CS182 Project Report: CGCNN

Yunyeong Choi[1] **Xin Chen**[1] **Jinho Shin**[1] **Aryan Chakrabarti**[1]
[1]UC Berkeley {yychoi94,chenxin0210,jhs0640,aryanc}@berkeley.edu
SID: 3035213351, 3032981876, 3034929951, 3034676984

Part 0: Abstract

Predicting material properties exclusively from atomic information has been a long-standing aspiration for physicists. The development of Density Functional Theory [1] and subsequent algorithms have enabled the calculation of relatively accurate properties. However, due to the constraints of the self-consistent calculation method, the time consumption increases faster than $O(N^3)$, where N is the number of atoms. Since 2007, after J. Behler published the first Machine Learning approach [2], researchers have focused on building large databases of computed materials properties, and many different models have been tried to achieve higher accuracy with adequate computation time. In this project, we will focus on one of the most impactful papers [3], which first adopted the graph-convolution idea to predict materials properties.

The key idea of this project is that inorganic materials have many properties that can be effectively modeled using Convolutional Neural Networks (CNNs). For instance:

1. **Invariance**: As we have seen in previous work, CNNs have an inductive bias that captures the translational and rotational invariance of images. In inorganic materials, symmetry operations are always used to define the material [4], and they affect the material's properties as well. In this context, one can expect that a graph-convolution network can capture symmetry with an inductive bias and perform better than other NNs.
2. **Locality**: Among the various interactions between atoms, physicists usually regard nearest-neighbor interaction as the most important and largest interaction. Except for some unique cases, this is considered true due to the form of the electrostatic potential, which is linear to $O(1/r^2)$. CNNs are an excellent choice for capturing the locality of data, and by using several CNN layers, we can also capture long-range interactions of atoms with enlarged receptive fields.
3. **Weight Sharing**: In machine learning approaches such as fully connected networks, at least 10,000 weights are typically used to predict energy. This is not ideal, since we know that the material's property is only determined by a quantum mechanical equation. To mimic this, we need to limit the number of weights and share them between different atoms. This is exactly what CNNs do.

Therefore by implementing this paper as a homework set, we not only expect to enforce students' understanding of graph-convolution networks but also expect to broaden their perspective by looking at how the ML approaches are used in other domains.

The goal of this project is to:

1. Implement the CGCNN (Crystal Graph Convolution Neural Network) with a better explanation and make a homework notebook set to understand and execute it easily.
2. Train it with a limited amount of data (focusing on only one type of material, such as perovskite.) We expect to train on $\sim$1,000 different materials, which can be done in a few mins on a local computer or colab.
3. Change the convolution layer and see what kind of graph convolution works better. Also, test the effect of batch normalization and activation functions

Part 1 & 2: Review comments & Response to comments

Reviewer 1

1. Question 1: Content and Correctness (Option 1). If this paper is not Option 1, write NA.
The coding question does engage with the selected key concepts, including the encoding of the data and the implementation of the actual network. The code in the solution is correct with no bugs. However, the import statement in the building a model section is incorrect (from model import CrystalGraphConvNet), it is importing from the solution file(model.py) instead of the blank homework file(model_hw). The assertion statement used for testing(assert out.item() == 0.6627039313316345) also raises an error, even though the solution code itself seems correct, probably due to rounding differences. It would be nice to change the testing method so that rounding differences would not be docked points. Overall the code runs smoothly and could be completed in a reasonable timeframe.

2. Question 1: Literature Background/Problem Formulation (Option 2). If this paper is not Option 2, write NA.
NA

3. Question 1 - grade
Small improvement needed

Answer to Question 1: Now we import CrystalGraphConvNet from model_hw.py, which asks students to fill in the blank. The assertion statement is fixed, we now considering rounding differences.

4. Question 2: Scaffolding (Option 1). If this paper is not Option 1, write NA.
The homework is mostly self-contained, though it would be helpful to include a little more information(hint) regarding the encoding part of the homework, especially the structure of the element_embedding_file and how to access the embedded specie numbers. All other parts seem well explained, and also enough hints are given. All datasets and required packages are provided, and the code for downloading/importing them runs smoothly.

5. Question 2: Technical Approach/Deep Learning Techniques (Option 2). If this paper is not Option 2, write NA.
NA

6. Question 2 - grade
Small improvement needed

Answer to Question 2: We added additional guidance hinting use of pymatgen.core.structure.specie.number, which addresses accessing the embedded specie numbers. Change is made in the encoding section.

7. Question 3: Readability/Clarity (Option 1). If this paper is not Option 1, write NA.
There are some minor typos, but they do not interfere with understanding the reading. The directions, hints, and background are easy to follow. The commentary explains the concepts well and also includes information regarding special symbols used. Overall a pretty clear assignment.

8. Question 3: Experiments and Contributions (Option 2). If this paper is not Option 2, write NA.
NA

9. Question 3 - grade
Small improvement needed

Answer to Question 3: The minor typos have been fixed and reviewed.

10. Question 4: Commentary on HW (Option 1). If this paper is not Option 1, write NA.
The commentary is clear and nicely structured. The main concepts and goals of the homework are clearly listed and explained. In the commentary, it is mentioned that there are several types of convolution layers, and some work better than others. It would be nice if the homework could allow the students to compare them side by side(or implement all if time allows) and explore the reasons for one's superiority over another. In the current version of the hw and solution, the comparison between them is not clear. Might also be nice to include a simple written sub-question exploring the invariance in the network.

11. Question 4: Code and Datasets (Option 2). If this paper is not Option 2, write NA.
NA

12. Question 4 - grade
Small improvement needed

Answer to Question 4: In the model_hw.py, We add an option that allows students can change the convolution layer, and compare their performance of them. Also, to make it clear we add a hint that why one performs than the other in the training section of the notebook.

13. Question 5: Going above and beyond (Option 1). If this paper is not Option 1, write NA.
The homework does a good job of extracting important concepts/information from the paper and especially targeting them in the homework. The dataset was also simplified so then students could run the network on their local computer without waiting too long. The commentary also mentions recent studies focused on improving the encoding of the data. The homework could explore the improvements if time allows.

14. Question 5: Readability/Clarity (Option 2). If this paper is not Option 2, write NA.
NA

15. Question 5 - grade
Small improvement needed

Answer to Question 5: This comment is a compilation of all the previous ones.

Reviewer 2

1. Question 1: Content and Correctness (Option 1). If this paper is not Option 1, write NA.
Score: 11/15 Overall, the submission is well done and mathematically correct. The student can follow along the flow of the assignment and tasks asked by the assignment are reasonable. It is however lacking polish which is evident in the fact that the written portion solutions were not included. Additionally, the provided code would not run and fail its own assertion checks at times. But these were relatively easily repaired by the end user.
I have included the problems below. (all are relatively easily fixed) (skip the bullet points below if you are not interested in reading these details):
1) Failed assertion check on assert torch.equal(nbr_fea_idx, nbr_fea_idx_standard), nbr_fea_idx - It fails because the order of the vertices pymatgen returns is probably not fixed - Solution is to probably sort the arrays before comparison
2) Failed assertion check for implementing the graph convolution assert out.item() == 0.6627039313316345
- Failed due to floating point imprecision
- Simple fix is to add an epsilon

3) Not an error but please import scikitlearn at the top of the notebook
- it is annoying to have to exit python notebook and install the module midway through the notebook. This just caused me to lose all my environment variables which means I had to rerun all cells. Probably not an issue if this notebook is run on Colab

One final comment ion the code s that the hardest part to follow was probably the very first task which asked the student to embed the neighbor matrices using the provided atom embeddings. The answer used nacl[i].specie.number however the homework gave absolutely no indication that these class members even existed or that the nacl object has __getitem__ method implemented. This is fixed relatively easily by a helpful comment.

2. Question 1: Literature Background/Problem Formulation (Option 2). If this paper is not Option 2, write NA.
NA

3. Question 1 - grade
Small improvement needed

Answer to Question 1: We fixed the import statement errors as well as the assertions. Also, now we import scikit learn at the top of the notebook. About the embedding part, we agree with the reviewer's point and now we gave enough hints about how to get the atomic numbers from the structure object from pymatgen package.

4. Question 2: Scaffolding (Option 1). If this paper is not Option 1, write NA.
Score: 14/15 The homework provides sufficient scaffolding for the student. The only issue was not providing more info on how to interact with the pymatgen library.

5. Question 2: Technical Approach/Deep Learning Techniques (Option 2). If this paper is not Option 2, write NA.
NA

6. Question 2 - grade
Small improvement needed

Answer to Question 2: With the additional hint about specie.number attribute, we now provide more guidance about the pymatgen library before we use that.

7. Question 3: Readability/Clarity (Option 1). If this paper is not Option 1, write NA.
Very readable and clear. It would be beneficial to include some more background info though in the problem statement. Like explaining what "properties" we are trying to predict of the material.

8. Question 3: Experiments and Contributions (Option 2). If this paper is not Option 2, write NA.
NA

9. Question 3 - grade
Small improvement needed

Answer to Question 3: We added an additional introductory explanation about the crystal convolution context and the properties (energy) we are trying to predict. Additional explanations are under the training section of the notebook.

10. Question 4: Commentary on HW (Option 1). If this paper is not Option 1, write NA.
30/30
The commentary is good and informative. No issues here. Included everything I wanted to know. The homework assignment reaches all the learning objectives set out in the commentary.

11. Question 4: Code and Datasets (Option 2). If this paper is not Option 2, write NA.
NA

12. Question 4 - grade
Excellent work, no actions needed.

Answer to Question 4: No actions needed.

13. Question 5: Going above and beyond (Option 1). If this paper is not Option 1, write NA.
10
Experimented with using batch norm and different activation functions and compared results.

14. Question 5: Readability/Clarity (Option 2). If this paper is not Option 2, write NA.
NA

15. Question 5 - grade
Medium improvement needed

Answer to Question 5: We give space for students to experiment with batch normalization and other activation functions, but decided not to enforce it to make the workload manageable.

Reviewer 3

1. Question 1: Content and Correctness (Option 1). If this paper is not Option 1, write NA.
Content and correctness look good to me. Adequate mathematical and coding context provided for questions.

2. Question 1: Literature Background/Problem Formulation (Option 2). If this paper is not Option 2, write NA.
NA

3. Question 1 - grade
Excellent work, no actions needed.

Answer to Question 1: No actions needed.

4. Question 2: Scaffolding (Option 1). If this paper is not Option 1, write NA.
The "!pip install" statements and visualizations are not present in the solutions notebook. It's a bit unclear what it means to expand neighbor features in the "expand" function. More context is needed regarding what columns in the nacl structure table represent. Give more context regarding how crystal data need to be encoded to atomistic features.

5. Question 2: Technical Approach/Deep Learning Techniques (Option 2). If this paper is not Option 2, write NA.
NA

6. Question 2 - grade
Small improvement needed

Answer to Question 2: We added a description of what the expand function is supposed to do, and added information about species numbers to help with understanding crystal encoding.

7. Question 3: Readability/Clarity (Option 1). If this paper is not Option 1, write

NA.
Comments can be more detailed with steps that students can follow to implement the functions.

8. Question 3: Experiments and Contributions (Option 2). If this paper is not Option 2, write NA.
NA

9. Question 3 - grade
Small improvement needed

Answer to Question 3: We added more hints to use functions, such as methods in pymatgen library. Now it become more straightforward to follow.

10. Question 4: Commentary on HW (Option 1). If this paper is not Option 1, write NA.
Additional information can be provided regarding what the model that students implement does, and some additional outputs can be shown for clarity rather than just loss curves.

11. Question 4: Code and Datasets (Option 2). If this paper is not Option 2, write NA.
NA

12. Question 4 - grade
Small improvement needed

Answer to Question 4: We provided more detailed information on the task of this model, which is to predict material properties(energy) based on its structure. These are written under the training section in the notebook we provided. Regarding visualizing, a current model only predicts energy so we decided to remain our visualization part. However, students can find their predictions and corresponding structures at test_result.csv file, and we mentioned that too now. All changes are in the training section of the notebook.

13. Question 5: Going above and beyond (Option 1). If this paper is not Option 1, write NA.
More systematic information needs to be provided regarding material science properties that can be exploited and predicted using CGCNNs. Background information missing for paper to be fully understood by students.

14. Question 5: Readability/Clarity (Option 2). If this paper is not Option 2, write NA.
NA

15. Question 5 - grade
Medium improvement needed

Answer to Question 5: We added an explanation about what property we are going to predict. Also added information about two different convolution layers how they are different and why one performs better.

Reviewer 4

1. Question 1: Content and Correctness (Option 1). If this paper is not Option 1, write NA.
NA

2. Question 1: Literature Background/Problem Formulation (Option 2). If this paper is not Option 2, write NA.
The key idea of this problem is that inorganic materials have many properties in common with

Convolutional Neural Networks (CNNs), such as invariance, locality and weight sharing, which indicates that inorganic materials can be represented using CNNs. In particular, inorganic materials can be represented using convolutional graph neural network.

3. Question 1 - grade
Excellent work, no actions needed.

Answer to Question 1: No actions needed.

4. Question 2: Scaffolding (Option 1). If this paper is not Option 1, write NA.
NA

5. Question 2: Technical Approach/Deep Learning Techniques (Option 2). If this paper is not Option 2, write NA.
They let each atomic feature vector, which encodes the property of the atom corresponding to node. The edge feature vector is represented as u(i,j)k, which encodes k-th bond connecting atom i, j. After the crystal graph is built, consecutive fully connected networks and pooling layer are used to predict the property of crystals. In the case of regression, the error function is set to MSE, and in the case of classification, NLL (Negative Log Likelihood) loss is used.

6. Question 2 - grade
Excellent work, no actions needed.

Answer to Question 2: No actions needed.

7. Question 3: Readability/Clarity (Option 1). If this paper is not Option 1, write NA.
NA

8. Question 3: Experiments and Contributions (Option 2). If this paper is not Option 2, write NA.
They did experiments on two datasets. First, they used to train only with 700 perovskite structures, and predicted the energy of materials. Second, they used a mixed data set of perovskite and random oxide materials, which had classifications of 1 and 0.

9. Question 3 - grade
Excellent work, no actions needed.

Answer to Question 3: No actions needed.

10. Question 4: Commentary on HW (Option 1). If this paper is not Option 1, write NA.
NA

11. Question 4: Code and Datasets (Option 2). If this paper is not Option 2, write NA.
The code for todo is clear and without no controversy. The dataset is also nice.

12. Question 4 - grade
Excellent work, no actions needed.

Answer to Question 4: No actions needed.

13. Question 5: Going above and beyond (Option 1). If this paper is not Option 1, write NA.
NA

14. Question 5: Readability/Clarity (Option 2). If this paper is not Option 2, write NA.

The file of the project is very clear and understandable.

15. Question 5 - grade
Excellent work, no actions needed.
Answer to Question 5: No actions needed.

Part 3: Final Submission

Encoding Crystal into crystal graph

The key idea of CGCNN is to represent crystal structure by a crystal graph, which encodes not only the atomic information but also bonding information between atoms based on the distance. Here we will let each atomic feature vector as $\mathbf{v}_i$, which encodes the property of the atom corresponding to node i. The edge feature vector is represented as $\mathbf{u}_{(i,j)k}$, which encodes k-th bond connecting atom i, j. Note that there can be several bonds between atoms, which originate from the periodic nature of crystals.
Obviously, there are many ways to encode atomic and bond data. For the sake of simplicity, we will follow the methods used in the CGCNN paper, which used a pre-defined encoding vector to change atoms into vectors. These vectors have 92 dimensions, and for 100 atoms in the periodic table, they have different encodings which consist of 0 and 1. For the edge feature vectors, there also can be many ways to encode bond information; bond length, angle, and covalency. However, we will only use bond-length information between nearest neighbors, by applying a Gaussian kernel to change it to encoding vectors. One can follow the homework notebook on how this embedding is done to NaCl(salt) Crystal.
Note that this encoding could be very simple to deliver all information of interactions in crystal. Recent studies also focused on improving this; Some consider angles to capture three-body interaction, or even Mol2vec, which is a molecule version of Word2vec, is developed to change encode effectively.

Convolutional graph neural network

After building the crystal graph, the convolutional neural network act on top of the graph, and consecutive fully connected networks and pooling layer are used to predict the property of crystals. The convolution layer iteratively updates the atom feature vector $\mathbf{v}_i$ by message passing with surrounding atoms and bonds with a non-linear graph convolution function. It can be written as Eq 1.

$$v_i^{(t+1)} = Conv(v_i^{(t)}, v_j^{(t)}, u_{(i,j)_k}), (i,j)_k \in G \tag{1}$$

After iterations of convolution layers, the pooling layer of Eq 2 is used to produce overall feature vector $\mathbf{v}_c$ for the crystal, and consecutive two fully connected hidden layers are used to generate outputs. The overall architecture is not only confined to the regression problem but also can be used for classification. In the case of regression, the error function is set to MSE, and in the case of classification, NLL (Negative Log Likelihood) loss is used.

$$v_c = Pool(v_0^{(0)}, v_1^{(0)}, ..., v_N^{(0)}, ..., v_N^{(R)}) \tag{2}$$

Note that this 'convolutional' graph is not exactly meaning the CNN; it is doing message passing in the crystal graph we built. Therefore, if we maintain the permutation invariance of the convolution layer we can choose any function we want. In the original paper, the following Eq 3 and Eq 4 are tested.

$$v_i^{(t+1)} = g[(\sum_{j,k} v_j^{(t)} \oplus u_{(i,j)_k})W_c^{(t)} + v_i^{(t)}W_s^{(t)} + b^{(t)}] \tag{3}$$

$$v_i^{(t+1)} = v_i^{(t+1)} + \sum_{j,k} \sigma(z_{(i,j)_k}^{(t)} W_f^{(t)} + b_f^{(t)}) \odot g(z_{(i,j)_k}^{(t)} W_s^{(t)} + b_s^{(t)}) \tag{4}$$

where $\oplus$ denotes the concatenation of vectors, $\odot$ denotes element-wise multiplication, and $W_c^{(t)}, W_s^{(t)}, b^{(t)}$ are the convolution weight matrix, self-weight matrix, and bias. g is the activation function and σ is sigmoid function. Note that Eq 3 weight matrix $W_c^{(t)}$ is shared between all different neighbors, which is not ideal to describe materials property and results in a large validation error. The author used Eq 4 instead of that, which differentiates interactions between neighbors and added residual connection to prevent gradient vanishing.

In this project, we will change the convolution layers to compare and analyze which type of convolution layer is the best to describe the property of crystals. Students will test the following two different convolution layers and compare their training/validation errors of them;

1. Shared convolution weight matrix between different sets of nodes. Case of Eq 3
2. Different convolution weight matrices between different sets of nodes. Case of Eq 4

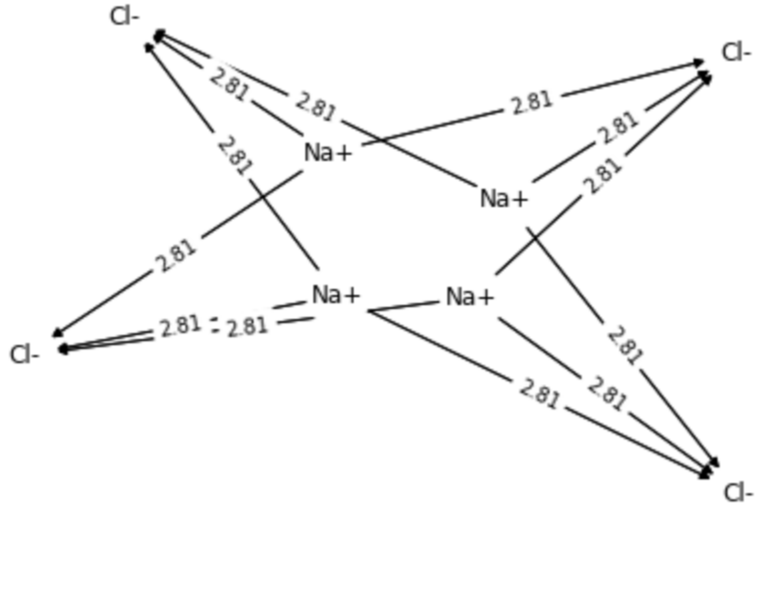
If time allows, we also recommend one to test no convolution layer, which means atom information is fixed as same as the initial encoding, and just fully connected network and activation function is used to predict materials properties.

Dataset

Ideally, this architecture can be used to predict any property of materials. However, for simplicity, we will limit this to predicting energy and simple classification of materials family. For the homework dataset, 700 structures of the Perovskite family in the Materials Project Database are chosen. One will use to train only with 700 perovskite structures, and going to predict the energy of materials. Note that we only use the perovskite dataset since it maintains similar symmetry to each other, and is a relatively simple material family to use as a train set. If time allows, we also recommend trying classification. One can use a mixed data set of perovskite and random oxide materials, which has classifications of 1 and 0. This dataset can be used to train the classification network and going to check whether this network can classify perovskite from other oxide materials.

Homework Problems

Deep neural networks are being used everywhere. In this problem we will explore how materials science researchers have used them to predict material properties. You can view the paper for reference:
https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.120.145301.



1. The graph above represents the crystal structure of NaCl, otherwise known as table salt. Complete the "Encoding Crystal into Crystal Graph" section in **cs182_project_hw.ipynb** to see how we can embed this crystal data into vectors that we can process in a CGCNN.

2. Complete the **model_hw.py** file, which will contain the elements of our CGCNN architecture. Then run the rest of the Python notebook to train and visualize model performance. How does training and validation loss differ with different convolution and different hyperparameters?
3. In the encoding section of the notebook, we use information from nearby atoms and encode this into our data vector. Experiment with the number of nearest neighbors we look at when embedding. How does changing this affect model performance? Why do you think this is?
4. The most important part of our model is the ConvLayer module, which performs a "similarity check" on the layer inputs (similar to a convolution operation). Note that this layer does not actually convolute our data, but performs a function that achieves the same goal. Experiment with changing the activation functions and batchnorms in the ConvLayer class. How does this affect performance?
5. Take a look at the pooling function in **model_hw.py**. We chose to use average pooling in our model, but how does using max pooling instead affect performance?

Part 4: Declaration of Contribution

The authors declare no conflicts of interest. All the team members contribute to this project equally. Yunyeong Choi led the project, found the relevant paper and collected the data. Yunyeong wrote the start part of the codes and drafted the first version of report. Xin Chen wrote the model construction and developing part of the codes. Xin organized the collecting and submission of codes and reports. Jinho Shin and Aryan Chakrabarti transformed the codes into homework, in the form of both written and programming. All the people participated into debugging, revision, and report writing.

Part 5: Link to code

All related codes and data can be found here: https://github.com/X-Chen97/cs182project
We added our notebook (pdf version) at the end of the document too.

References

[1] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133–A1138, Nov 1965.

[2] Jörg Behler and Michele Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Phys. Rev. Lett.*, 98:146401, Apr 2007.

[3] Tian Xie and Jeffrey C. Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett.*, 120:145301, Apr 2018.

[4] Wikipedia. Space group — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Space%20group&oldid=1119183693`, 2023. [Online; accessed 23-March-2023].

CS182 project - Deliver ideas of CGCNN

General Explanation

The key idea of CGCNN is to represent crystal structure by a crystal graph, which encodes not only the atomic information but also bonding information between atoms based on the distance. Here we will let each atomic feature vector as vi, which encodes the property of the atom corresponding to node i. The edge feature vector is represented as u(i,j)k, which encodes k-th bond connecting atom i, j. Note that there can be several bonds between atoms, which originate from the periodic nature of crystals.
Obviously, there are many ways to encode atomic and bond data. For the sake of simplicity, we will follow the methods used in the CGCNN paper, which used a pre-defined encoding vector to change atoms into vectors. These vectors have 92 dimensions, and for 100 atoms in the periodic table, they have different encodings which consist of 0 and 1. For the edge feature vectors, there also can be many ways to encode bond information; bond length, angle, and covalency. However, we will only use bond-length information between nearest neighbors, by applying a Gaussian kernel to change it to encoding vectors. One can follow the homework notebook on how this embedding is done to NaCl(salt) Crystal.

Encoding Crystal into Crystal Graph

In [3]:
```
# Note: Need pymatgen
```

In [2]:
```
# Uncommnet this to install pymatgen
#!pip install pymatgen
```

In [ ]:
```
import os
import sys
import csv
import json
import torch
import torch.nn as nn
import random
import warnings
warnings.filterwarnings('ignore')
import functools
import sklearn
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.colors import to_rgba_array

from pymatgen.core.structure import Structure
from pymatgen.analysis.graphs import StructureGraph
from pymatgen.analysis.local_env import CrystalNN
from torch.utils.data import Dataset, DataLoader
from torch.utils.data.dataloader import default_collate
from torch.utils.data.sampler import SubsetRandomSampler
import torch.optim as optim
from torch.optim.lr_scheduler import MultiStepLR
from data_utils import CIFData
from data_utils import AtomCustomJSONInitializer
from data_utils import AtomInitializer
from data_utils import GaussianDistance
```

In [4]:
```
# Uncomment this to extract data
# !unzip cgcnn_data.zip
# !unzip hw_data.zip
```

Let's convert salt (NaCl) to crystadddl graph.
You can use print method to see the lattice and position of Na, Cl atoms in the cell.
We will also visualize it to a graph. Each node of graph is atom in the crystal, and edge is bonds that have distance as information.

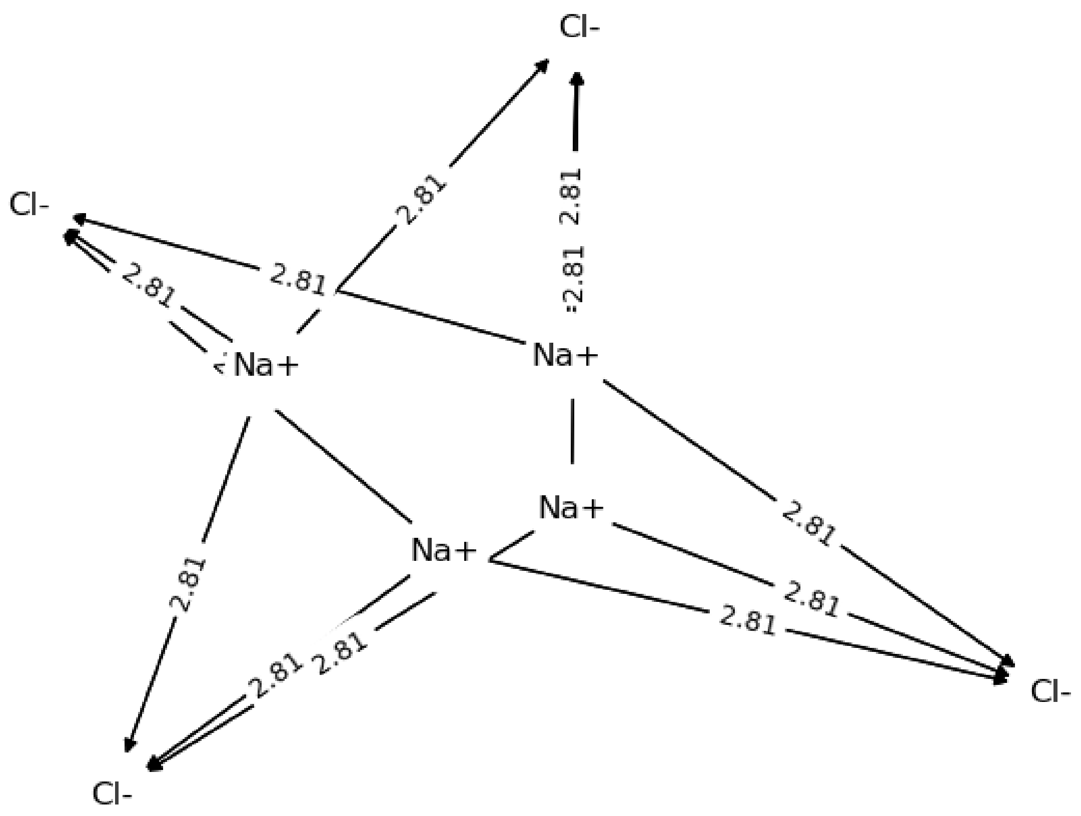
Note that we will use pymatgen package (Python Materials Genomics) which is an open source Python library for materials analysis. But you don't need to worry about that too much. Only important class is 'Structure' class, which can load structure data of materials by Structure.from_file method. After you load it, you can get the site information using Structure_object[index_want_to_check], and this Site object has some useful attributes, such as specie. You can check what specie is in the site by using Structure_object[index].specie

In [5]:
```
# Load structure from CIF file
nacl = Structure.from_file('hw_data/sample_regression/1000041.cif')
print(nacl)
```

```
Full Formula (Na4 Cl4)
Reduced Formula: NaCl
abc   :   5.620000   5.620000   5.620000
angles:  90.000000  90.000000  90.000000
pbc   :       True       True       True
Sites (8)
  #  SP      a    b    c
---  ----  ---  ---  ---
  0  Na+   0    0    0
  1  Na+   0    0.5  0.5
  2  Na+   0.5  0    0.5
  3  Na+   0.5  0.5  0
  4  Cl-   0.5  0.5  0.5
  5  Cl-   0.5  0    0
  6  Cl-   0    0.5  0
  7  Cl-   0    0    0.5
```

In [6]:
```
# Visualization of the structure using the CIF file
crystal = CrystalNN()

sg = StructureGraph.with_local_env_strategy(nacl, crystal)

node_labels = {}
for i, site in enumerate(nacl):
    label = "{}".format(site.specie)
    node_labels[i] = label

edge_weights = {}
for edge in sg.graph.edges():
    dist = nacl.get_distance(edge[0], edge[1])
    edge_weights[edge] = dist

pos = nx.spring_layout(sg.graph)
```

```
nx.draw(sg.graph, pos, with_labels=True, labels=node_labels, font_size=12, node_
nx.draw_networkx_edge_labels(sg.graph, pos, edge_labels=edge_weights, font_size=
plt.axis("off")
plt.show()
```



Next we will encode atoms in the node to atomistic feature vectors using pre-difined atom embedding.

atom_init.json is containing vector embedding of atoms, where key (1, 2, 3, ..., 100) represent atomic number and values are encoding vectors.

You can try different atom embedding too if you want.

In [ ]:

```
# Load embedding file.
element_embedding_file = 'hw_data/sample_regression/atom_init.json'
with open(element_embedding_file) as f:
    elem_embedding = json.load(f)
elem_embedding = {int(key): value for key, value
                  in elem_embedding.items()}

atom_fea = None
##############################################################################
# TODO: Encode crsytal data to atomistic features.
# (Hint: Use np.vstack and elem_embedding to join all embedded specie numbers of
# Also, check out pymatgen.core.structure.specie.number attribute.
# nacl[i].specie will give you what kind of specie it is,
# And number attribute will give atomic number of that specie.)
# Atom_feature shd have shape of (# of atoms, len(embedding vector))

# YOUR CODE HERE
# Solution
atom_fea = np.vstack([elem_embedding[nacl[i].specie.number]
                      for i in range(len(nacl))])
############################################################################
assert atom_fea is not None
atom_fea = torch.Tensor(atom_fea)


assert atom_fea.shape == (8,92)
```

Next, we will get neighbor information from each atoms in the cell. We will get help from pymatgen package. get_all_neighbor function of structure object returns atoms within the input radius. Note that here len(all_nbrs) is 8 since there are 8 atoms (4 Na+, 4 Cl-) in the cell. Each list contain the neighbor atom information considering periodicity. We will use 12 nearest neighbors after sorting with distance.

In [8]:
```
all_nbrs = nacl.get_all_neighbors(r = 8, include_index=True)
all_nbrs = [sorted(nbrs, key=lambda x: x[1]) for nbrs in all_nbrs]

assert len(all_nbrs) == 8

nbr_fea_idx, nbr_fea = [], []
for nbr in all_nbrs:
    # Note: x[1] returns distance to neighbors.
    # Note: x[2] returns index of original structure object
    nbr_fea_idx.append(list(map(lambda x: x[2],
                            nbr[:12])))
    nbr_fea.append(list(map(lambda x: x[1],
                        nbr[:12])))

# nbr_fea_idx contain information of nearest neighbor atoms
# from ith row (ith atom in the cell)
# For example, 0th atom (Na+ (0.0000, 0.0000, 0.0000)) is neighbored
# with 5th, 6th, 7th, etc...
# nbr_fea contain information of nearest neighbot distance.
nbr_fea_idx, nbr_fea = np.array(nbr_fea_idx), np.array(nbr_fea)
nbr_fea_idx = torch.LongTensor(nbr_fea_idx)

nbr_fea_idx_standard = torch.LongTensor([[5, 6, 7, 7, 6, 5, 2, 1, 2, 1, 3, 3],
                                         [4, 7, 6, 7, 6, 4, 2, 0, 3, 2, 3, 3],
                                         [4, 7, 5, 7, 5, 4, 0, 3, 1, 3, 3, 1],
                                         [4, 6, 5, 6, 5, 4, 2, 1, 1, 2, 0, 0],
                                         [3, 1, 2, 3, 2, 1, 6, 5, 6, 5, 7, 7],
                                         [2, 3, 0, 3, 0, 2, 4, 4, 6, 6, 4, 7],
                                         [1, 3, 0, 3, 0, 1, 4, 4, 5, 5, 5, 4],
                                         [2, 1, 0, 2, 1, 0, 4, 5, 5, 4, 6, 6]])

assert torch.equal(nbr_fea_idx, nbr_fea_idx_standard)
```

Now we have two features, atomic feature and neighbor feature. Note that neighbor feature is discontionous information with respect to the distance. Therefore we will expand (which means we will stack data based on the current discontinous distance value) neighbor feature using Gaussian Kernel (or Gaussian filter) https://en.wikipedia.org/wiki/Gaussian_filter
While expanding the neighbor feature, one can set the distance range and step. Here we will use minimum distance 0, maximum distance 12, and step size of 0.2, which will expanded to dimension size of 61.

In [9]:
```
dmin = 0
dmax = 12
step = 0.2
var = step
filter_step = np.arange(dmin, dmax+step, step)

def expand(distances):
    ##########################################################################
    # TODO: Implement the expand function using a Gaussian kernel.
    # raise NotImplementedError()

    # Solution
    return np.exp(-(distances[..., np.newaxis] - filter_step)**2 / var**2)
    ##########################################################################


nbr_fea_gaussian = expand(nbr_fea)
```

In [10]:
```
gdf = GaussianDistance(dmin=0, dmax=12, step=0.2)
nbr_fea = gdf.expand(nbr_fea)
```

In [11]:
```
assert np.array_equal(nbr_fea_gaussian, nbr_fea)
```

Build a Model

After building the crystal graph, the convolutional neural network act on top of the graph, and consecutive fully connected networks and pooling layer are used to predict the property of crystals. The convolution layer iteratively updates the atom feature vector v_i by message passing with surrounding atoms and bonds with a non-linear graph convolution function:

$$v_i^{(t+1)} = Conv(v_i^{(t)}, v_j^{(t)}, u_{(i,j)_k}), (i,j)_k \in G$$

And then consequetive pooling layer and hidden fully connected layer will generate output of model:

$$v_c = Pool(v_0^{(0)}, v_1^{(0)}, \ldots, v_N^{(0)}, \ldots, v_N^{(R)})$$

Note that we can use any Convolution and Pooling layer if we maintain the permutation invariance. Here we will try two different convolution layer. Try to implement WeightShareConvLayer in the TODO section of model_hw.py. We will offer solution for the ConvLayer. Note that we will do not change pooling layer, and you can use any activation function you use. But try to use sigmoid function here. After you implement WeightShareConvLayer, try to implement CrystalGraphConvNet by filling in the TODO section of model_hw.py. You can find a detailed explanation there.

1. WeightShareConvLayer; This is a Convolution weight matrix is shared by all neighbors.
   $v_i^{(t+1)} = g[(\sum_{j,k} v_j^{(t)} \oplus u_{(i,j)_k})W_c^{(t)} + v_i^{(t)} W_s^{(t)} + b^{(t)}]$
   Here W_c, W_s, and b are the convolution weight matrix, self-weight matrix, and bias.

1. ConvLayer; This is a Convolution weight matrix is differentiated at all neighbor pairs.
   $v_i^{(t+1)} = v_i^{(t+1)} + \sum_{j,k} \sigma(z_{(i,j)_k}^{(t)} W_f^{(t)} + b_f^{(t)}) \odot g(z_{(i,j)_k}^{(t)} W_s^{(t)} + b_s^{(t)})$
   Here $z_{(i,j)_k} = v_i \oplus v_j \oplus u_{(i,j)_k}$, $\odot$ denotes element-wise multiplication and σ denotes a sigmoid function.

In [12]:
```
from model_hw import CrystalGraphConvNet
from model_hw import ConvLayer
from model_hw import WeightShareConvLayer
```

In [13]:
```
# For testing purpose
data_dir = './cgcnn_data/sample-regression'
test = CIFData(data_dir)
(atom_fea, nbr_fea, nbr_fea_idx), target, cif_id = test[-1]
structures, _, _ = test[-1]
orig_atom_fea_len = structures[0].shape[-1]
nbr_fea_len = structures[1].shape[-1]
```

In [14]:
```
torch.manual_seed(123)

convlayer_test = ConvLayer(92, 41)
convlayer_test_result = convlayer_test.forward(atom_fea, nbr_fea, nbr_fea_idx)

#############################################################################
# TODO: Implement the WeightShareConvLayer model in model_hw.py
#############################################################################

weightshare_test = WeightShareConvLayer(92, 41)
weightshare_test_result = weightshare_test.forward(atom_fea, nbr_fea, nbr_fea_id

assert convlayer_test_result.shape == (8, 92)
assert weightshare_test_result.shape == (8, 92)
```

In [15]:
```
# Simple test
# Set seed using manual_seed
# TODO explanation about crystal_atom_idx
torch.manual_seed(123)
crystal_atom_idx = [torch.Tensor([0, 1, 2, 3, 4, 5, 6, 7]).long()]
#############################################################################
# TODO: Implement the CrystalGraphConvNet model in model_hw.py
#############################################################################

model1 = CrystalGraphConvNet(orig_atom_fea_len, nbr_fea_len, option='C')
out = model1.forward(atom_fea, nbr_fea, nbr_fea_idx, crystal_atom_idx)
assert (out.item() - 0.6627038) < 1e-6

model2 = CrystalGraphConvNet(orig_atom_fea_len, nbr_fea_len, option='WC')
out = model2.forward(atom_fea, nbr_fea, nbr_fea_idx, crystal_atom_idx)
assert (out.item() - -0.5442343) < 1e-6
```

Training

Now we are all set to test our model. We will train our model to simplified dataset, which consists of ~700 Perovskite materials. We will predict the energy (which is calculated already with DFT, unit is eV) of Perovskites, and see how the model performs. After run the following section, try to anser the question. Note that you can find your prediction and corresponding structures in test_result.csv file.

Q. If you built a model, let's train and visualize it. What is the difference between two models? Which one performs better and why do you think so?
Based on what you observed, can you make a convolution layer that performs better?

Hint: Among two convolution layers, ConvLayer performs better in general. This is because the local environments can always change, so maintaining same weights to all local environments makes prediction worse. However, you can see that in Perovskite dataset, WeightShareConvLayer performs better because Perovskite all have similar symmetry, and local environments of them are not much different

In [16]:
```
from data_utils import collate_pool, get_train_val_test_loader
from train_utils import Normalizer, train, validate, save_checkpoint
from model_hw import CrystalGraphConvNet
from random import sample
```

In [17]:
```
torch.manual_seed(123)

# set parameters
data_dir = './hw_data/perovskite_energy'
batch_size = 8
train_ratio, val_ratio, test_ratio = 0.8, 0.1, 0.1

# get dataset

dataset = CIFData(data_dir)
collate_fn = collate_pool
train_loader, val_loader, test_loader = get_train_val_test_loader(
    dataset=dataset,
    collate_fn=collate_fn,
    batch_size=batch_size,
    train_ratio=train_ratio,
    val_ratio=val_ratio,
    test_ratio=test_ratio,
    return_test=True)
```

In [18]:
```
# normalize target

if len(dataset) < 500:
    warnings.warn('Dataset has less than 500 data points. '
                  'Lower accuracy is expected. ')
    sample_data_list = [dataset[i] for i in range(len(dataset))]
else:
    sample_data_list = [dataset[i] for i in
                        sample(range(len(dataset)), 500)]
_, sample_target, _ = collate_pool(sample_data_list)
normalizer = Normalizer(sample_target)
```

In [19]:
```
# build model
structures, _, _ = dataset[0]
orig_atom_fea_len = structures[0].shape[-1]
nbr_fea_len = structures[1].shape[-1]
############################################################################
```

```
# TODO: Tune the following parameters and model to optimize performance.
# Note: You can change your convolution layer with option argument in
# CrystalGraphConvNet. C is ConvLayer, WC is WeightShareConvLayer.
################################################################################
# number of hidden atom features in conv layers
atom_fea_len = 64
# number of hidden features after pooling
h_fea_len = 128
# number of conv layers
n_conv = 3
# number of hidden layers after pooling
n_h = 1

model = CrystalGraphConvNet(orig_atom_fea_len, nbr_fea_len,
                            atom_fea_len=atom_fea_len,
                            n_conv=n_conv,
                            h_fea_len=h_fea_len,
                            n_h=n_h, option='C')
```

In [20]:

```
# set hyperparameters, change the parameters to get better results
epochs = 15
criterion = nn.MSELoss()
lr = 0.01
momentum = 0.9
weight_decay = 0

optimizer = optim.SGD(model.parameters(), lr,
                              momentum=momentum,
                              weight_decay=weight_decay)

# optimizer = optim.Adam(model.parameters(), lr,
#                        weight_decay=weight_decay)
lr_milestones = [100]
scheduler = MultiStepLR(optimizer, milestones=lr_milestones,
                            gamma=0.1)
```

In [21]:

```
running_record = {'train_loss': [], 'train_mae': [],
                    'val_loss': [], 'val_mae': []}

for epoch in range(epochs):
    print('Epoch {}/{}'.format(epoch, epochs - 1))
    print('-' * 10)

    best_mae_error = 1e10
    # train for one epoch
    train_loss, train_mae = train(train_loader, model, criterion, optimizer, epo

    # evaluate on validation set
    val_loss, val_mae = validate(val_loader, model, criterion, normalizer)

    # append loss and mae to running record. Convert tensor to float if necessar
    running_record['train_loss'].append(train_loss)
    running_record['train_mae'].append(train_mae)
    running_record['val_loss'].append(val_loss)
    running_record['val_mae'].append(val_mae)


    if val_mae != val_mae:
        print('Exit due to NaN')
        sys.exit(1)
    scheduler.step()

    # remember the best mae_eror and save checkpoint
    is_best = val_mae < best_mae_error
    best_mae_error = min(val_mae, best_mae_error)

    save_checkpoint({
        'epoch': epoch + 1,
        'state_dict': model.state_dict(),
        'best_mae_error': best_mae_error,
        'optimizer': optimizer.state_dict(),
        'normalizer': normalizer.state_dict(),
    }, is_best)
```

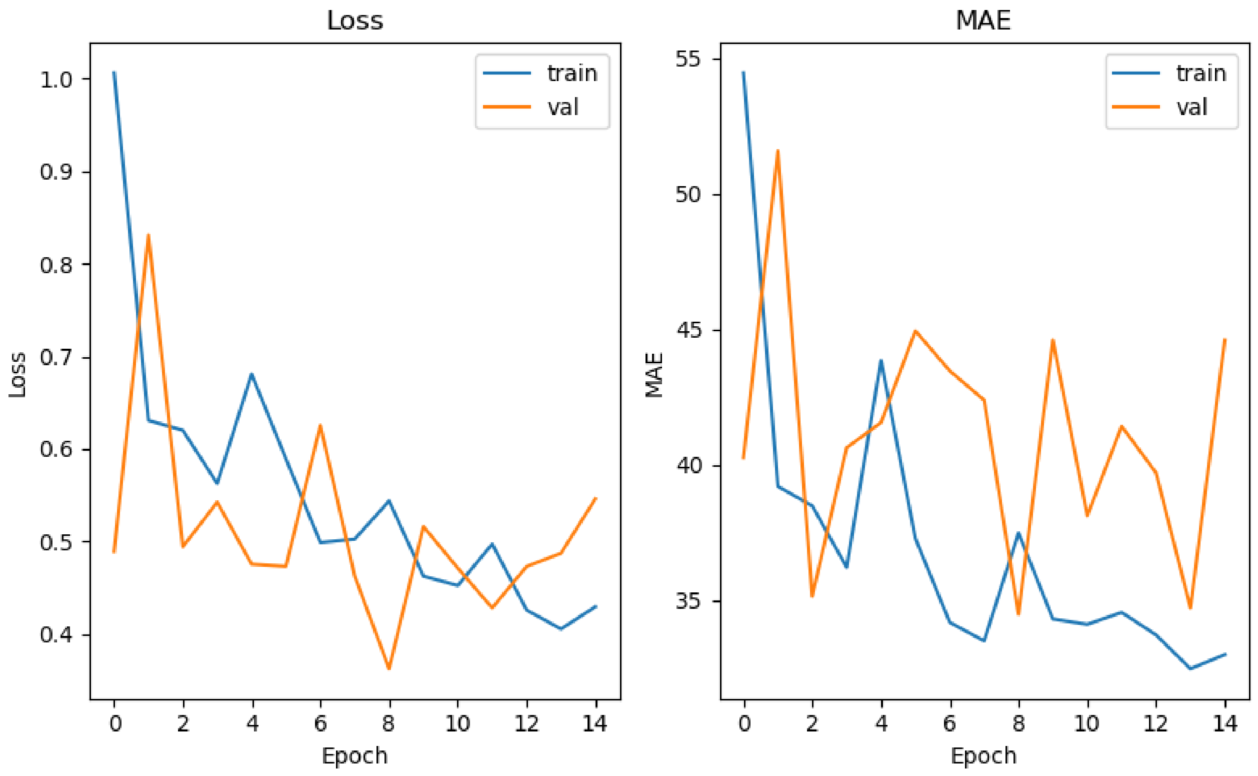
```
Epoch 0/14
----------
100%|██████████████████████████████████████████| 67/67 [00:21<00:00,  3.12it/s]
Train:  Time 0.246      Data 0.168      Loss 1.0064     MAE 54.459
100%|████████████████████████████████████████████| 9/9 [00:08<00:00,  1.03it/s]
Test:   Time 0.412      Loss 0.4892     MAE 40.253
Epoch 1/14
----------
100%|██████████████████████████████████████████| 67/67 [00:19<00:00,  3.40it/s]
Train:  Time 0.220      Data 0.152      Loss 0.6307     MAE 39.189
100%|████████████████████████████████████████████| 9/9 [00:08<00:00,  1.04it/s]
Test:   Time 0.405      Loss 0.8308     MAE 51.584
Epoch 2/14
----------
100%|██████████████████████████████████████████| 67/67 [00:18<00:00,  3.54it/s]
Train:  Time 0.208      Data 0.150      Loss 0.6201     MAE 38.471
100%|████████████████████████████████████████████| 9/9 [00:08<00:00,  1.04it/s]
Test:   Time 0.406      Loss 0.4942     MAE 35.135
Epoch 3/14
----------
100%|██████████████████████████████████████████| 67/67 [00:19<00:00,  3.49it/s]
Train:  Time 0.212      Data 0.149      Loss 0.5625     MAE 36.203
100%|████████████████████████████████████████████| 9/9 [00:13<00:00,  1.46s/it]
Test:   Time 0.904      Loss 0.5425     MAE 40.618
Epoch 4/14
----------
100%|██████████████████████████████████████████| 67/67 [00:19<00:00,  3.50it/s]
Train:  Time 0.211      Data 0.150      Loss 0.6808     MAE 43.837
100%|████████████████████████████████████████████| 9/9 [00:08<00:00,  1.06it/s]
Test:   Time 0.389      Loss 0.4754     MAE 41.545
Epoch 5/14
----------
100%|██████████████████████████████████████████| 67/67 [00:19<00:00,  3.36it/s]
Train:  Time 0.223      Data 0.156      Loss 0.5892     MAE 37.267
100%|████████████████████████████████████████████| 9/9 [00:08<00:00,  1.05it/s]
Test:   Time 0.395      Loss 0.4730     MAE 44.930
Epoch 6/14
----------
100%|██████████████████████████████████████████| 67/67 [00:19<00:00,  3.46it/s]
Train:  Time 0.214      Data 0.153      Loss 0.4989     MAE 34.168
100%|████████████████████████████████████████████| 9/9 [00:09<00:00,  1.01s/it]
Test:   Time 0.458      Loss 0.6255     MAE 43.454
Epoch 7/14
----------
```

```
100%|████████████████████████████████████████████| 67/67 [00:19<00:00,  3.51it/s]
Train:  Time 0.210      Data 0.151      Loss 0.5025     MAE 33.486
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.05it/s]
Test:   Time 0.405      Loss 0.4627     MAE 42.372
Epoch 8/14
----------
100%|████████████████████████████████████████████| 67/67 [00:18<00:00,  3.56it/s]
Train:  Time 0.206      Data 0.149      Loss 0.5437     MAE 37.471
100%|██████████████████████████████████████████████| 9/9 [00:09<00:00,  1.04s/it]
Test:   Time 0.484      Loss 0.3625     MAE 34.471
Epoch 9/14
----------
100%|████████████████████████████████████████████| 67/67 [00:18<00:00,  3.55it/s]
Train:  Time 0.207      Data 0.149      Loss 0.4624     MAE 34.295
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.06it/s]
Test:   Time 0.393      Loss 0.5161     MAE 44.592
Epoch 10/14
----------
100%|████████████████████████████████████████████| 67/67 [00:18<00:00,  3.62it/s]
Train:  Time 0.202      Data 0.142      Loss 0.4524     MAE 34.097
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.05it/s]
Test:   Time 0.401      Loss 0.4711     MAE 38.106
Epoch 11/14
----------
100%|████████████████████████████████████████████| 67/67 [00:20<00:00,  3.35it/s]
Train:  Time 0.224      Data 0.159      Loss 0.4972     MAE 34.538
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.05it/s]
Test:   Time 0.401      Loss 0.4282     MAE 41.408
Epoch 12/14
----------
100%|████████████████████████████████████████████| 67/67 [00:18<00:00,  3.59it/s]
Train:  Time 0.204      Data 0.147      Loss 0.4259     MAE 33.707
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.06it/s]
Test:   Time 0.392      Loss 0.4729     MAE 39.691
Epoch 13/14
----------
100%|████████████████████████████████████████████| 67/67 [00:19<00:00,  3.52it/s]
Train:  Time 0.209      Data 0.149      Loss 0.4054     MAE 32.458
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.05it/s]
Test:   Time 0.399      Loss 0.4873     MAE 34.707
Epoch 14/14
----------
100%|████████████████████████████████████████████| 67/67 [00:18<00:00,  3.64it/s]
Train:  Time 0.200      Data 0.138      Loss 0.4295     MAE 32.980
100%|██████████████████████████████████████████████| 9/9 [00:08<00:00,  1.05it/s]
Test:   Time 0.398      Loss 0.5460     MAE 44.589
```

In [22]:
```
# test best model
print('---------Evaluate Model on Test Set---------------')
best_checkpoint = torch.load('model_best.pth.tar')
model.load_state_dict(best_checkpoint['state_dict'])
_, l=validate(test_loader, model, criterion, normalizer, test=True)
```

```
---------Evaluate Model on Test Set---------------
100%|██████████████████████████████████████████████| 9/9 [00:09<00:00,  1.01s/it]
Test:   Time 0.459      Loss 1.2363     MAE 53.165
```

In [23]:
```
# visualize the training and val loss
# visualize the training and val mae
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

fig, ax = plt.subplots(1, 2, figsize=(8, 5))
ax[0].plot(running_record['train_loss'], label='train')
ax[0].plot(running_record['val_loss'], label='val')
ax[0].set_xlabel('Epoch')
ax[0].set_ylabel('Loss')
ax[0].legend()
ax[0].set_title('Loss')
# set x axis to be integer
ax[0].xaxis.set_major_locator(MaxNLocator(integer=True))

ax[1].plot(running_record['train_mae'], label='train')
ax[1].plot(running_record['val_mae'], label='val')
ax[1].set_xlabel('Epoch')
ax[1].set_ylabel('MAE')
ax[1].legend()
ax[1].set_title('MAE')
# set x axis to be integer
ax[1].xaxis.set_major_locator(MaxNLocator(integer=True))

plt.tight_layout()
plt.show()
```



In [ ]: