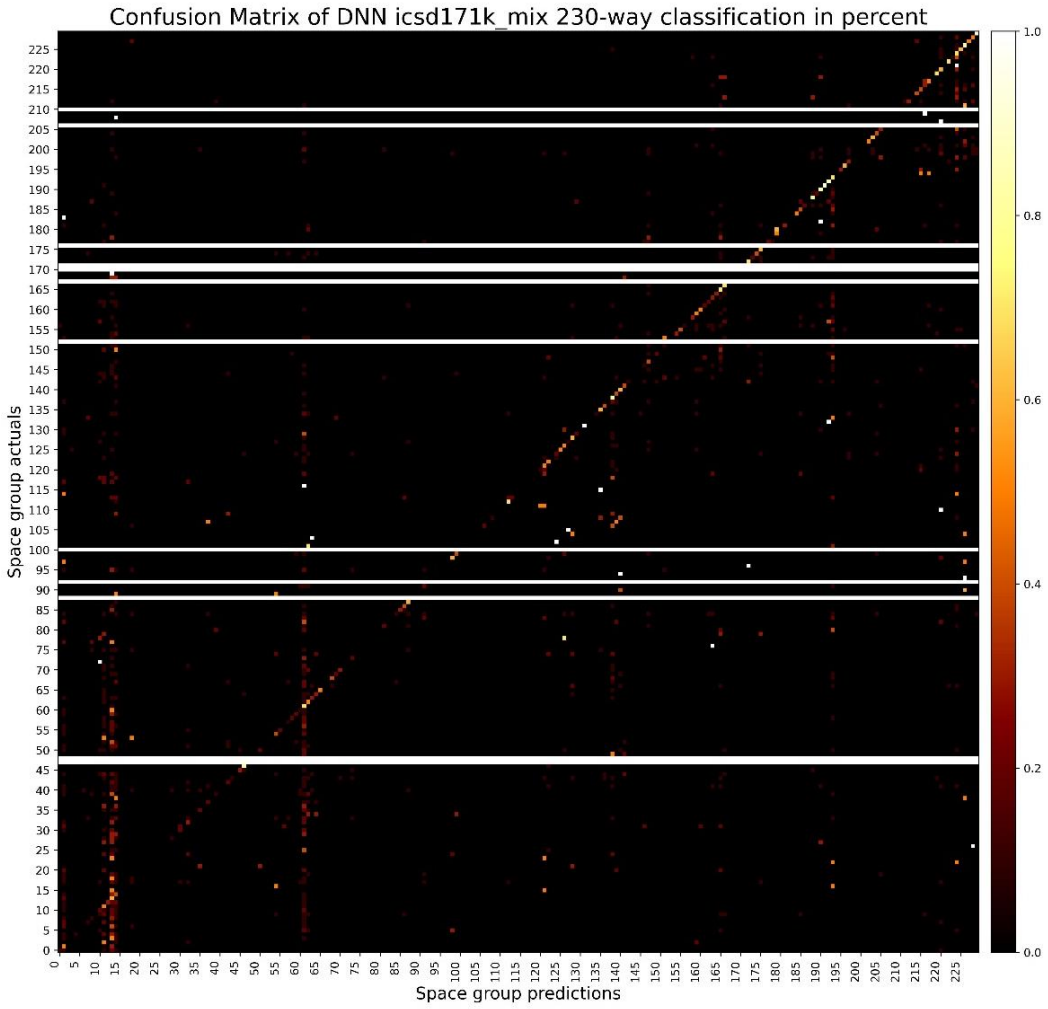

Figure 4.10 The percentage confusion matrix on testing set of 230-way DNN trained on dataset mix.

Where, in this plot, the diagonal shows good matching between predictions and labels. However, there are several classes that are wrongly predicted. Notice that not all space groups are shown in the testing set, understandably for some minority space group.

4.3 Applying CNN model

Convolutional Neural network models are several groups of convoluted layers and pooling layers, and followed by several fully connected layers. We only run training on the mix dataset.

4.3.1 Architecture

The architecture of CNN is followed by common setting and tuned based on several trials. For 7-way and 230-way classification, we use different convolutional layers parameter. The model architecture for 7-way CNN is shown below.

```
model = nn.Sequential(

    nn.Conv1d(1, 64, kernel_size=50, stride=5),
    nn.BatchNorm1d(64),
    nn.Dropout(0.2),
    nn.ReLU(),

    nn.MaxPool1d(kernel_size=2, stride=2),

    nn.Conv1d(64, 64, kernel_size=20, stride=2),
    nn.BatchNorm1d(64),
    nn.Dropout(0.2),
    nn.ReLU(),

    nn.MaxPool1d(kernel_size=2, stride=2),
```

```
    nn.Conv1d(64, 64, kernel_size=20, stride=2),
    nn.BatchNorm1d(64),
    nn.Dropout(0.2),
    nn.ReLU(),

    nn.MaxPool1d(kernel_size=2, stride=2),

    nn.Conv1d(64, 64, kernel_size=10, stride=2),
    nn.BatchNorm1d(64),
    nn.Dropout(0.2),
    nn.ReLU(),

    nn.MaxPool1d(kernel_size=2, stride=2),

    nn.Flatten(1),

    nn.Linear(576, 512),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, num_classes),
  )
```

For 230-way classification, the CNN architecture is shown below.

```
model = nn.Sequential(
```

```
nn.Conv1d(1, 8, kernel_size=5, stride=1),
nn.BatchNorm1d(8),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(8, 8, kernel_size=5, stride=1),
nn.BatchNorm1d(8),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(8, 16, kernel_size=5, stride=1),
nn.BatchNorm1d(16),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(16, 16, kernel_size=5, stride=1),
nn.BatchNorm1d(16),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(16, 32, kernel_size=5, stride=1),
```

```
nn.BatchNorm1d(32),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(32, 32, kernel_size=5, stride=1),
nn.BatchNorm1d(32),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(32, 64, kernel_size=5, stride=1),
nn.BatchNorm1d(64),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Conv1d(64, 64, kernel_size=5, stride=1),
nn.BatchNorm1d(64),
nn.Dropout(0.2),
nn.ReLU(),

nn.MaxPool1d(kernel_size=2, stride=2),

nn.Flatten(1),
```

```
nn.Linear(1856, 512),
nn.ReLU(),
nn.Dropout(0.2),
nn.Linear(512, 256),
nn.ReLU(),
nn.Dropout(0.2),
nn.Linear(256, num_classes),
)
```

Where, each convolutional layer is immediately followed by a max pooling layer to reduce the dimensions in half. We apply dropout and ReLu activation function.

4.3.2 Training testing accuracy and loss

The train test accuracy and loss of CNN of mix dataset is shown below.

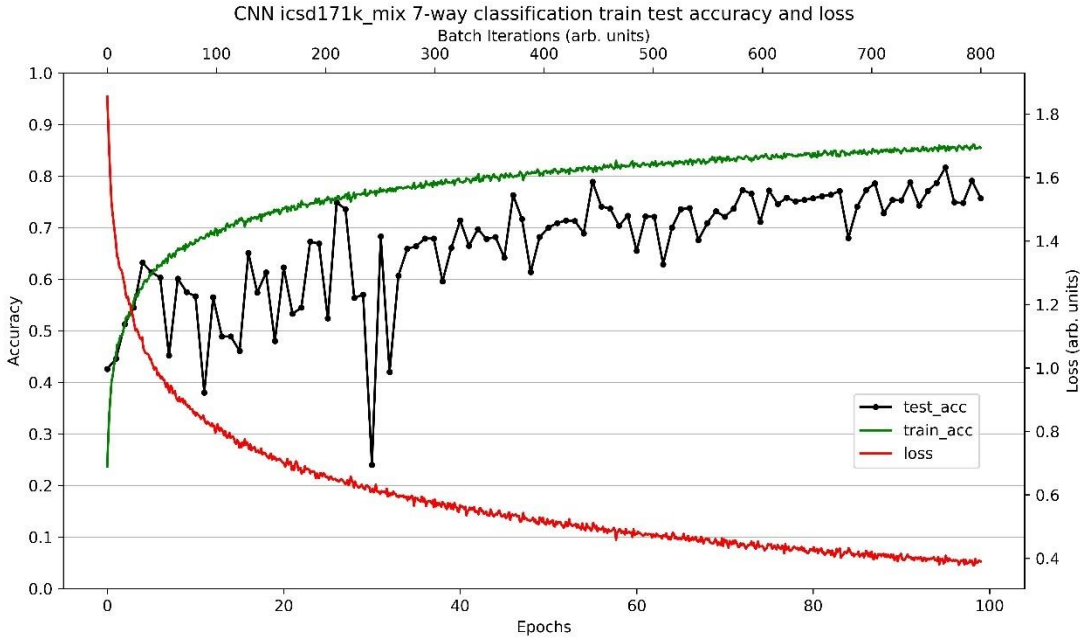

Figure 4.11 Training and testing accuracy and loss of 7-way CNN trained on dataset mix.

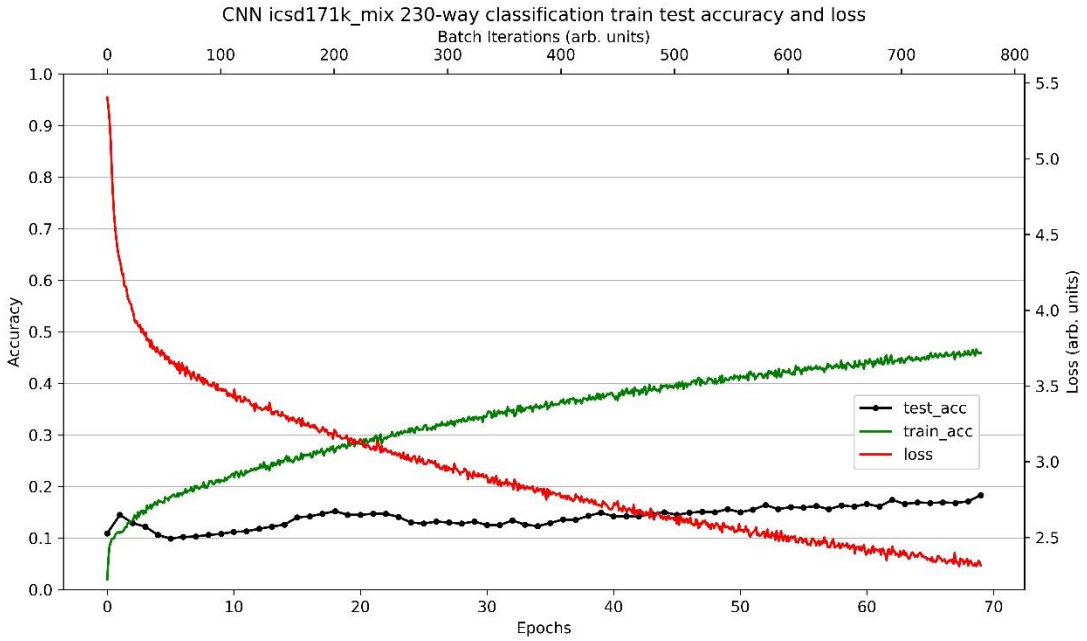

Figure 4.12 Training and testing accuracy and loss of 230-way CNN trained on dataset mix.

4.3.3 Confusion matrix

The confusion matrix of CNN of dataset “mix” in 7-way is shown below. Same as DNN confusion matrix, this is count up based on testing dataset, and the crystal system distribution is similar.

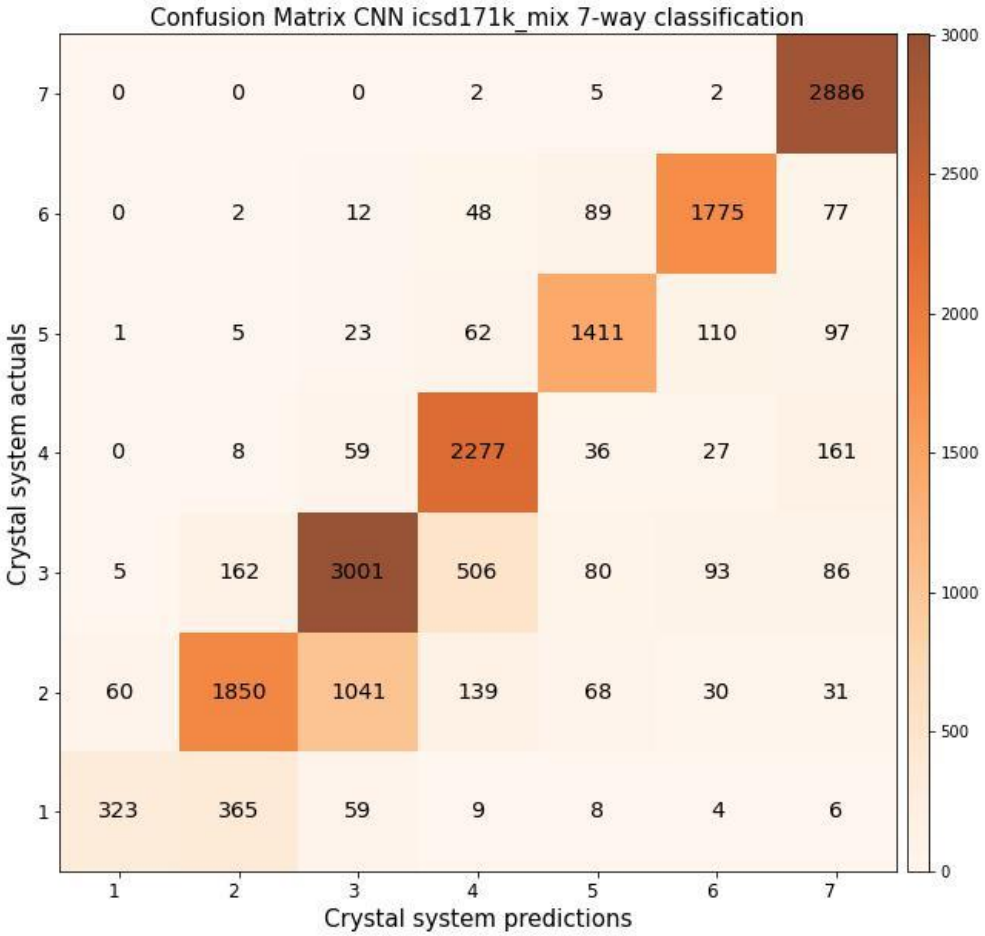

Figure 4.13 The confusion matrix on testing set of 7-way CNN trained on dataset mix.

Despite showing absolute value, percent diagonal value better indicate that the predictions are matching with the labels.

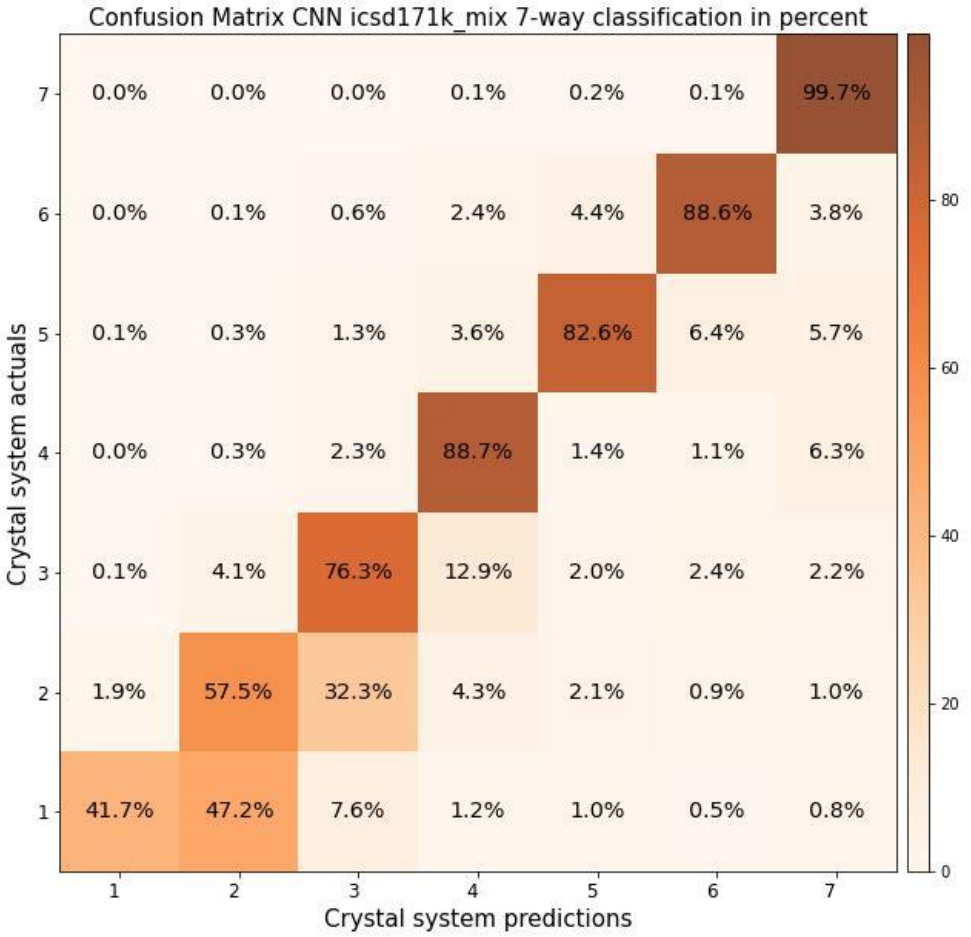

Figure 4.14 The percentage confusion matrix on testing set of 7-way CNN trained on dataset mix.

Here shows the 230-way confusion matrix. Similarly, absolute value darkness shows the class distribution.

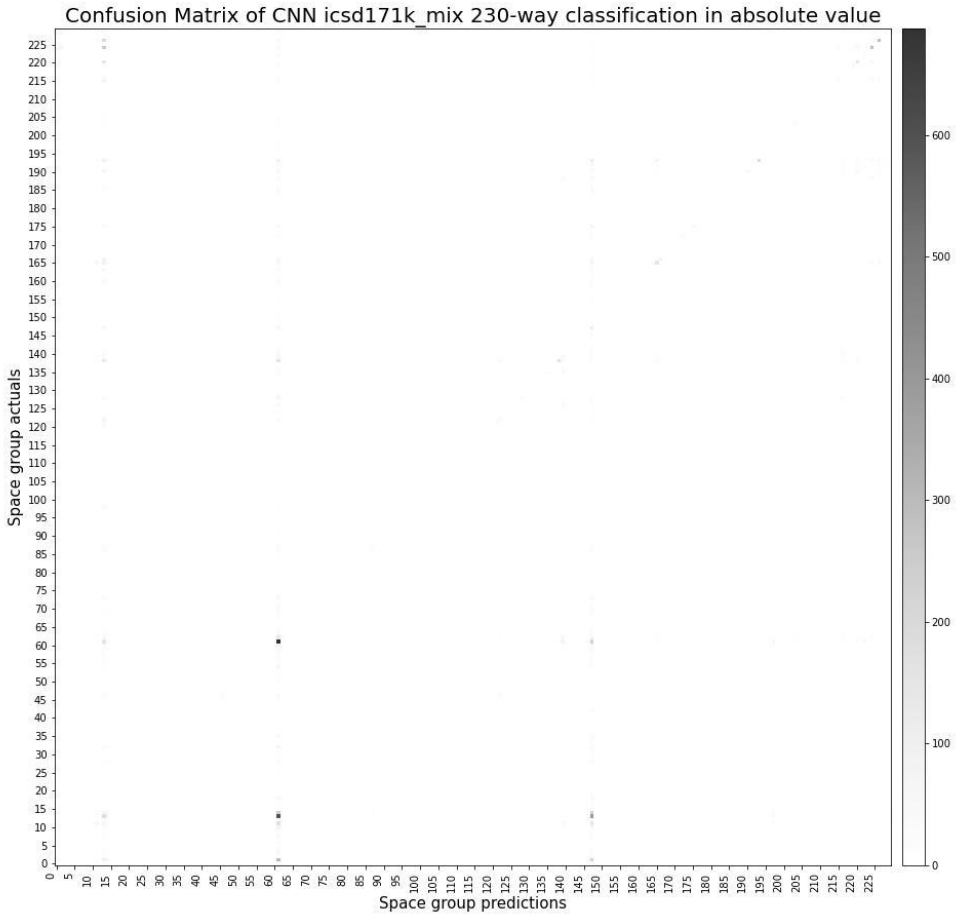

Figure 4.15 The confusion matrix on testing set of 230-way CNN trained on dataset mix.

The percent confusion matrix is shown below. Here we observe few classes are wrongly predicted. And also, we observe few missing classed, due to the limited test set.

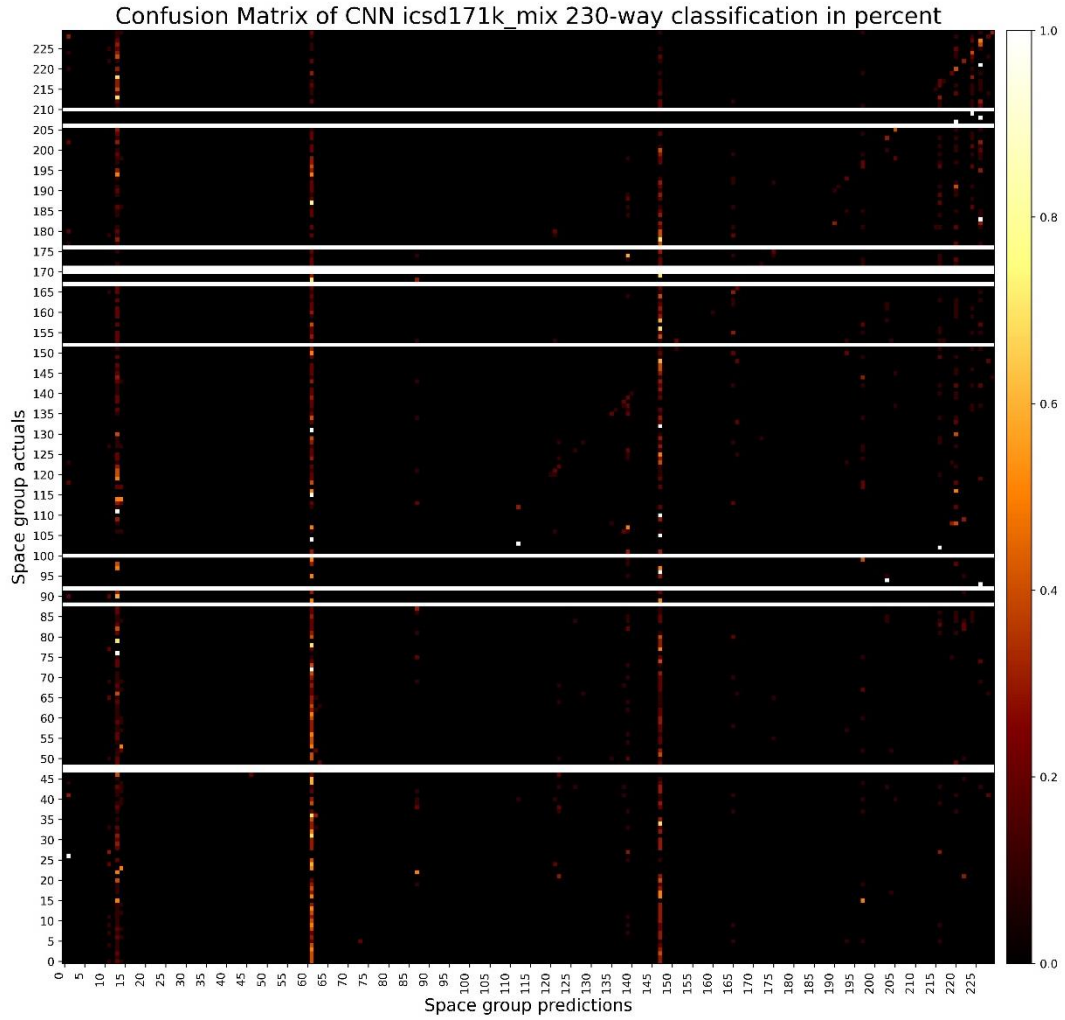

Figure 4.16 The percentage confusion matrix on testing set of 230-way CNN trained on dataset mix.

4.4 Comparing DNN and CNN results

For CNN training and testing accuracy and loss, we observe significant difference than DNN models. This conclusion is limited to the training of synthetic dataset generated through our pipeline. The synthetic data generated from ICSD follows all physical equations. All peak positions and intensities are perfect and fixed.

The confusion matrixes of CNN and DNN are different, especially in the 230-way classification. CNN has more wrongly predicted space groups are not the same, comparing Figure 4.16 and Figure 4.9.

Clearly, DNN has significant advantage towards CNN. Since we are looking at a 1D problem, the performance difference is obvious through the training. We discover significant difference on the second domain, and that will be discussed in Section 5.2.3.

4.5 Conclusion

In this Chapter, we talked about the deep learning training pipeline. The dataset settings and model architectures are introduced. We showed the performance of models on different datasets by plotting train test accuracy and loss. We also explicitly showed the performance on testing set by plotting confusion matrix.

The datasets have two peak shape functions and two noise settings. Combining, we obtain 4 groups of datasets. Additionally, we introduce a mix dataset that mix the

previous four, by random selecting. Eventually, we obtained 5 datasets.

Out of the 5 datasets, we plot out the DNN model train test accuracy and loss, both on 7-way and 230-way. We plot the confusion matrix of the “mix” dataset for both DNN and CNN and 7-way and 230-way. The performance on testing set looks promising, which suggest that out synthetic data is perfectly self-consistent.

Chapter 5 RRUFF domain pipeline

In previous chapters, we generate synthetic XRD patterns based on ICSD crystal structures, which is considered as synthetic domain. The training and testing accuracy shows the consistence of the patterns within this single synthetic domain. In synthetic domain, the patterns follow certain predefined peak shapes and noise amplitudes, and need to be tested on another experimental pattern. In this chapter, we introduce RRUFF database, which is an open database that have high quality spectral data of minerals. By introducing the RRUFF data, which can be considered as an experimental domain, to be compared with ICSD synthetic domain.

Since the XRD patterns from Synthetic domain and RRUFF domain are not following the same peak positions, peak intensities, and peak shapes, we can fully test the performance of our model on a second domain, and to improve.

5.1 Converting RRUFF data

The data of RRUFF database can be directly downloaded from RRUFF websites[21]. For XRD patterns, there are six folders contains different kind of source files. To test our models, we need the same information as synthetic domain: features, both 7-ways and 230-ways labels, and RRUFF ids.

5.1.1 XY data and crystal system labels

The XY data, which is the patterns themselves, can be found in folder “XY_Processed”. Here shows an example of the files.

Acanthite__R080016-1__Powder__Xray_Data_XY_Processed__9701.txt

```
##NAMES=Acanthite
##RRUFFID=R080016
##IDEAL CHEMISTRY=Ag^1+^_2_S^2-^
##LOCALITY=Imiter mine, Djebel Sarhro, near Quarzazate, Anti Atlas Mountains, Morocco
##OWNER=RRUFF
##SOURCE=William W. Pinch
##DESCRIPTION=Group of black poorly formed crystals
##DIFFRACTION SAMPLE DESCRIPTION=Powder
##STATUS=The identification of this mineral has been confirmed only by X-ray diffraction
##URL=rruff.info/R080016
##CELL PARAMETERS=a: 4.2244(9) b: 6.9297(7) c: 7.8653(6) alpha: 90 beta: 99.65(1)
gamma: 90 volume: 226.99(4) crystal system: monoclinic
5.000000, 0.0000000E+00
5.010000, 2.1404000E-02
5.020000, 12.70571
……
```

Where, the first several headers provide some additional information about the pattern. We have the name of the mineral, the RRUFF ID, the diffraction type, the cell parameters and the crystal system.

We extract mineral name, RRUFF ID, crystal systems and XY data. After read files to list, we search for information based on its labels and stores to variable names. The XY

data is appended to lists. And we check the description to see if it is XRD.

[Step 1 2] Read rruff pattern to features and labels csv.ipynb

```
for line in powderFileLines:
    if "#" not in line and len(line.split()) > 1 and line[0]!=".":
        xList.append(float(line.replace(",", "").split()[0]))
        yList.append(float(line.replace(",", "").split()[1]))
    if "NAMES" in line:
        rruffName = line.split("=")[1].strip()
    if "RRUFFID" in line:
        rruffId = line.split("=")[1].strip()
    if "crystal system" in line:
        crysSys = (line.split()[-1]).strip().lower()
    if "DIFFRACTION SAMPLE DESCRIPTION" in line:
        if "Powder" in line.strip():
            powderJudgeBool=True
```

Notice that this file is lacking space group labels. To have both 7-way labels and 230-way labels, we have to extract 230-way labels from a different folder.

5.1.2 Space group labels

The space group labels can be extracted from folder “DIF”. Here shows an example of the files.

Abhurite__R060227-9__Powder__DIF_File__5543.txt

```
        Abhurite
Diffraction data computed using the structure from the paper listed below,
along with the cell parameters refined from single crystal data of R060227
```

Schnering H G, Nesper R, Pelshenke H

Zeitschrift fur Naturforschung B36 (1981) 1551-1560

Sn21Cl16(OH)14O6, das sogenannte basische zinn(II)-clorid

Locality: synthetic

CELL PARAMETERS: 9.9810 9.9810 43.8400 90.000 90.000 120.000

SPACE GROUP: R32

ATOM	X	Y	Z	OCCUPANCY	ISO(B)
Sn	0.00000	0.00000	0.30310	1.000	1.145
Sn	0.00000	0.00000	0.06860	1.000	1.192

……

Where, the first several headers contain additional information. The space group labels can be found after cell parameters.

The entries in this folder is not consistent with the XY data folder, so we have to make a comparison. Prior to that, we extract RRUFF ID, material name and space group labels from these files, and save that information to a table.

rruff DIF information summary.csv

R060227,Abhurite,155,R32

R070578,Acanthite,14,P21/n

R080016,Acanthite,14,P21/n

R040063,Actinolite,12,C2/m

R040064,Actinolite,12,C2/m

……

After we have a table of the information of RRUFF ID and space groups, we can easily

relate XY folder's RRUFF ID and crystal systems. To make the comparison, we first import the table as pandas DataFrame and try to locate the RRUFF ID from XY folder. Once we locate the RRUFF ID, we store the space group value and reexam if space group label and crystal system label matches.

[Step 1 2] Read rruff pattern to features and labels csv.ipynb

```
# Check if RRUFFID exist in DIF folder
if df1.loc[df1[0]==rruffId, 2].values[:].size == 1:
    print("Valid data: DIF folder")
    # Assign a variable to store space group
    sg = df1.loc[df1[0]==rruffId, 2].values[0]
    # Check if crystal system and space group matches
    if space_group_map_dict[sg] == crysDict[crysSys]:
        print("Valid data: XY DIF match")
```

Only when the space group label and crystal system label are both identified and secured, the pipeline will output the data to formatted files.

5.1.3 Total count of consistent labels

We only output those consistent space group labeled and crystal system labeled data. From a total of 1,361 crystal system labels and 2,875 space groups labels, we eventually secured 908 overlapped consistent labels of minerals, identified by same RRUFF ID.

The pie plot of the 908 RRUFF data class distribution is shown below. Inner circle stands for 7-way crystal system. Outer circle stands for 230-way space group. Not all

230 space groups appeared in RRUFF. Only the indexes of major space groups are printed. Major space groups stand for those space group classes who have more than 1% of datapoints of total RRUFF.

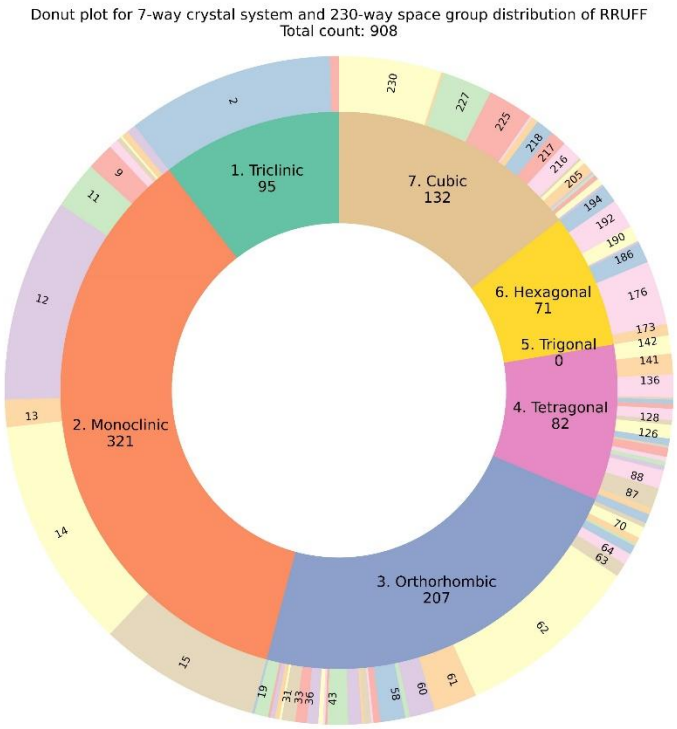

Figure 5.1 Class distribution of 908 RRUFF data.

5.1.4 Noise amplitude of RRUFF data

Prior to the export of the data, we manual use a simple rule to identify the noise amplitude of the patterns of RRUFF data. Instead of using any complicated algorithm to find the noise amplitude, we can simply use median value to find the most existence intensity and treat that as average noise amplitude. Based on the noise amplitude, we partition the RRUFF data to four groups.

1) RRUFF data of all noise amplitude: “noiseAll”

2) RRUFF data has noise amplitude smaller than 2%(intensity 20 out of 1000): “noise20”
3) RRUFF data has noise amplitude smaller than 1%(intensity 10 out of 1000): “noise10”
4) RRUFF data has noise amplitude smaller than 0.5%(intensity 5 out of 1000): “noise5”

The noise amplitude can be easily found by manipulating the intensity array.

[Step 1 2] Read rruff pattern to features and labels csv.ipynb

```
noiseAmp = int(np.median(origY)/np.max(origY)*1000)
```

The size of the different portion of RRUFF dataset is shown in Table 5-1.

	noiseAll	noise20	noise10	noise5
Data point	908	649	445	223

Table 5-1 The dataset size of different portion of RRUFF dataset.

And all dataset is stored in format. There are four files for the format: features.csv, labels7.csv, labels230.csv, and ids.csv. For each pattern, we append the information to the corresponding file.

```
with open(featuresFile, "a") as f:
    np.savetxt(f, arrayY.T, fmt='%d', delimiter=",")
with open(labels7File, "a") as f:
    np.savetxt(f, labels7Array, fmt='%d', delimiter=",")
with open(labels230File, "a") as f:
```

```
        np.savetxt(f, labels230_array, fmt='%d', delimiter=",")
with open(idsFile, "a") as f:
        print(f"{rruffId},{rruffName},{crysSys},{sg},{noiseAmp}", file=f)
```

For file “ids.csv”, for every RRUFF entries, we store the RRUFF ID, mineral name, crystal system, space group, and estimated noise amplitude. Here is an example of the contents of “ids.csv”.

ids.csv

```
R080016,Acanthite,monoclinic,14,40
R040063,Actinolite,monoclinic,12,6
R040064,Actinolite,monoclinic,12,9
R050025,Actinolite,monoclinic,12,21
R050336,Actinolite,monoclinic,12,16
R060041,Actinolite,monoclinic,12,18
R060045,Actinolite,monoclinic,12,12
R040130,Adamite,orthorhombic,58,7
……
```

5.2 Testing models on RRUFF data

After we have the RRUFF dataset, we can apply models on the features of RRUFF dataset and to check the accuracy with RRUFF labels. Both DNN and CNN model can be tested on RRUFF with the same script, in which we need to rebuild Dataset class and use DataLoader. We iterate through RRUFF folders of different noises first, and then iterate through different epochs of models. The models are saved in .pt files every 5 epochs shown in Chapter 4, and can be loaded directly.

5.2.1 DNN models performance

DNN models trained by five datasets of 7-way classification is applied on RRUFF data.

1) "171k_ps1"

The performance of DNN model trained by ICSD 171k peak shape 1 dataset is shown below. The accuracy of all RRUFF dataset is around 25%. As the RRUFF dataset noise amplitude decrease, the performance increase. For RRUFF noise5, the highest accuracy is around 50%.

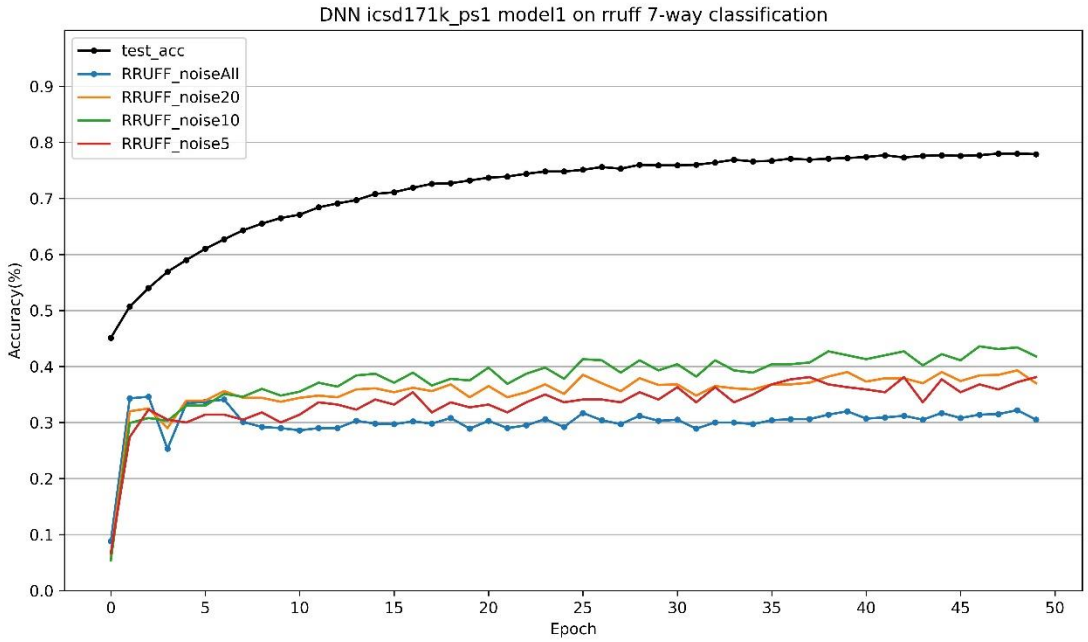

Figure 5.2 DNN trained on dataset peak shape 1 with no noise 7-way testing accuracy and RRUFF domain accuracy.

2) “171k_ps1_noise20”

The performance of DNN model trained by ICSD 171k peak shape 1 with noise20 is shown below. The accuracy of all RRUFF dataset is around 45%. However, compared with the dataset with no noise, the accuracy on RRUFF noise5 explicitly decrease.

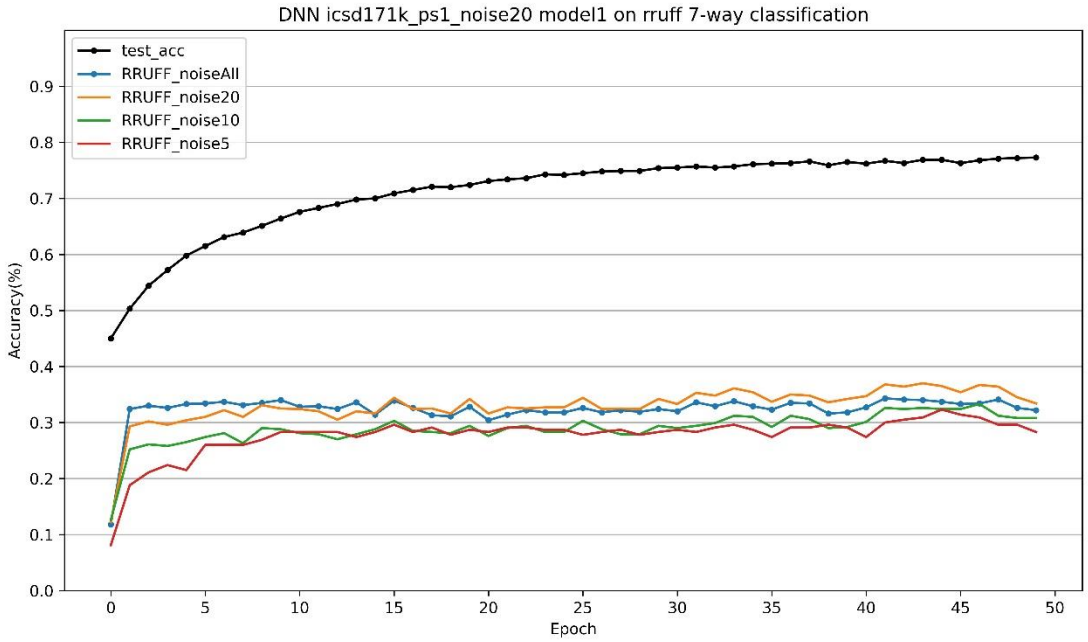

Figure 5.3 DNN trained on dataset peak shape 1 with noise 7-way testing accuracy and RRUFF domain accuracy.

3) “171k_ps2”

The performance of ICSD 171k peak shape 2 with no noise dataset is shown below. The accuracy on all RRUFF dataset is around 35%. Same observation can be found compare with peak shape 1, as the RRUFF noise amplitude decrease, the performance improves.

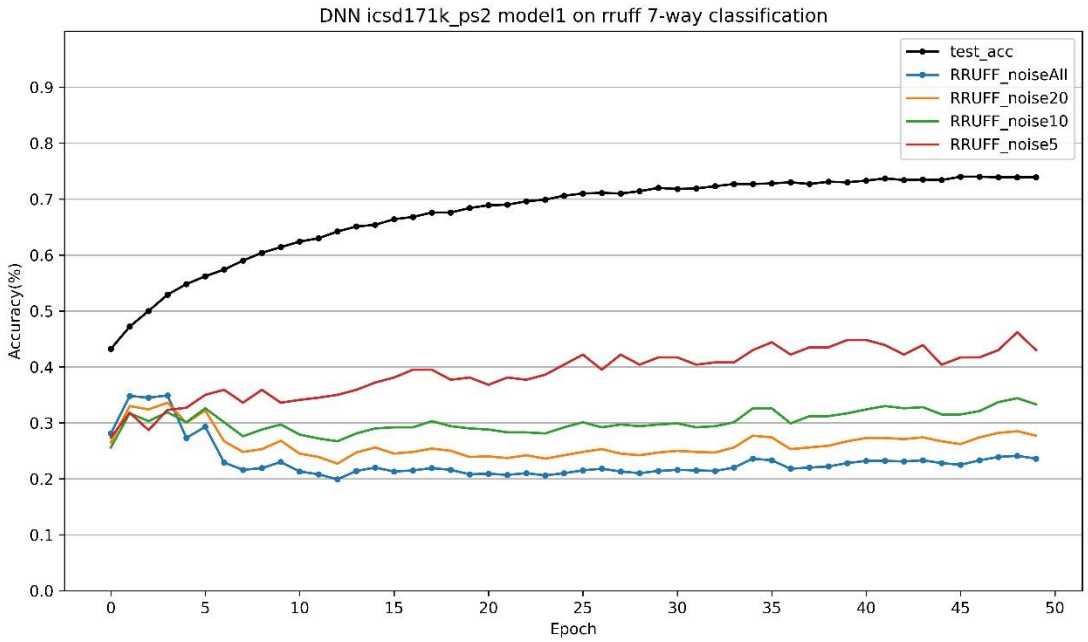

Figure 5.4 DNN trained on dataset peak shape 2 with no noise 7-way testing accuracy and RRUFF domain accuracy.

4) “171k_ps2_noise20”

The performance of ICSD 171k peak shape 2 with noise20 dataset is shown below. The accuracy on all RRUFF dataset is around 40%. Same observation can be found compare with peak shape 1 with noise 20, that RRUFF noise20 has the highest accuracy, and RRUFF noise5 has the lowest, which is a direct inverse with ICSD 171k with no noise.

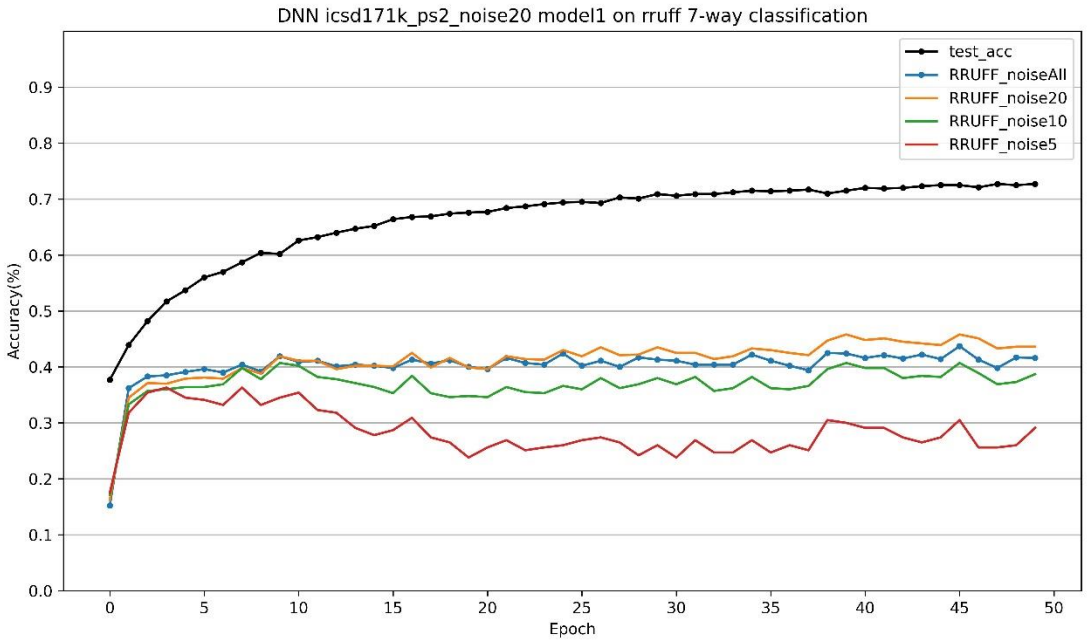

Figure 5.5 DNN trained on dataset peak shape 2 with noise 7-way testing accuracy and RRUFF domain accuracy.

The RRUFF accuracy results are shown in table.

DNN 7-way	Ps1	Ps1+noise20	Ps2	Ps2+noise
Train test acc.	> 70%	> 70%	> 70%	> 70%
RRUFF all	30.5%	32.2%	23.6%	41.6%
RRUFF noise20	37%	33.4%	27.7%	43.6%
RRUFF noise5	38.1%	28.3%	43%	29.1%

Table 5-2 Different models’ RRUFF accuracy based on different dataset.

A comparison between four models shows interesting results:

a) Comparison between peak shapes suggest that PS2 has lower performance on RRUFF than PS1, where PS2 has smaller FWHM Gauss peak shape than PS1. According to our comparison between synthetic ICSD pattern and RRUFF pattern, PS2 has the closest peak shapes. However, with noise added, PS2 has higher performance than PS1.

b) Comparison between no noise and noise20 suggests that for either PS1 or PS2, the overall performance on RRUFF increases, which suggests that adding noises can improve the performance of the model.

c) If we focus on noise20 datasets, we observe better performance on RRUFF noise20, and worse performance on RRUFF noise5, which is opposite than no noise datasets. This observation suggests that if the training set is artificially added random noise, the model tend to overfit to similar noise amplitude and fails to recognize no noise patterns.

Combining both peak shapes and both noises, the DNN model trained by 171k_mix dataset shows better performance on RRUFF, around 58% for all noise amplitude, and a 63% on partitioned RRUFF noise20 and noise10.

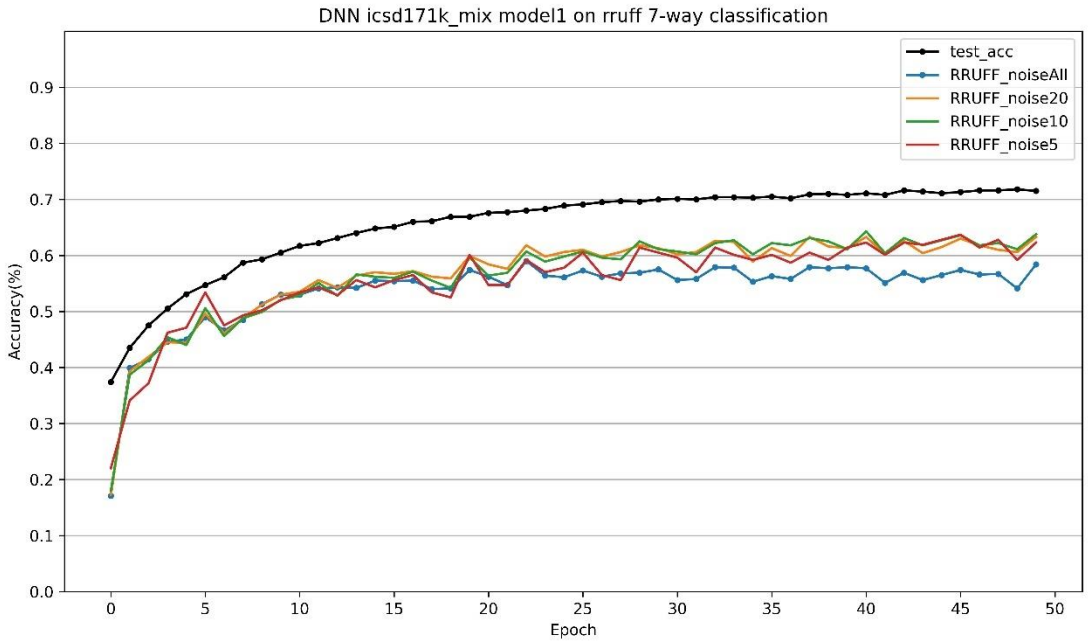

Figure 5.6 DNN trained on dataset mix 7-way testing accuracy and RRUFF domain accuracy.

The RRUFF accuracy results are shown in table.

DNN 7-way	Ps1	Ps1+noise20	Ps2	Ps2+noise	Mix
Train test acc.	> 90%	> 90%	> 90%	> 90%	> 70%
RRUFF all	30.5%	32.2%	23.6%	41.6%	58.4%

Table 5-3 Train test accuracy and RRUFF accuracy of DNN 7-way on different dataset.

The performance of model 171k_mix is significantly higher than all previous four datasets, while keeping the same size, and is only randomly selecting out of them. This lets to an amazing conclusion, that diversified dataset greatly helps improve the performance of our model. Based on this conclusion, we only train DNN 230-way based on the mix dataset.

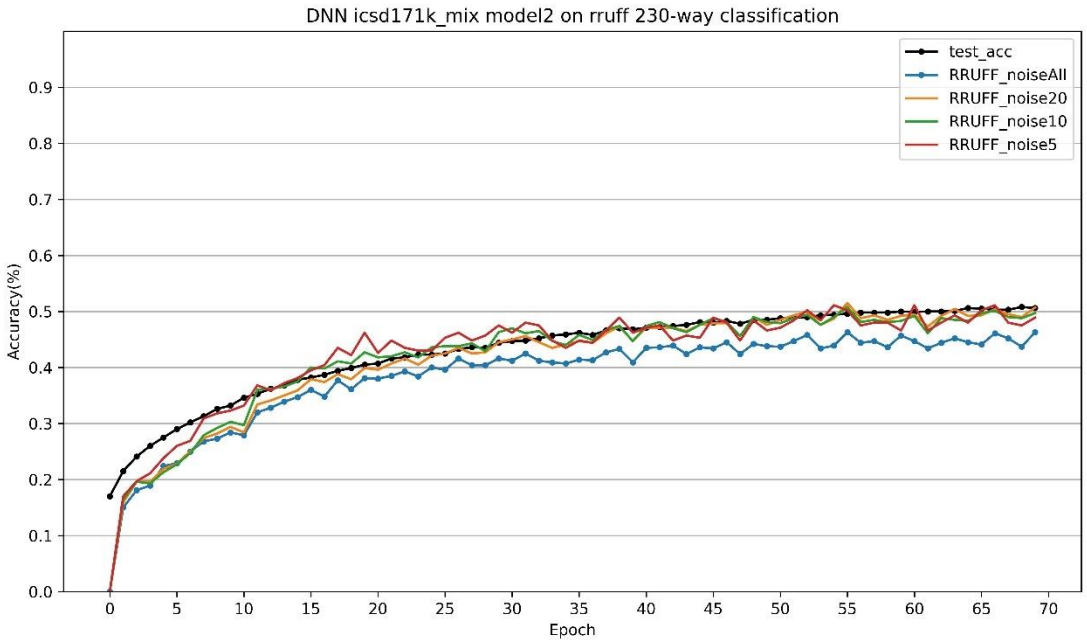

Figure 5.7 DNN trained on dataset mix 230-way testing accuracy and RRUFF domain accuracy.

The 230-way accuracy is summarized below.

DNN 230-way	Mix
Train test acc.	> 50%
RRUFF all	46.3%

Table 5-4 Train test accuracy and RRUFF accuracy of DNN 230-way on dataset "Mix".

5.2.2 CNN models performance

Following the conclusion from DNN models, we only test CNN performance with 171k_mix dataset. The mixed dataset has the highest potential for a model to learn. For 7-way classification, the CNN reaches around 50% on RRUFF dataset, which is lower than DNN counterparts.

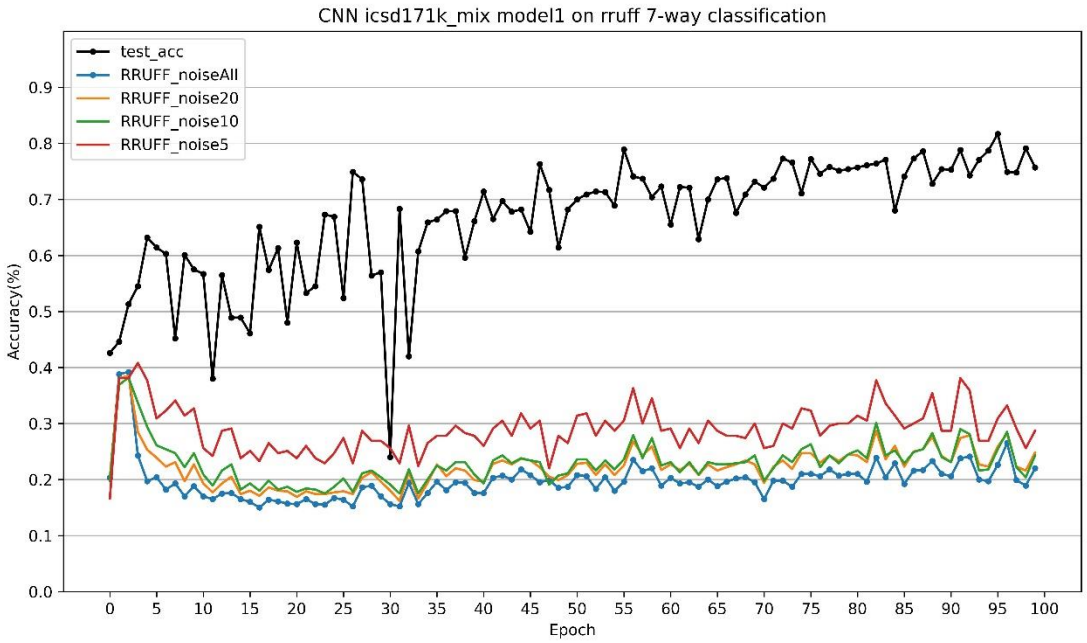

Figure 5.8 CNN trained on dataset "mix" 7-way testing accuracy and RRUFF domain accuracy.

For 230-way classification, the CNN only reaches less than 20% on RRUFF dataset, which is significantly lower that DNN counterparts.

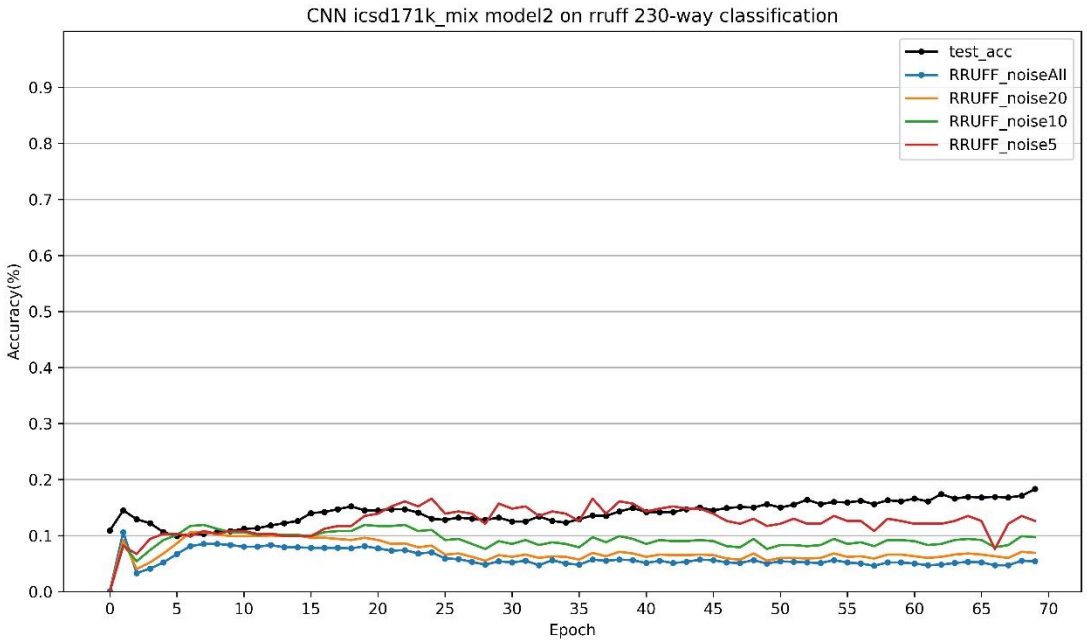

Figure 5.9 CNN trained on dataset mix 230-way testing accuracy and RRUFF domain accuracy.

The CNN performance is shown in table.

CNN	Mix 7-way	Mix 230-way
Train test acc.	>70%	~ 20%
RRUFF all	~20%	< 10%

Table 5-5 Train test accuracy and RRUFF accuracy of CNN 7 and 230-way on dataset mix.

5.2.3 Comparison between DNN and CNN models

A comparison between DNN and CNN is shown in table.

RRUFF all	DNN	CNN
7-way	58.4%	~20%
230-way	46.3%	<10%

Table 5-6 DNN and CNN 7 and 230-way RRUFF domain accuracy.

Figure 5.6 and Figure 5.8 showed that both DNN and CNN are sensitive to RRUFF noise amplitude. The accuracies of different partitions of RRUFF datasets are separated. There are obvious gaps between different RRUFF partitions in both Figure 5.6 and Figure 5.8. Based on the performance on different partitions of RRUFF, DNN has better generalizability.

The accuracy of CNN is lower than DNN, for 230-way classification, the CNN accuracy is significantly lower, which suggest than CNN is taking shortcuts due to 171k_mix dataset is not diversified enough. In figure, when testing performance of

CNN 230-way on RRUFF, as the epoch increase and testing accuracy increase, we observe a decrease on RRUFF dataset, which suggest that the synthetic domain cannot be learned any more to predict RRUFF domain. Our pipeline generated synthetic patterns are behaving differently that RRUFF patterns.

5.3 Understanding DNN models on RRUFF data

To better understand the performance of our model on RRUFF domain, we pick some examples from RRUFF and check the prediction accuracy. We use DNN models for both 7-way and 230-way to visualized, since DNN models are having superior performance on RRUFF domain compared with CNN models.

5.3.1 Performance on cerussite

The information of cerussite is shown in Table 5-7.

Name	Formula	Crystal system	Space group
Cerussite	Pb(CO3)	Orthorhombic: 3	P m c n: 61

Table 5-7 Information of cerussite.

There are 6 RRUFF patterns as shown in Figure. One of the same minerals in ICSD is 247490. The synthetic generated XRD is also plotted in Figure 5.10.

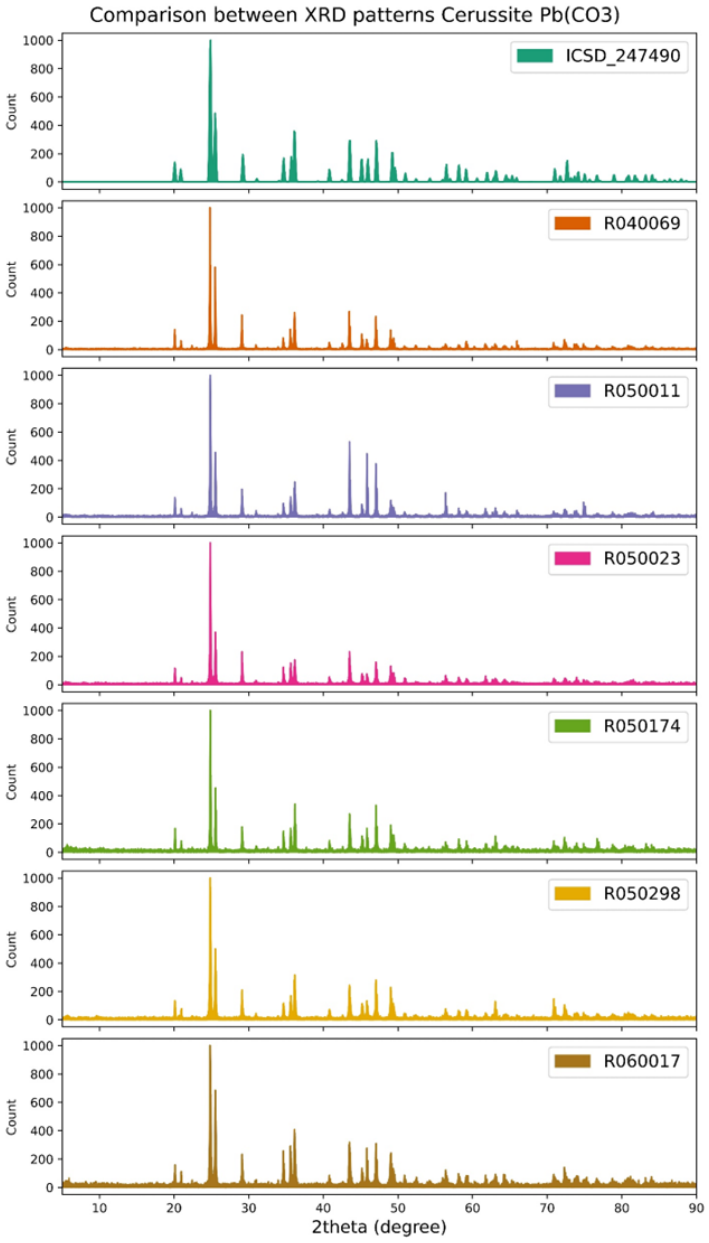

Figure 5.10 XRD patterns comparison of cerussite.

The results of both 7-way and 230-way predictions of DNN is shown in Table 5-8.

RRUFF ID	Crystal System	Label	Predict	Space group	Label	Predict
R040069	Orthorhombic	3	3	P m c n	61	61
R050011	Orthorhombic	3	3	P m c n	61	61
R050023	Orthorhombic	3	3	P m c n	61	61
R050174	Orthorhombic	3	3	P m c n	61	61
R050298	Orthorhombic	3	3	P m c n	61	61
R060017	Orthorhombic	3	3	P m c n	61	61

Table 5-8 Results of cerussite on DNN models.

For cerussite, the accuracy for both crystal system and space group prediction is 100%. Figure shows that ICSD synthetic pattern has similar peak positions and peak intensities as RRUFF patterns.

5.3.2 Performance on whewellite

The information of whewellite is shown in Table 5-9.

Name	Formula	Crystal system	Space group
whewellite	Ca(C2O4)H2O	Monoclinic: 2	P 1 21/c 1: 14

Table 5-9 Information of whewellite.

There are 2 RRUFF patterns as shown in Figure. One of the same minerals in ICSD is 434200.

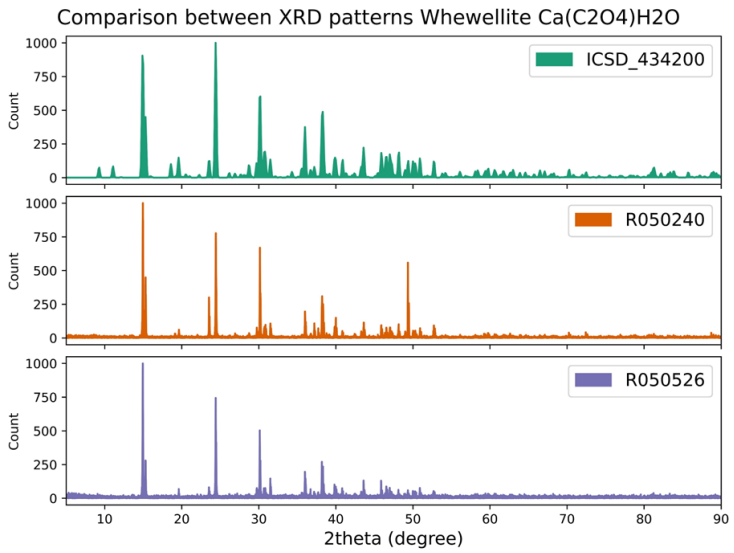

Figure 5.11 XRD patterns comparison of whewellite.

The results of both 7-way and 230-way predictions of DNN is shown in Table 5-10.

RRUFF ID	Crystal System	Label	Predict	Space group	Label	Predict
R050240	Monoclinic	2	2	P 1 21/c 1	14	14
R050526	Monoclinic	2	2	P 1 21/c 1	14	14

Table 5-10 Results of whewellite on DNN models.

For whewellite, the accuracy for both crystal system and space group prediction is 100%. However, figure shows that ICSD synthetic pattern has similar peak positions and peak intensities as RRUFF patterns only at lower angles (< 40 degrees). For high angles (> 40 degrees), there are varied peak positions and peak positions. This might suggest that the DNN model learnt to make dominant decision based on lower angles,

and make minor decisions based on higher angles.

5.3.3 Performance on albite

The information of albite is shown in Table 5-11.

Name	Formula	Crystal system	Space group
albite	Na(AlSi3O8)	Triclinic: 1	C -1 : 2

Table 5-11 Information of albite.

There are 5 RRUFF patterns as shown in Figure 5.12. One of the same minerals in ICSD is 009829.

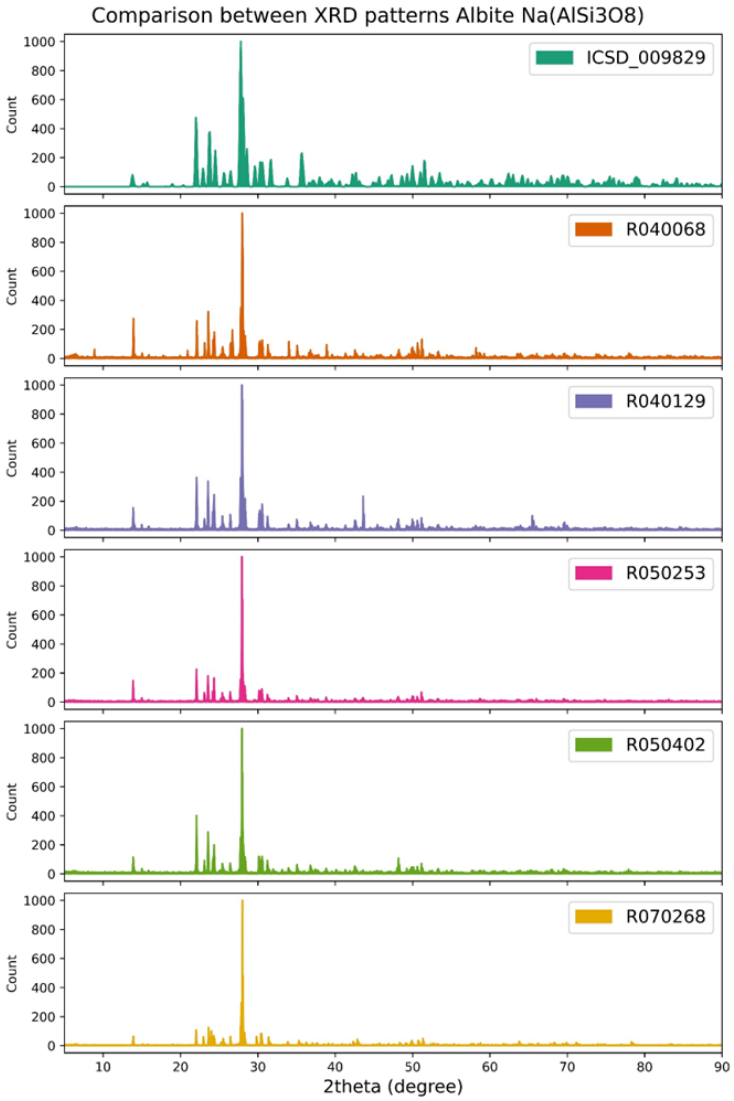

Figure 5.12 XRD patterns comparison of albite.

The results of both 7-way and 230-way predictions of DNN is shown in Table 5-12.

RRUFF ID	Crystal System	Label	Predict	Space group	Label	Predict
R040068	Triclinic	1	2	C -1	2	2
R040129	Triclinic	1	1	C -1	2	2
R050253	Triclinic	1	1	C -1	2	15
R050402	Triclinic	1	1	C -1	2	2
R070268	Triclinic	1	5	C -1	2	166

Table 5-12 Results of albite on DNN models.

For albite, the accuracy for crystal system and space group are 60%. The figure shows that ICSD synthetic pattern has similar peak positions and peak intensities as RRUFF patterns only at lower angles (< 40 degrees). For high angles (> 40 degrees), there are varied peak positions and peak positions but are not obvious. However, the result shows 2 wrong predicted crystal system, one is monoclinic and one is tetragonal. And also 2 wrong space group predictions. However, for R070268, we notice that space group 166 belongs to crystal system 5 trigonal, which indicates a total miss prediction for both 7-way and 230-way simultaneously. Simultaneous miss prediction is due to wrong classification during training, which is caused by lacking sufficient data on similar pattern of this class, or overfitting to another class in the training data.

5.3.4 Performance on wulfenite

The information of wulfenite is shown in Table 5-13.

Name	Formula	Crystal system	Space group
wulfenite	PbMoO4	Tetragonal: 4	I 41/a: 88

Table 5-13 Information of wulfenite.

There are 3 RRUFF patterns as shown in Figure 5.13. One of the same minerals in ICSD is 028951.

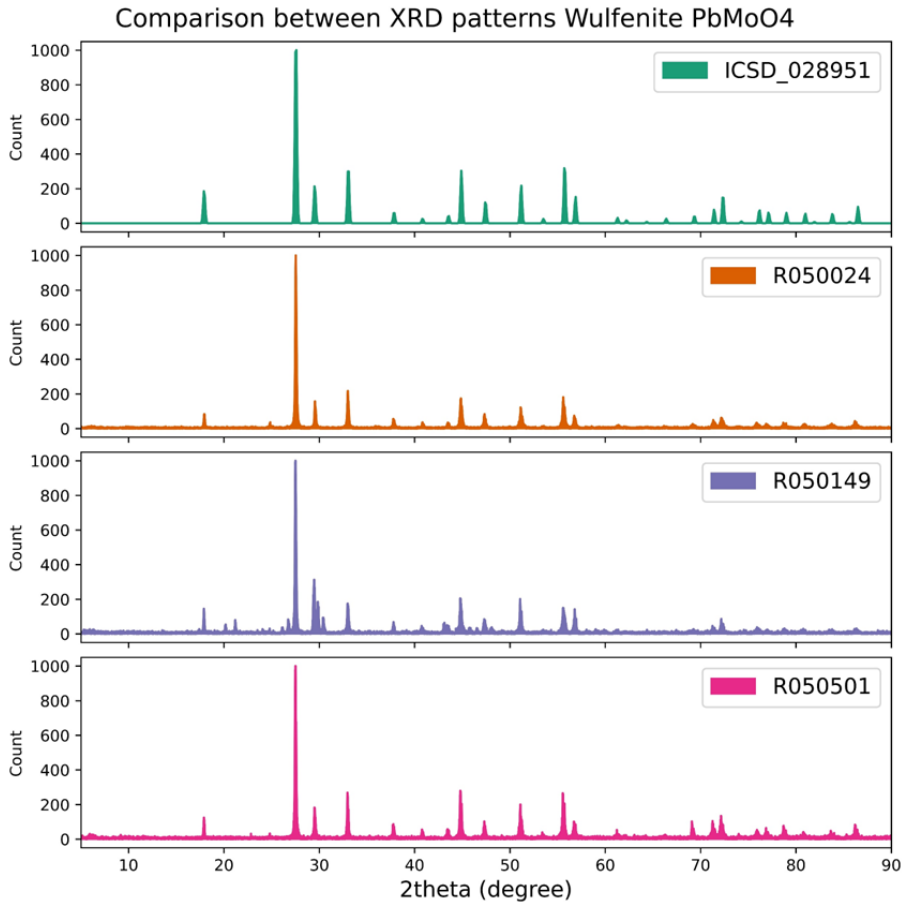

Figure 5.13 XRD patterns comparison of wulfenite.

The results of both 7-way and 230-way predictions of DNN is shown in Table 5-14.

RRUFF ID	Crystal System	Label	Predict	Space group	Label	Predict
R050024	Tetragonal	4	4	I 41/a	88	88
R050149	Tetragonal	4	2	I 41/a	88	15
R050501	Tetragonal	4	4	I 41/a	88	88

Table 5-14 Results of wulfenite on DNN models.

For wulfenite, the accuracy for both crystal system and space group prediction is 66%. The figure shows that ICSD synthetic pattern has obvious different peaks compared RRUFF patterns. Between 20 to 25 degrees, all RRUFF patterns have additional small

peaks that ICSD doesn't have. Also, clear difference can be observed at 30 degrees. The only failed prediction predict crystal system 2 monoclinic, and predict space group 15, which is belong to crystal system 2 monoclinic. The failed predictions are consistent with themselves, which might suggest overfitting to a different pattern that belong to crystal system 2 monoclinic, similarly as a previous example of albite.

5.4 Selective adopting consistent DNN predictions

As we have seen in Section 5.3, during the examples, we observed inconsistent predictions between 7-way and 230-way. The inconsistent here represents that the 230-way DNN predict a space group that is not within the crystal system prediction of the 7-way DNN. The inconsistent here is inevitable since we are training 7-way and 230-way DNN models separately. Although, the architectures of 7-way and 230-way models are the same, where the architectures share exact same hidden layers, the learned weights can be quite different due to different length of output classes.

Therefore, we propose an algorithm to only accept those predictions that have consistent 7-way and 230-way predictions. To be specific, for one entry, or pattern in RRUFF, we apply 7-way model and 230-way model, and obtained two predictions. If the space group prediction is within the crystal system prediction, as the mapping relation shown in Table 1-1, we accept both results, and reevaluate the total count of entries in RRUFF and the DNN models performance. The mapping relation is stores in dictionary as shown.

```
space_group_map_dict = {}
for i in range(1, 3):
    space_group_map_dict[i] = 1
for i in range(3, 16):
    space_group_map_dict[i] = 2
for i in range(16, 75):
    space_group_map_dict[i] = 3
for i in range(75, 143):
    space_group_map_dict[i] = 4
for i in range(143, 168):
    space_group_map_dict[i] = 5
for i in range(168, 195):
    space_group_map_dict[i] = 6
for i in range(195, 231):
    space_group_map_dict[i] = 7
```

We can convert space group prediction to the corresponding crystal system, to compare with actual crystal system prediction. When they are consistent, we update the total count and to count for tor the labels matching. We implemented the algorithm to the code as shown below.

```
for i, (x, y7, y230, z) in enumerate(loader):
    x = x.float()
    if config_model == "CNN":
        x = x.unsqueeze(1)


    y_pred7 = model1(x).detach()
```

```
    y_pred230 = model2(x).detach()


    pred7 = torch.argmax(y_pred7, dim=-1).numpy().tolist()[0]
    pred230 = torch.argmax(y_pred230, dim=-1).numpy().tolist()[0]


    label7 = y7.detach().numpy().tolist()[0]
    label230 = y230.detach().numpy().tolist()[0]
    count0 += 1


    if space_group_map_dict[int(pred230)+1] == int(pred7)+1:
        total_count += 1
        if pred230 == label230:
            count_230way += 1
        if pred7==label7:
            count_7way += 1


print(f" Total:{total_count}, 230-way acc:{count_230way/total_count:.3f}, 7-way acc:{count_7way/total_count:.3f}")
```

During this algorithm, the total count is decreasing, since we neglect the results of inconsistent predictions. The total count and classification accuracies are shown in Table 5-15 .

DNN models	Total count	Crystal system	Space group
All RRUFF patterns	908	58.4%	46.3%
Consistent predictions	521	77.0%	63.5%
Accuracy improvement	-387	+18.6%	+17.2%

Table 5-15 Statistics of selective adopting consistent predictions.

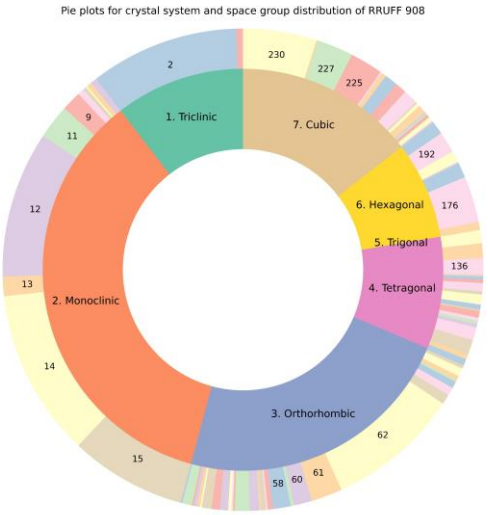

Figure 5.14 Pie plot for RRUFF all 908 patterns.

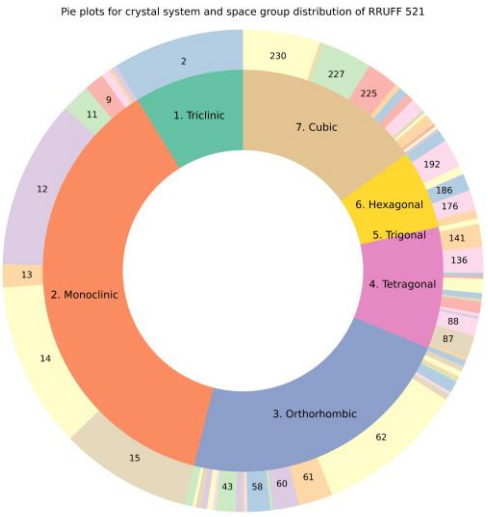

Figure 5.15 Pie plot for RRUFF after cleaning inconsistent results to 512 patterns.

We achieve considerable accuracy increase by filter out inconsistent predictions. We finally achieved 77% on 7-way classification and 63.5% on 230-way classification on RRUFF experimental data. Vecsei et al reported 82% at the expanse of having half of the experimental data dropped. We are having 77% with less than half data dropped.

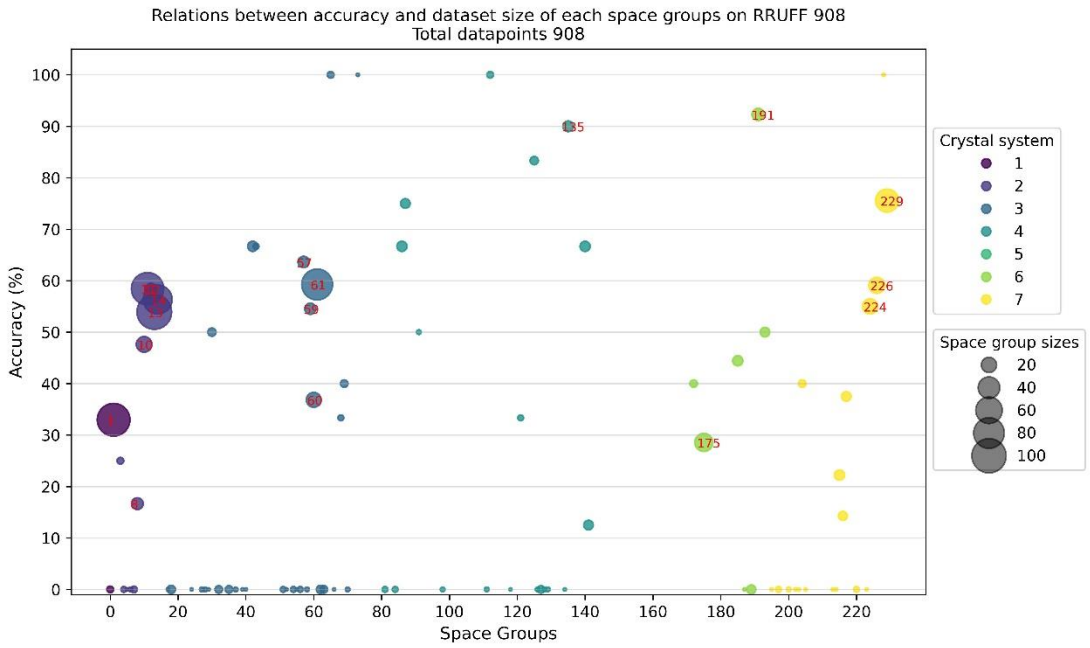

Figure 5.16 Relations between accuracy and datapoints of RRUFF all 908 patterns.

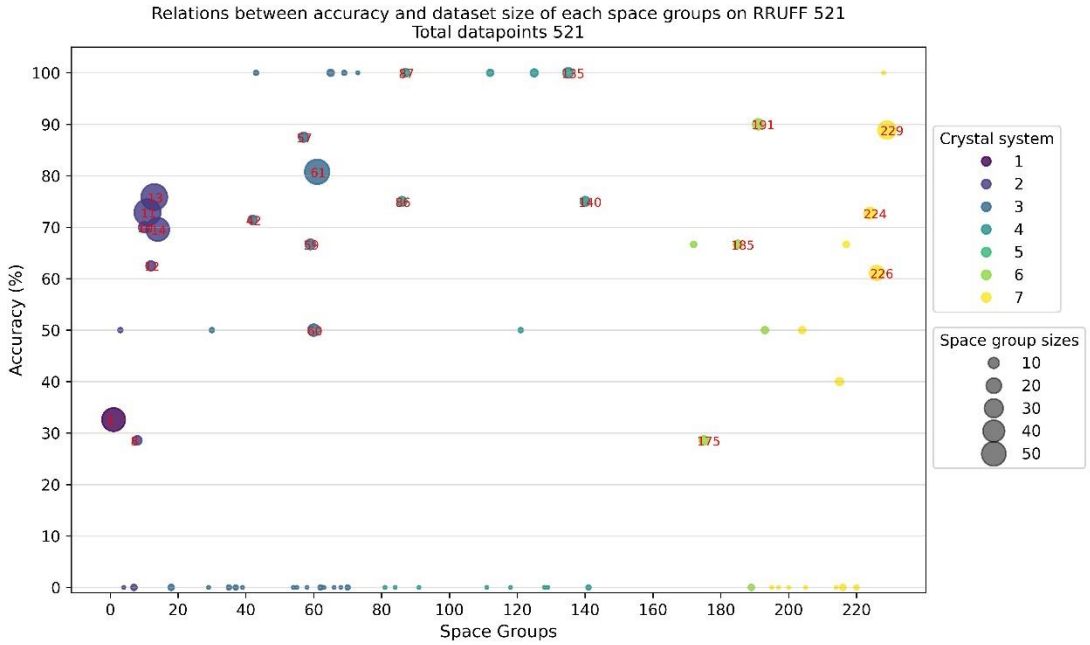

Figure 5.17 Relations between accuracy and datapoints of RRUFF after cleaning inconsistent result patterns to 521.

A comparison between Figure 5.16 and Figure 5.17 shows that the distribution of all datapoints. Here we could visualize the reason why accuracy increases. Some space groups are entirely dropped. We could observe clear increase accuracy on some space groups labeled by index.

5.5 Conclusion

In this chapter, we tested the model at a second domain, the RRUFF domain. As we trained the model in ICSD synthetic domain, the performance shown in RRUFF domain is the key of our goal: to train using synthetic data, while been able to predict real experimental pattern. Our highest accuracy on 908 RRUFF patterns is 77% on 7-way classification, and 63.5% on 230-way classification at the cost of rejecting 387 predictions.

Also, we identified the pros and cons of DNN and CNN. Although, DNN models outperform CNN models, but CNN has the highest potential to generalize and without been affected by noises in experimental patterns.

The key to improve the performance of the model goes all the way back the Chapter 1, the generation of synthetic data. With only two peak shapes and two noise amplitude applied, the dataset is not diversified enough for the model to learn the math of symmetries behind the patterns.

However, initial trials on few examples demonstrate the huge potential of our model to predict the symmetry of crystals by only learning from synthetic patterns. With more methods and improvements to the dataset, the model will be better generalized and greatly improved.

Chapter 6 Future work

6.1 Domain adaptation

By domain adaptation, we are referring to synthetic patterns' domain and experimental patterns' domain. Although both synthetic and experimental XRD patterns are both reflecting miller indices planes at certain angles, only experimental ones contain all real-world instabilities, such as measurement noise, or peak shape functions caused by X-ray beam property.

The fundamental reason for training with synthetic patterns, is due to the huge demands of large data. Large amount of experimental data means huge financial cost, while with crystal solution, we could theoretically calculate big data with only computational cost.

While we are only using computational cost with fewer financial cost, we face gaps between synthetic patterns and experimental pattern. Keep in mind, we are aiming to apply classification to real experimental data. Therefore, domain adaptation is necessary additional cost to compensate for the fact that we saved cost while only using synthetic data. A successful domain adaptation means we could use the same group of synthetic data to adapt to all different experimental XRD databases.

So far, in chapter 6, we demonstrate the performance on RRUFF database without

domain adaptation, and we already observed acceptable accuracies.

6.2 Lattice augmentation

The augmentation pipeline of 1D pattern aims to provide more training set for deep learning algorithms to improve its accuracy, as well as enhance the reliability of the generated synthetic data. The augmentation is based on initial generated 171k synthetic pattern, each corresponding to single crystal structure solution data. The augmentation is based on the variability of several self-defined parameters in the generation pipeline.

1) The change of wavelength and unit cell dimension will affect peak positions. With slightly controlled variation, the peak position will shift.
2) The factors for intensities will be changed to generate different integrated intensities. The absorption factor and preferred orientation will affect peak integrated intensities. The effects are not set to be linear along 2θ,thus the resulted augmented pattern will show different intensities than initial ones.
3) The peak shape functions are Pseudo-Voigt and its linear factor will be changed to form a different shape. Background noise will also be considered.

With already different integrated intensities and different distribution of shape function, the synthetic pattern can simulate various different pattern describing the same crystal. The change of peak shift, peak intensity and peak shape functions will be randomly selected and combined, to simulate different kind of XRD pattern variation.

Bibliography

1. Wehrenberg, C.E. et al, 2017. In situ X-ray diffraction measurement of shock-wave-driven twinning and lattice dynamics. Nature 550, 496–499.
2. Polsin, D.N. et al, 2017. Measurement of Body-Centered-Cubic Aluminum at 475 GPa. Physical Review Letters 119.
3. White et al, 2015. The Linac Coherent Light Source. Journal of Synchrotron Radiation 22, 472–476.
4. Park, W.B. et al, 2017. Classification of crystal structure using a convolutional neural network. IUCrJ 4, 486–494.
5. Ziletti, A. et al, 2018. Insightful classification of crystal structures using deep learning. Nature Communications 9.
6. Vecsei, P.M. et al, 2019. Neural network based classification of crystal symmetries from x-ray diffraction patterns. Physical Review B 99.
7. Oviedo, F et al, 2019. Fast and interpretable classification of small X-ray diffraction datasets using data augmentation and deep neural networks. npj Computational Materials 5.
8. Kaufmann, K et al, 2020. Crystal symmetry determination in electron diffraction using machine learning. Science 367, 564–568.
9. Wang, H. et al, 2020. Rapid Identification of X-ray Diffraction Patterns Based on Very Limited Data by Interpretable Convolutional Neural Networks. Journal of Chemical Information and Modeling 60, 2004–2011.

10. Lee, J.-W et al, 2020. A deep-learning technique for phase identification in multiphase inorganic compounds using synthetic XRD powder patterns. Nature Communications 11.
11. Lee, J.-W. et al, 2021. A data-driven XRD analysis protocol for phase identification and phase-fraction prediction of multiphase inorganic compounds. Inorganic Chemistry Frontiers 8, 2492–2504.
12. Li, Y. et al, 2021. Composition based crystal materials symmetry prediction using machine learning with enhanced descriptors. Computational Materials Science 198, 110686.
13. Dong, H. et al, 2021. A deep convolutional neural network for real-time full profile analysis of big powder diffraction data. npj Computational Materials 7.
14. Chakraborty, A. et al, 2022. A deep crystal structure identification system for X-ray diffraction patterns. The Visual Computer 38, 1275–1282.
15. .International Tables for Crystallography (2006). Vol. G. ch. 2.2, pp. 20-36
16. Leverhulme Research Centre for Functional Materials Design, ICSDClient, (2021), GitHub repository, https://github.com/lrcfmd/ICSDClient.
17. ICSD WEB. https://icsd.fiz-karlsruhe.de.
18. Gainsford, G.J. et al, Redetermination of the borax structure from laboratory X-ray data at 145 K. Acta Crystallographica, Section E: Structure Reports Online (2008) 64, (5) p. i24-i25.
19. V. Pecharesky et al, 2005, Fundamentals of Powder Diffraction and Structural Characterization of Materials. Springer Science & Business Media.

20. Li Deng and Dong Yu, 2014, "Deep Learning: Methods and Applications", Foundations and Trends in Signal Processing: Vol. 7: No. 3–4, pp 197-387.
21. Lafuente B, Downs R T, Yang H, Stone N (2015) The power of databases: the RRUFF project. In: Highlights in Mineralogical Crystallography, T Armbruster and R M Danisi, eds. Berlin, Germany, W. De Gruyter, pp 1-30

Appendix A

H 1 HF 0.489918 20.6593 0.262003 7.74039 0.196767 49.5519 0.049879 2.20159 0.001305 0.000 0.17 0.000
He 2 RHF 0.873400 9.10370 0.630900 3.35680 0.311200 22.9276 0.178000 0.982100 0.006400 0.001 1.01 0.000
Li 3 RHF 1.12820 3.95460 0.750800 1.05240 0.617500 85.3905 0.465300 168.261 0.037700 0.005 2.00 0.001
Be 4 RHF 1.59190 43.6427 1.12780 1.86230 0.539100 103.483 0.702900 0.542000 0.038500 0.003 0.56 0.001
B 5 RHF 2.05450 23.2185 1.33260 1.02100 1.09790 60.3498 0.706800 0.140300 -0.19320 0.002 0.75 0.001
C 6 RHF 2.31000 20.8439 1.02000 10.2075 1.58860 0.568700 0.865000 51.6512 0.215600 0.006 2.00 0.001
N 7 RHF 12.2126 0.005700 3.13220 9.89330 2.01250 28.9975 1.16630 0.582600 -11.529 0.007 0.11 0.002
O 8 RHF 3.04850 13.2771 2.28680 5.70110 1.54630 0.323900 0.867000 32.9089 0.250800 0.001 0.22 0.000
F 9 RHF 3.53920 10.2825 2.64120 4.29440 1.51700 0.261500 1.02430 26.1476 0.277600 0.001 0.01 0.000
Ne 10 RHF 3.95530 8.40420 3.11250 3.42620 1.45460 0.230600 1.12510 21.7184 0.351500 0.002 0.25 0.001
Na 11 RHF 4.76260 3.28500 3.17360 8.84220 1.26740 0.313600 1.11280 129.424 0.676000 0.009 0.13 0.002
Mg 12 RHF 5.42040 2.82750 2.17350 79.2611 1.22690 0.380800 2.30730 7.19370 0.858400 0.015 0.08 0.003
Al 13 RHF 6.42020 3.03870 1.90020 0.742600 1.59360 31.5472 1.96460 85.0886 1.11510 0.018 2.00 0.005
Si 14 RHF 6.29150 2.43860 3.03530 32.3337 1.98910 0.678500 1.54100 81.6937 1.14070 0.009 2.00 0.002
P 15 RHF 6.43450 1.90670 4.17910 27.1570 1.78000 0.526000 1.49080 68.1645 1.11490 0.003 0.65 0.001
S 16 RHF 6.90530 1.46790 5.20340 22.2151 1.43790 0.253600 1.58630 56.1720 0.866900 0.005 0.67 0.002
Cl 17 RHF 11.4604 0.010400 7.19640 1.16620 6.25560 18.5194 1.64550 47.7784 -9.5574 0.007 0.78 0.003
Ar 18 RHF 7.48450 0.907200 6.77230 14.8407 0.653900 43.8983 1.64420 33.3929 1.44450 0.029 2.00 0.006
K 19 RHF 8.21860 12.7949 7.43980 0.774800 1.05190 213.187 0.865900 41.6841 1.42280 0.011 0.90 0.005
Ca 20 RHF 8.62660 10.4421 7.38730 0.659900 1.58990 85.7484 1.02110 178.437 1.37510 0.016 0.99 0.006
Sc 21 RHF 9.18900 9.02130 7.36790 0.572900 1.64090 136.108 1.46800 51.3531 1.33290 0.014 1.07 0.006
Ti 22 RHF 9.75950 7.85080 7.35580 0.500000 1.69910 35.6338 1.90210 116.105 1.28070 0.014 2.00 0.006

V 23 RHF 10.2971 6.86570 7.35110 0.438500 2.07030 26.8938 2.05710 102.478 1.21990 0.014 2.00 0.005
Cr 24 RHF 10.6406 6.10380 7.35370 0.392000 3.32400 20.2626 1.49220 98.7399 1.18320 0.011 2.00 0.004
Mn 25 RHF 11.2819 5.34090 7.35730 0.343200 3.01930 17.8674 2.24410 83.7543 1.08960 0.009 2.00 0.004
Fe 26 RHF 11.7695 4.76110 7.35730 0.307200 3.52220 15.3535 2.30450 76.8805 1.03690 0.011 0.08 0.004
Co 27 RHF 12.2841 4.27910 7.34090 0.278400 4.00340 13.5359 2.34880 71.1692 1.01180 0.013 0.08 0.004
Ni 28 RHF 12.8376 3.87850 7.29200 0.256500 4.44380 12.1763 2.38000 66.3421 1.0341 0.014 0.08 0.004
Cu 29 RHF 13.3380 3.58280 7.16760 0.247000 5.61580 11.3966 1.67350 64.8126 1.19100 0.015 0.08 0.005
Zn 30 RHF 14.0743 3.26550 7.03180 0.233300 5.16520 10.3163 2.41000 58.7097 1.30410 0.016 0.08 0.005
Ga 31 RHF 15.2354 3.06690 6.70060 0.241200 4.35910 10.7805 2.96230 61.4135 1.71890 0.025 0.08 0.008
Ge 32 RHF 16.0816 2.85090 6.37470 0.251600 3.70680 11.4468 3.68300 54.7625 2.13130 0.024 0.08 0.008
As 33 RHF 16.6723 2.63450 6.07010 0.264700 3.43130 12.9479 4.27790 47.7972 2.53100 0.019 0.09 0.008
Se 34 RHF 17.0006 2.40980 5.81960 0.272600 3.97310 15.2372 4.35430 43.8163 2.84090 0.016 2.00 0.006
Br 35 RHF 17.1789 2.17230 5.23580 16.5796 5.63770 0.260900 3.98510 41.4328 2.95570 0.012 2.00 0.004
Kr 36 RHF 17.3555 1.93840 6.72860 16.5623 5.54930 0.226100 3.53750 39.3972 2.82500 0.008 2.00 0.002
Rb 37 RHF 17.1784 1.78880 9.64350 17.3151 5.13990 0.274800 1.52920 164.934 3.48730 0.028 0.12 0.008
Sr 38 RHF 17.5663 1.55640 9.81840 14.0988 5.42200 0.166400 2.66940 132.376 2.50640 0.021 0.13 0.005
Y 39 *RHF 17.7760 1.40290 10.2946 12.8006 5.72629 0.125599 3.26588 104.354 1.91213 0.028 0.07 0.006
Zr 40 *RHF 17.8765 1.27618 10.9480 11.9160 5.41732 0.117622 3.65721 87.6627 2.06929 0.035 0.07 0.008
Nb 41 *RHF 17.6142 1.18865 12.0144 11.7660 4.04183 0.204785 3.53346 69.7957 3.75591 0.042 0.08 0.011
Mo 42 RHF 3.70250 0.277200 17.2356 1.09580 12.8876 11.0040 3.74290 61.6584 4.38750 0.046 0.08 0.012
Tc 43 *RHF 19.1301 0.864132 11.0948 8.14487 4.64901 21.5707 2.71263 86.8472 5.40428 0.061 2.00 0.011
Ru 44 *RHF 19.2674 0.808520 12.9182 8.43467 4.86337 24.7997 1.56756 94.2928 5.37874 0.041 2.00 0.006
Rh 45 *RHF 19.2957 0.751536 14.3501 8.21758 4.73425 25.8749 1.28918 98.6062 5.32800 0.021 2.00 0.004
Pd 46 *RHF 19.3319 0.698655 15.5017 7.98929 5.29537 25.2052 0.605844 76.8986 5.26593 0.012 1.08 0.005
Ag 47 RHF 19.2808 0.644600 16.6885 7.47260 4.80450 24.6605 1.04630 99.8156 5.17900 0.016 1.14 0.007
Cd 48 RHF 19.2214 0.594600 17.6444 6.90890 4.46100 24.7008 1.60290 87.4825 5.06940 0.020 2.00 0.008

In 49 RHF 19.1624 0.547600 18.5596 6.37760 4.29480 25.8499 2.03960 92.8029 4.93910 0.027 2.00 0.009
Sn 50 RHF 19.1889 5.83030 19.1005 0.503100 4.45850 26.8909 2.46630 83.9571 4.78210 0.032 2.00 0.009
Sb 51 RHF 19.6418 5.30340 19.0455 0.460700 5.03710 27.9074 2.68270 75.2825 4.59090 0.035 2.00 0.009
Te 52 *RHF 19.9644 4.81742 19.0138 0.420885 6.14487 28.5284 2.52390 70.8403 4.35200 0.038 2.00 0.009
I 53 RHF 20.1472 4.34700 18.9949 0.381400 7.51380 27.7660 2.27350 66.8776 4.07120 0.037 2.00 0.009
Xe 54 RHF 20.2933 3.92820 19.0298 0.344000 8.97670 26.4659 1.99000 64.2658 3.71180 0.038 2.00 0.009
Cs 55 RHF 20.3892 3.56900 19.1062 0.310700 10.6620 24.3879 1.49530 213.904 3.33520 0.032 2.00 0.010
Ba 56 RHF 20.3361 3.21600 19.2970 0.275600 10.8880 20.2073 2.69590 167.202 2.77310 0.032 2.00 0.009
La 57 *RHF 20.5780 2.94817 19.5990 0.244475 11.3727 18.7726 3.28719 133.124 2.14678 0.032 2.00 0.009
Ce 58 *RHF 21.1671 2.81219 19.7695 0.226836 11.8513 17.6083 3.33049 127.113 1.86264 0.026 2.00 0.008
Pr 59 *RHF 22.0440 2.77393 19.6697 0.222087 12.3856 16.7669 2.82428 143.644 2.05830 0.021 0.12 0.007
Nd 60 *RHF 22.6845 2.66248 19.6847 0.210628 12.7740 15.8850 2.85137 137.903 1.98486 0.024 0.13 0.007
Pm 61 *RHF 23.3405 2.56270 19.6095 0.202088 13.1235 15.1009 2.87516 132.721 2.02876 0.026 0.13 0.008
Sm 62 *RHF 24.0042 2.47274 19.4258 0.196451 13.4396 14.3996 2.89604 128.007 2.20963 0.029 0.13 0.009
Eu 63 RHF 24.6274 2.38790 19.0886 0.194200 13.7603 13.7546 2.92270 123.174 2.57450 0.031 0.14 0.010
Gd 64 *RHF 25.0709 2.25341 19.0798 0.181951 13.8518 12.9331 3.54545 101.398 2.41960 0.036 0.15 0.011
Tb 65 *RHF 25.8976 2.24256 18.2185 0.196143 14.3167 12.6648 2.95354 115.362 3.58324 0.035 0.14 0.012
Dy 66 *RHF 26.5070 2.18020 17.6383 0.202172 14.5596 12.1899 2.96577 111.874 4.29728 0.037 0.15 0.013
Ho 67 *RHF 26.9049 2.07051 17.2940 0.197940 14.5583 11.4407 3.63837 92.6566 4.56796 0.040 0.15 0.013
Er 68 *RHF 27.6563 2.07356 16.4285 0.223545 14.9779 11.3604 2.98233 105.703 5.92046 0.040 0.15 0.015
Tm 69 *RHF 28.1819 2.02859 15.8851 0.238849 15.1542 10.9975 2.98706 102.961 6.75621 0.041 0.15 0.016
Yb 70 *RHF 28.6641 1.98890 15.4345 0.257119 15.3087 10.6647 2.98963 100.417 7.56672 0.042 0.15 0.016
Lu 71 *RHF 28.9476 1.90182 15.2208 9.98519 15.1000 0.261033 3.71601 84.3298 7.97628 0.043 0.16 0.016
Hf 72 *RHF 29.1440 1.83262 15.1726 9.59990 14.7586 0.275116 4.30013 72.0290 8.58154 0.047 0.08 0.016
Ta 73 *RHF 29.2024 1.77333 15.2293 9.37046 14.5135 0.295977 4.76492 63.3644 9.24354 0.049 0.08 0.017
W 74 *RHF 29.0818 1.72029 15.4300 9.22590 14.4327 0.321703 5.11982 57.0560 9.88750 0.051 0.09 0.017

Re 75 *RHF 28.7621 1.67191 15.7189 9.09227 14.5564 0.350500 5.44174 52.0861 10.4720 0.052 0.09 0.017
Os 76 *RHF 28.1894 1.62903 16.1550 8.97948 14.9305 0.382661 5.67589 48.1647 11.0005 0.051 0.09 0.017
Ir 77 *RHF 27.3049 1.59279 16.7296 8.86553 15.6115 0.417916 5.83377 45.0011 11.4722 0.050 0.09 0.017
Pt 78 *RHF 27.0059 1.51293 17.7639 8.81174 15.7131 0.424593 5.78370 38.6103 11.6883 0.046 0.10 0.016
Au 79 RHF 16.8819 0.461100 18.5913 8.62160 25.5582 1.48260 5.86000 36.3956 12.0658 0.045 0.10 0.015
Hg 80 RHF 20.6809 0.545000 19.0417 8.44840 21.6575 1.57290 5.96760 38.3246 12.6089 0.046 0.10 0.017
Tl 81 *RHF 27.5446 0.655150 19.1584 8.70751 15.5380 1.96347 5.52593 45.8149 13.1746 0.059 0.09 0.021
Pb 82 RHF 31.0617 0.690200 13.0637 2.35760 18.4420 8.61800 5.96960 47.2579 13.4118 0.060 2.00 0.021
Bi 83 RHF 33.3689 0.704000 12.9510 2.92380 16.5877 8.79370 6.46920 48.0093 13.5782 0.065 2.00 0.020
Po 84 *RHF 34.6726 0.700999 15.4733 3.55078 13.1138 9.55642 7.02588 47.0045 13.6770 0.066 2.00 0.018
At 85 *RHF 35.3163 0.685870 19.0211 3.97458 9.49887 11.3824 7.42518 45.4715 13.7108 0.062 2.00 0.015
Rn 86 RHF 35.5631 0.663100 21.2816 4.06910 8.00370 14.0422 7.44330 44.2473 13.6905 0.054 2.00 0.012
Fr 87 *RHF 35.9299 0.646453 23.0547 4.17619 12.1439 23.1052 2.11253 150.645 13.7247 0.055 2.00 0.017
Ra 88 *RHF 35.7630 0.616341 22.9064 3.87135 12.4739 19.9887 3.21097 142.325 13.6211 0.037 2.00 0.012
Ac 89 *RHF 35.6597 0.589092 23.1032 3.65155 12.5977 18.5990 4.08655 117.020 13.5266 0.030 0.06 0.009
Th 90 *RHF 35.5645 0.563359 23.4219 3.46204 12.7473 17.8309 4.80703 99.1722 13.4314 0.031 0.07 0.008
Pa 91 *RHF 35.8847 0.547751 23.2948 3.41519 14.1891 16.9235 4.17287 105.251 13.4287 0.033 0.06 0.010
U 92 RHF 36.0228 0.529300 23.4128 3.32530 14.9491 16.0927 4.18800 100.613 13.3966 0.035 0.07 0.010
Np 93 *RHF 36.1874 0.511929 23.5964 3.25396 15.6402 15.3622 4.18550 97.4908 13.3573 0.037 0.07 0.011
Pu 94 *RHF 36.5254 0.499384 23.8083 3.26371 16.7707 14.9455 3.47947 105.980 13.3812 0.038 0.14 0.013
Am 95 *RHF 36.6706 0.483629 24.0992 3.20647 17.3415 14.3136 3.49331 102.273 13.3592 0.040 0.07 0.013
Cm 96 *RHF 36.6488 0.465154 24.4096 3.08997 17.3990 13.4346 4.21665 88.4834 13.2887 0.041 0.07 0.013
Bk 97 *RHF 36.7881 0.451018 24.7736 3.04619 17.8919 12.8946 4.23284 86.0030 13.2754 0.042 0.07 0.014
Cf 98 *RHF 36.9185 0.437533 25.1995 3.00775 18.3317 12.4044 4.24391 83.7881 13.2674 0.043 0.07 0.014
H1- 99 HF 0.897661 53.1368 0.565616 15.1870 0.415815 186.576 0.116973 3.56709 0.002389 0.002 0.09 0.001
Li1+ 100 RHF 0.696800 4.62370 0.788800 1.95570 0.341400 0.631600 0.156300 10.0953 0.016700 0.001 1.78 0.000

Be2+ 101 RHF 6.26030 0.002700 0.884900 0.831300 0.799300 2.27580 0.164700 5.11460 -6.1092 0.001 1.97 0.000
Cval 102 HF 2.26069 22.6907 1.56165 0.656665 1.05075 9.75618 0.839259 55.5949 0.286977 0.001 0.16 0.000
O1- 103 HF 4.19160 12.8573 1.63969 4.17236 1.52673 47.0179 -20.307 -0.01404 21.9412 0.011 1.50 0.004
F1- 104 HF 3.63220 5.27756 3.51057 14.7353 1.26064 0.442258 0.940706 47.3437 0.653396 0.003 0.09 0.001
Na1+ 105 RHF 3.25650 2.66710 3.93620 6.11530 1.39980 0.200100 1.00320 14.0390 0.404000 0.001 0.70 0.000
Mg2+ 106 RHF 3.49880 2.16760 3.83780 4.75420 1.32840 0.185000 0.849700 10.1411 0.485300 0.001 1.34 0.000
Al3+ 107 HF 4.17448 1.93816 3.38760 4.14553 1.20296 0.228753 0.528137 8.28524 0.706786 0.000 1.50 0.000
Si4+ 108 HF 4.43918 1.64167 3.20345 3.43757 1.19453 0.214900 0.416530 6.65365 0.746297 0.000 1.50 0.000
Cl1- 109 RHF 18.2915 0.006600 7.20840 1.17170 6.53370 19.5424 2.33860 60.4486 -16.378 0.007 0.76 0.003
K1+ 110 RHF 7.95780 12.6331 7.49170 0.767400 6.35900 -0.00200 1.19150 31.9128 -4.9978 0.011 0.91 0.005
Ca2+ 111 RHF 15.6348 -0.00740 7.95180 0.608900 8.43720 10.3116 0.853700 25.9905 -14.875 0.017 2.00 0.004
Sc3+ 112 HF 13.4008 0.298540 8.02730 7.96290 1.65943 -0.28604 1.57936 16.0662 -6.6667 0.002 1.50 0.000
Ti2+ 113 HF 9.11423 7.52430 7.62174 0.457585 2.27930 19.5361 0.087899 61.6558 0.897155 0.006 1.50 0.001
Ti3+ 114 HF 17.7344 0.220610 8.73816 7.04716 5.25691 -0.15762 1.92134 15.9768 -14.652 0.001 0.00 0.000
Ti4+ 115 HF 19.5114 0.178847 8.23473 6.67018 2.01341 -0.29263 1.52080 12.9464 -13.280 0.002 1.50 0.000
V2+ 116 RHF 10.1060 6.88180 7.35410 0.440900 2.28840 20.3004 0.022300 115.122 1.22980 0.015 2.00 0.004
V3+ 117 HF 9.43141 6.39535 7.74190 0.383349 2.15343 15.1908 0.016865 63.9690 0.656565 0.004 1.50 0.001
V5+ 118 HF 15.6887 0.679003 8.14208 5.40135 2.03081 9.97278 -9.5760 0.940464 1.71430 0.000 0.34 0.000
Cr2+ 119 HF 9.54034 5.66078 7.75090 0.344261 3.58274 13.3075 0.509107 32.4224 0.616898 0.002 1.50 0.000
Cr3+ 120 HF 9.68090 5.59463 7.81136 0.334393 2.87603 12.8288 0.113575 32.8761 0.518275 0.002 1.50 0.000
Mn2+ 121 RHF 10.8061 5.27960 7.36200 0.343500 3.52680 14.3430 0.218400 41.3235 1.08740 0.009 2.00 0.002
Mn3+ 122 HF 9.84521 4.91797 7.87194 0.294393 3.56531 10.8171 0.323613 24.1281 0.393974 0.001 1.50 0.000
Mn4+ 123 HF 9.96253 4.84850 7.97057 0.283303 2.76067 10.4852 0.054447 27.5730 0.251877 0.001 1.50 0.000
Fe2+ 124 RHF 11.0424 4.65380 7.37400 0.305300 4.13460 12.0546 0.439900 31.2809 1.00970 0.008 2.00 0.002
Fe3+ 125 RHF 11.1764 4.61470 7.38630 0.300500 3.39480 11.6729 0.072400 38.5566 0.970700 0.008 2.00 0.002
Co2+ 126 RHF 11.2296 4.12310 7.38830 0.272600 4.73930 10.2443 0.710800 25.6466 0.932400 0.006 2.00 0.001

Co3+ 127 HF 10.3380 3.90969 7.88173 0.238668 4.76795 8.35583 0.725591 18.3491 0.286667 0.000 1.50 0.000
Ni2+ 128 RHF 11.4166 3.67660 7.40050 0.244900 5.34420 8.87300 0.977300 22.1626 0.861400 0.003 2.00 0.001
Ni3+ 129 HF 10.7806 3.54770 7.75868 0.223140 5.22746 7.64468 0.847114 16.9673 0.386044 0.000 0.57 0.000
Cu1+ 130 RHF 11.9475 3.36690 7.35730 0.227400 6.24550 8.66250 1.55780 25.8487 0.89000 0.003 0.24 0.001
Cu2+ 131 HF 11.8168 3.37484 7.11181 0.244078 5.78135 7.98760 1.14523 19.8970 1.14431 0.001 0.26 0.000
Zn2+ 132 RHF 11.9719 2.99460 7.38620 0.203100 6.46680 7.08260 1.39400 18.0995 0.780700 0.001 0.62 0.000
Ga3+ 133 HF 12.6920 2.81262 6.69883 0.227890 6.06692 6.36441 1.00660 14.4122 1.53545 0.008 1.45 0.000
Ge4+ 134 HF 12.9172 2.53718 6.70003 0.205855 6.06791 5.47913 0.859041 11.6030 1.45572 0.000 0.32 0.000
Br1- 135 RHF 17.1718 2.20590 6.33380 19.3345 5.57540 0.287100 3.72720 58.1535 3.17760 0.016 2.00 0.006
Rb1+ 136 RHF 17.5816 1.71390 7.65980 14.7957 5.89810 0.160300 2.78170 31.2087 2.07820 0.002 1.99 0.001
Sr2+ 137 RHF 18.0874 1.49070 8.13730 12.6963 2.56540 24.5651 -34.193 -0.01380 41.4025 0.008 2.00 0.002
Y3+ 138 *DS 17.9268 1.35417 9.15310 11.2145 1.76795 22.6599 -33.108 -0.01319 40.2602 0.005 2.00 0.001
Zr4+ 139 *DS 18.1668 1.21480 10.0562 10.1483 1.01118 21.6054 -2.6479 -0.10276 9.41454 0.004 2.00 0.001
Nb3+ 140 *DS 19.8812 0.019175 18.0653 1.13305 11.0177 10.1621 1.94715 28.3389 -12.912 0.006 2.00 0.002
Nb5+ 141 *DS 17.9163 1.12446 13.3417 0.028781 10.7990 9.28206 0.337905 25.7228 -6.3934 0.007 2.00 0.003
Mo3+ 142 *DS 21.1664 0.014734 18.2017 1.03031 11.7423 9.53659 2.30951 26.6307 -14.421 0.009 2.00 0.003
Mo5+ 143 *DS 21.0149 0.014345 18.0992 1.02238 11.4632 8.78809 0.740625 23.3452 -14.316 0.010 2.00 0.003
Mo6+ 144 *DS 17.8871 1.03649 11.1750 8.48061 6.57891 0.058881 0.000000 0.000000 0.344941 0.014 0.00 0.006
Ru3+ 145 *DS 18.5638 0.847329 13.2885 8.37164 9.32602 0.017662 3.00964 22.8870 -3.1892 0.013 2.00 0.004
Ru4+ 146 *DS 18.5003 0.844582 13.1787 8.12534 4.71304 0.36495 2.18535 20.8504 1.42357 0.014 2.00 0.004
Rh3+ 147 *DS 18.8785 0.764252 14.1259 7.84438 3.32515 21.2487 -6.1989 -0.01036 11.8678 0.014 2.00 0.004
Rh4+ 148 *DS 18.8545 0.760825 13.9806 7.62436 2.53464 19.3317 -5.6526 -0.01020 11.2835 0.014 2.00 0.003
Pd2+ 149 *DS 19.1701 0.696219 15.2096 7.55573 4.32234 22.5057 0.000000 0.000000 5.29160 0.011 2.00 0.004
Pd4+ 150 *DS 19.2493 0.683839 14.7900 7.14833 2.89289 17.9144 -7.9492 0.005127 13.0174 0.014 2.00 0.003
Ag1+ 151 *DS 19.1812 0.646179 15.9719 7.19123 5.27475 21.7326 0.357534 66.1147 5.21572 0.012 1.13 0.005
Ag2+ 152 *DS 19.1643 0.645643 16.2456 7.18544 4.37090 21.4072 0.000000 0.000000 5.21404 0.011 1.14 0.005

Cd2+ 153 *DS 19.1514 0.597922 17.2535 6.80639 4.47128 20.2521 0.000000 0.000000 5.11937 0.014 1.17 0.007
In3+ 154 *DS 19.1045 0.551522 18.1108 6.32470 3.78897 17.3595 0.000000 0.000000 4.99635 0.022 2.00 0.007
Sn2+ 155 RHF 19.1094 0.503600 19.0548 5.83780 4.56480 23.3752 0.487000 62.2061 4.78610 0.032 2.00 0.009
Sn4+ 156 RHF 18.9333 5.76400 19.7131 0.465500 3.41820 14.0049 0.019300 -0.75830 3.91820 0.016 2.00 0.004
Sb3+ 157 *DS 18.9755 0.467196 18.9330 5.22126 5.10789 19.5902 0.288753 55.5113 4.69626 0.028 2.00 0.007
Sb5+ 158 *DS 19.8685 5.44853 19.0302 0.467973 2.41253 14.1259 0.000000 0.000000 4.69263 0.030 2.00 0.008
I1- 159 RHF 20.2332 4.35790 18.9970 0.381500 7.80690 29.5259 2.88680 84.9304 4.07140 0.038 2.00 0.009
Cs1+ 160 RHF 20.3524 3.55200 19.1278 0.308600 10.2821 23.7128 0.961500 59.4565 3.27910 0.037 2.00 0.009
Ba2+ 161 *DS 20.1807 3.21367 19.1136 0.283310 10.9054 20.0558 0.77634 51.7460 3.02902 0.029 2.00 0.007
La3+ 162 *DS 20.2489 2.92070 19.3763 0.250698 11.6323 17.8211 0.336048 54.9453 2.40860 0.028 2.00 0.007
Ce3+ 163 *DS 20.8036 2.77691 19.5590 0.231540 11.9369 16.5408 0.612376 43.1692 2.09013 0.023 2.00 0.005
Ce4+ 164 *DS 20.3235 2.65941 19.8186 0.218850 12.1233 15.7992 0.144583 62.2355 1.59180 0.026 2.00 0.007
Pr3+ 165 *DS 21.3727 2.64520 19.7491 0.214299 12.1329 15.3230 0.975180 36.4065 1.77132 0.019 2.00 0.004
Pr4+ 166 *DS 20.9413 2.54467 20.0539 0.202481 12.4668 14.8137 0.296689 45.4643 1.24285 0.021 2.00 0.005
Nd3+ 167 *DS 21.9610 2.52722 19.9339 0.199237 12.1200 14.1783 1.51031 30.8717 1.47588 0.015 2.00 0.003
Pm3+ 168 *DS 22.5527 2.41740 20.1108 0.185769 12.0671 13.1275 2.07492 27.4491 1.19499 0.012 2.00 0.002
Sm3+ 169 *DS 23.1504 2.31641 20.2599 0.174081 11.9202 12.1571 2.71488 24.8242 0.954586 0.009 2.00 0.002
Eu2+ 170 *DS 24.0063 2.27783 19.9504 0.173530 11.8034 11.6096 3.87243 26.5156 1.36389 0.004 2.00 0.002
Eu3+ 171 *DS 23.7497 2.22258 20.3745 0.163940 11.8509 11.3110 3.26503 22.9966 0.759344 0.006 2.00 0.001
Gd3+ 172 *DS 24.3466 2.13553 20.4208 0.155525 11.8708 10.5782 3.71490 21.7029 0.645089 0.004 2.00 0.001
Tb3+ 173 *DS 24.9559 2.05601 20.3271 0.149525 12.2471 10.0499 3.77300 21.2773 0.691967 0.005 0.00 0.001
Dy3+ 174 *DS 25.5395 1.98040 20.2861 0.143384 11.9812 9.34972 4.50073 19.5810 0.689690 0.003 0.00 0.001
Ho3+ 175 *DS 26.1296 1.91072 20.0994 0.139358 11.9788 8.80018 4.93676 18.5908 0.852795 0.003 0.00 0.001
Er3+ 176 *DS 26.7220 1.84659 19.7748 0.137290 12.1506 8.36225 5.17379 17.8974 1.17613 0.003 0.00 0.001
Tm3+ 177 *DS 27.3083 1.78711 19.3320 0.136974 12.3339 7.96778 5.38348 17.2922 1.63929 0.003 0.00 0.001
Yb2+ 178 *DS 28.1209 1.78503 17.6817 0.159970 13.3335 8.18304 5.14657 20.3900 3.70983 0.008 0.00 0.003

Yb3+ 179 *DS 27.8917 1.73272 18.7614 0.138790 12.6072 7.64412 5.47647 16.8153 2.26001 0.003 0.00 0.002
Lu3+ 180 *DS 28.4628 1.68216 18.1210 0.142292 12.8429 7.33727 5.59415 16.3535 2.97573 0.004 0.14 0.002
Hf4+ 181 *DS 28.8131 1.59136 18.4601 0.128903 12.7285 6.76232 5.59927 14.0366 2.39699 0.002 0.00 0.001
Ta5+ 182 *DS 29.1587 1.50711 18.8407 0.116741 12.8268 6.31524 5.38695 12.4244 1.78555 0.002 2.00 0.001
W6+ 183 *DS 29.4936 1.42755 19.3763 0.104621 13.0544 5.93667 5.06412 11.1972 1.01074 0.001 0.00 0.000
Os4+ 184 *DS 30.4190 1.37113 15.2637 6.84706 14.7458 0.165191 5.06795 18.0030 6.49804 0.006 0.29 0.003
Ir3+ 185 *DS 30.4156 1.34323 15.8620 7.10909 13.6145 0.204633 5.82008 20.3254 8.27903 0.009 0.28 0.004
Ir4+ 186 *DS 30.7058 1.30923 15.5512 6.71983 14.2326 0.167252 5.53672 17.4911 6.96824 0.006 0.29 0.003
Pt2+ 187 *DS 29.8429 1.32927 16.7224 7.38979 13.2153 0.263297 6.35234 22.9426 9.85329 0.014 0.00 0.006
Pt4+ 188 *DS 30.9612 1.24813 15.9829 6.60834 13.7348 0.168640 5.92034 16.9392 7.39534 0.006 0.14 0.003
Au1+ 189 *DS 28.0109 1.35321 17.8204 7.73950 14.3359 0.356752 6.58077 26.4043 11.2299 0.023 0.12 0.009
Au3+ 190 *DS 30.6886 1.21990 16.9029 6.82872 12.7801 0.212867 6.52354 18.6590 9.09680 0.009 0.14 0.004
Hg1+ 191 *DS 25.0853 1.39507 18.4973 7.65105 16.8883 0.443378 6.48216 28.2262 12.0205 0.027 0.12 0.011
Hg2+ 192 *DS 29.5641 1.21152 18.0600 7.05639 12.8374 0.284738 6.89912 20.7482 10.6268 0.013 0.00 0.006
Tl1+ 193 *DS 21.3985 1.47110 20.4723 0.517394 18.7478 7.43463 6.82847 28.8482 12.5258 0.028 0.12 0.011
Tl3+ 194 *DS 30.8695 1.10080 18.3481 6.53852 11.9328 0.219074 7.00574 17.2114 9.80270 0.008 0.01 0.004
Pb2+ 195 *DS 21.7886 1.33660 19.5682 0.488383 19.1406 6.77270 7.01107 23.8132 12.4734 0.020 2.00 0.008
Pb4+ 196 *DS 32.1244 1.00566 18.8003 6.10926 12.0175 0.147041 6.96886 14.7140 8.08428 0.005 0.31 0.002
Bi3+ 197 *DS 21.8053 1.23560 19.5026 6.24149 19.1053 0.469999 7.10295 20.3185 12.4711 0.015 2.00 0.006
Bi5+ 198 *DS 33.5364 0.916540 25.0946 0.39042 19.2497 5.71414 6.91555 12.8285 -6.7994 0.003 0.00 0.001
Ra2+ 199 *DS 35.2150 0.604909 21.6700 3.57670 7.91342 12.6010 7.65078 29.8436 13.5431 0.029 2.00 0.006
Ac3+ 200 *DS 35.1736 0.579689 22.1112 3.41437 8.19216 12.9187 7.05545 25.9443 13.4637 0.021 2.00 0.004
Th4+ 201 *DS 35.1007 0.555054 22.4418 3.24498 9.78554 13.4661 5.29444 23.9533 13.3760 0.014 2.00 0.002
U3+ 202 *DS 35.5747 0.520480 22.5259 3.12293 12.2165 12.7148 5.37073 26.3394 13.3092 0.009 2.00 0.002
U4+ 203 *DS 35.3715 0.516598 22.5326 3.05053 12.0291 12.5723 4.79840 23.4582 13.2671 0.007 2.00 0.001
U6+ 204 *DS 34.8509 0.507079 22.7584 2.89030 14.0099 13.1767 1.21457 25.2017 13.1665 0.003 2.00 0.001

Np3+ 205 *DS 35.7074 0.502322 22.6130 3.03807 12.9898 12.1449 5.43227 25.4928 13.2544 0.006 2.00 0.002
Np4+ 206 *DS 35.5103 0.498626 22.5787 2.96627 12.7766 11.9484 4.92159 22.7502 13.2116 0.005 2.00 0.001
Np6+ 207 *DS 35.0136 0.489810 22.7286 2.81099 14.3884 12.3300 1.75669 22.6581 13.1130 0.002 2.00 0.001
Pu3+ 208 *DS 35.8400 0.484938 22.7169 2.96118 13.5807 11.5331 5.66016 24.3992 13.1991 0.005 2.00 0.001
Pu4+ 209 *DS 35.6493 0.481422 22.6460 2.89020 13.3595 11.3160 5.18831 21.8301 13.1555 0.003 2.00 0.001
Pu6+ 210 *DS 35.1736 0.473204 22.7181 2.73848 14.7635 11.5530 2.28678 20.9303 13.0582 0.001 1.36 0.001
D 211 assumedWithH 0.489918 20.6593 0.262003 7.74039 0.196767 49.5519 0.049879 2.20159 0.001305 0.000 0.17 0.000