Master Thesis

From Potential Energy Surface to Gas Adsorption via Deep Learning

Antonios P. Sarikas

chempp1160@edu.chemistry.uoc.gr

Supervised by
George E. Froudakis

Materials Modeling
&
Design Group

Department of Chemistry

University of Crete

Contents

Contents

List of Figures

List of Algorithms

METAL-ORGANIC FRAMEWORKS, or in short MOFs, thanks to their ultra high porosity and surface area, are deemed as prominent candidates for applications involving gas adsorption. However, their intrinsic combinatorial nature, translates to a practically infinite material space, rendering the identification of novel materials with traditional methods cumbersome. Over the last years, machine learning approaches based on predictive models have been developed, allowing researchers to rapidly screen large databases of MOFs. The quality of these models highly depends on the mathematical representation of a material, thus necessitating the use of informative inputs. In this thesis, we propose a generalized framework for predicting gas adsorption properties, using as one and only input the potential energy surface. We treat the latter as a 3D energy image and then pass it through 3D convolutional neural network, known for its ability to process image-like data. The proposed pipeline is applied in MOFs for predicting CO_2 uptake. The resulting model outperforms both in terms of performance and data efficiency a conventional one built upon textual properties. Additionally, we demonstrate the transferability of the approach to other host-guest systems, by examining CH_4 uptake in covalent organic frameworks. Finally, discussion for improving and extending the suggested scheme is provided.

Chapter 1

Introduction

RETICULAR CHEMISTRY, a field that bridges inorganic and organic chemistry (Yaghi 2020), has emerged from a simple albeit powerful idea: *combining molecular building blocks to form extended crystalline structures* (Yaghi 2019). It all started in 1990s, with the advent of metal-organic frameworks (MOFs), the first "offspring" of reticular chemistry. MOFs, a class of nanoporous materials *composed of metal ions or clusters coordinated to organic ligands aka organic linkers*, possess extraordinary properties, such as ultrahigh porosity and huge surface areas (Farha et al. 2012). To get a sense of how extraordinary these materials are, it is suffice to say that *one gram of such a material can have a surface area as large as a soccer field*. The fact that reticular materials are "brought to life" by combining simple building blocks, allows chemists and material scientists to design materials in a judicious manner. The epitome of design in reticular chemistry is found in the synthesis of a zirconium-based MOF (Alezi et al. 2016), incorporating the polybenzene network or "cubic graphite" structure, predicted about 70 years ago.

1.1 Applications of Reticular Chemistry

Owing to their aforedescribed properties along with their extremely tunable and modular nature, MOFs have been considered prominent solutions for gas-adsorption related problems (Y. Li et al. 2007; Jiang et al. 2022). MOFs find application in fields such as gas storage and separation, catalysis and drug delivery, just to name a few.

Carbon capture is a prime example (An et al. 2009; Sumida et al. 2011; Qazvini et al. 2021), where MOF-based sorbents have been deemed as green, low-cost and energy-efficient solutions. These materials provide versatile solutions to carbon capture, spanning various phases of the capture process, with direct air capture (DAC) being a noteworthy example. DAC includes chemical or physical methods for extracting carbon dioxide directly from the ambient air, with MOF-powered DAC showing great potential as a green and sustainable strategy for reducing carbon dioxide levels, contributing to the combating of climate change (Bose et al. 2023).

Hydrogen storage is one of the greatest challenges of hydrogen economy, currently inhibiting the transition from fossil fuels to hydrogen. Fortunately, characteristics of MOF adsorbents such as fast adsorption/desorption kinetics, low operating pressures and high hydrogen capacities, render them as promising answers to the aforementioned challenge (Suh et al. 2011; Suresh et al. 2021).

Methane is an attractive fuel for vehicular applications, being a relatively clean-burning fuel compared to gasoline. ***Methane storage*** in sorbents known as adsorbed natural gas (ANG) exhibit advantages over compressed natural gas (CNG) and liquefied natural gas (LNG), both in terms of energy-efficiency and vehicular safety. MOFs (S. Ma et al. 2007; Spanopoulos et al. 2016; Tsangarakis et al.

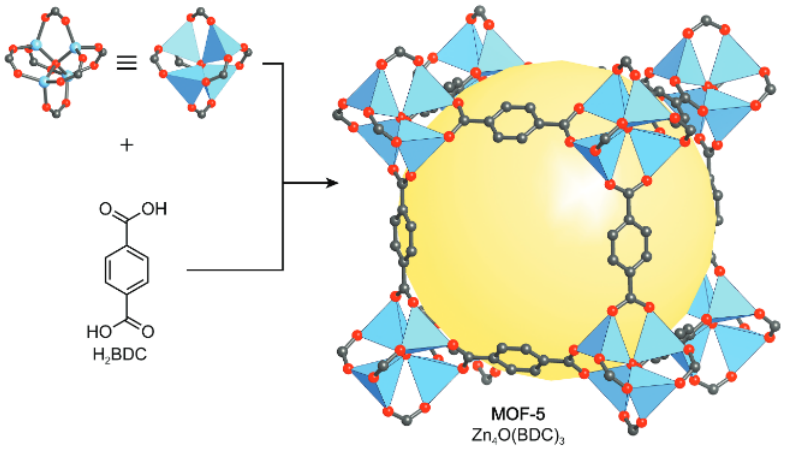


Figure 1.1: Unit cell structure of MOF-5 or IRMOF-1, one of the highest surface area to volume ratio among MOFs, at 2200 m^2 cm^3 and the first MOF studied for hydrogen storage (Rosi et al. 2003).

2023) and their "reticular siblings" covalent organic frameworks (COFs)—composed only of light elements—show great promise as ANG solutions (Furukawa et al. 2009; Mendoza-Cortes et al. 2011; Martin et al. 2014; Tong et al. 2018).

1.2 The Problem

The intrinsic combinatorial character of reticular chemistry, translates to practically an infinite number of realizable structures. Currently, the Cambridge Structural Database (CSD) contains more than 100 000 experimentally synthesized MOFs (Moghadam, A. Li, et al. 2017) while the arrival of in silico designed MOFs (Wilmer et al. 2011; Colón et al. 2017; Boyd et al. 2019; Chung, Haldoupis, et al. 2019; Lee et al. 2021; Rosen et al. 2021; De Vos et al. 2023) has immensely expanded the available material pool. The huge size of current and future MOF databases (Lee et al. 2021) is both a blessing and a curse for the identification of novel materials. Blessing, since a large number of candidate structures doesn't limit our choices and as such, the chances to find the right material for a given problem. Curse, since the enormous size of MOFs space makes it harder for researchers to efficiently explore it, complicating the tracing of materials with the desired properties. It is therefore crucial to find a way that allows us to efficiently explore such a huge material space (see Figure 1.2). Another way to rephrase our problem is the following: ***Given a large catalog of MOFs, is there a way to efficiently filter out the most promising ones for the application of interest?***

As a first approach to deal with this challenge, one could, in principle experimentally synthesize and characterize each one of the materials listed in the given catalog. Although *experimental synthesis and characterization* is the ultimate way to assess the performance of a material[1], the fact that a single laboratory study can take days or even months, renders experimental techniques impractical. A more efficient approach is computational screening based on *molecular simulations*, which for years has served

[1] As Richard Feynmann said: *"The test of all knowledge is experiment. Experiment is the sole judge of scientific truth"*.

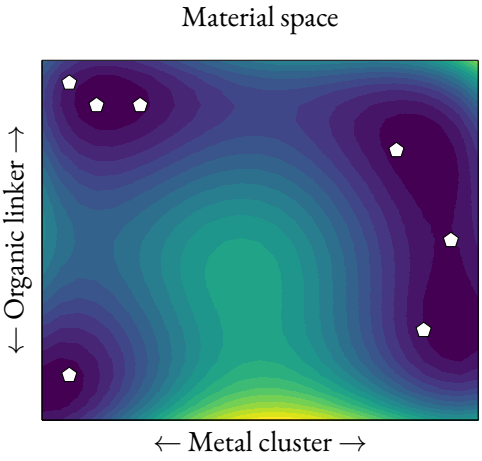


Figure 1.2: Material space of MOFs. Each point in this space corresponds to a unique combination of organic linker and metal cluster, whereas the associated color denotes the "score" of material for a given application. Finding the best material (the pentagons) for a given application, amounts to solving a (probably) non-convex optimization problem.

as the principal tool for the discovery of high-performing MOFs (Simon, J. Kim, et al. 2015; Banerjee et al. 2016; Gómez-Gualdrón et al. 2016; Jeong et al. 2017; Moghadam, Islamoglu, et al. 2018). Although computational screening dramatically accelerates the assessment of a single material compared to experimental techniques, brute-force screening of current and upcoming databases is considered suboptimal, given the size of the latter.

Machine learning (ML) aka *data-driven techniques* come to the rescue when dealing with *big data* and over the last years have picked up the torch from molecular simulations regarding material characterization. Given a collection of *input-output* pairs, i.e. a mathematical representation of a material and a corresponding property, a ML algorithm[2] seeks to *uncover the underlying structure-property relationship*. To put it in a nutshell, a ML algorithm "eats" *data*—which may come either from experiments, simulations or a combination of the two—and "spits out" a *model*, which can be *used to sort a large catalog of MOFs in just few seconds*. Obviously, for ML approaches to be effective and reliable, it is necessary that the resulting models are of high quality.

1.3 Literature Review

One of, if not the most important factor for the performance of ML models, is the way we select to mathematically represent materials or molecules. In other words, the type and amount of chemical information that is "injected" into these representations commonly known as *descriptors*, can make the difference between a high-performing and a baseline model. As such, it is of uttermost importance to employ descriptors that provide sufficient information for the properties of materials or molecules we are interested in to predict.

[2] Note that ML algorithms are not limited to solve only such kind of problems, which fall under the umbrella of supervised learning. See Section 2.1.1 for other types of problem tackled by ML.

With regards to gas adsorption in MOFs, one of the first and most commonly used descriptors, are the so called *geometric* ones, which aim to capture the pore environment of the framework. This type of descriptors includes textual characteristics of MOFs such as void fraction, gravimetric surface area and pore limiting diameter. Although ML models build with these descriptors work particularly well at the high pressure regime (Fernandez et al. 2013; Dureckova et al. 2019; Wu et al. 2020), their performance deteriorates when adsorption takes place at low pressures or the guest molecules exhibit non-negligible electrostatic interactions with the framework atoms. This performance drop should be expected, since geometric descriptors completely ignore the "cornerstone" of adsorption: *host-guest interactions*.

In order to improve the performance of ML models and bypass the limitations of geometric descriptors in the aforementioned conditions, another type of descriptors known as *energy-based* descriptors (Simon, Mercado, et al. 2015; Fanourgakis, Gkagkas, Tylianakis, Klontzas, et al. 2019; Orhan et al. 2023; Shi et al. 2023), has been introduced. This type of descriptors supply ML algorithms with information regarding the energetics of adsorption, and can be used standalone or in combination with geometric descriptors.

In one of the first works to fingerprint the energetic landscape of MOFs (Bucior, Bobbitt, et al. 2019), energy histograms derived from the interactions of guest molecules with the framework atoms were used to predict hydrogen and methane uptake with remarkable accuracy. Prior to calculating the energy histograms, a 3D grid is overlayed on the unit cell of the MOF. Next, at each point of the grid, the interaction between the guest molecule with the framework atoms is calculated, producing a 3D energy grid. The latter is finally converted into a histogram, by partitioning the energy values of the grid into bins of specific energy width. By using solely these histograms as descriptors—without including any textual property—Bucior, Bobbitt, et al. (2019) trained Lasso regression models, for predicting: i). H_2 swing capacity between 100 bar and 2 bar at 77 K ii). CH_4 swing capacity between 65 bar and 5.8 bar at 298 K. The resulting models were extremely accurate, achieving a mean absolute error (MAE) of 2.3 g L^{-1} and 12.9 cm^3 cm^{-3} for H_2 and CH_4, respectively, tested on the hMOFs database (Wilmer et al. 2011).

In another work (Fanourgakis, Gkagkas, Tylianakis, and Froudakis 2020), a set of descriptors based on the average interaction between fictitious probe particles and the framework atoms was introduced. Two different types of probe particles were proposed: i). Vprobe particles, which account for the van der Waals interactions ii). Dprobe particles, which are neutrally charged electric dipoles and account for the electrostatic interactions. Each of these fictitious probe particles is randomly inserted at different positions of the unit cell, and the interaction energy between the probe and the framework atoms is calculated. The interaction energies at the different positions are averaged out, producing an energetic fingerprint of the material. These fingerprints in combination with six geometric descriptors formed the input for the Random Forest (RF) algorithm, which was trained to predict gas uptake for a plethora of guest molecules and thermodynamic conditions, on the Computation-Ready Experimental (CoRE) MOF database (Chung, Camp, et al. 2014). The ML models showed impressive performance, showing an R^2 value of: i). 0.874 for H_2 uptake at 77 K and 2 bar ii). 0.889 for CH_4 uptake at 298 K and 5.8 bar. A highlight of this work was the exceptional performance of the ML model with regards to CO_2 uptake at 300 K and 0.1 bar, achieving an R^2 score of 0.832. At the same conditions, the ML model trained with geometric descriptors only achieved an R^2 score of 0.507. That is, the "injection" of energetic information resulted in 60 % increase in accuracy.

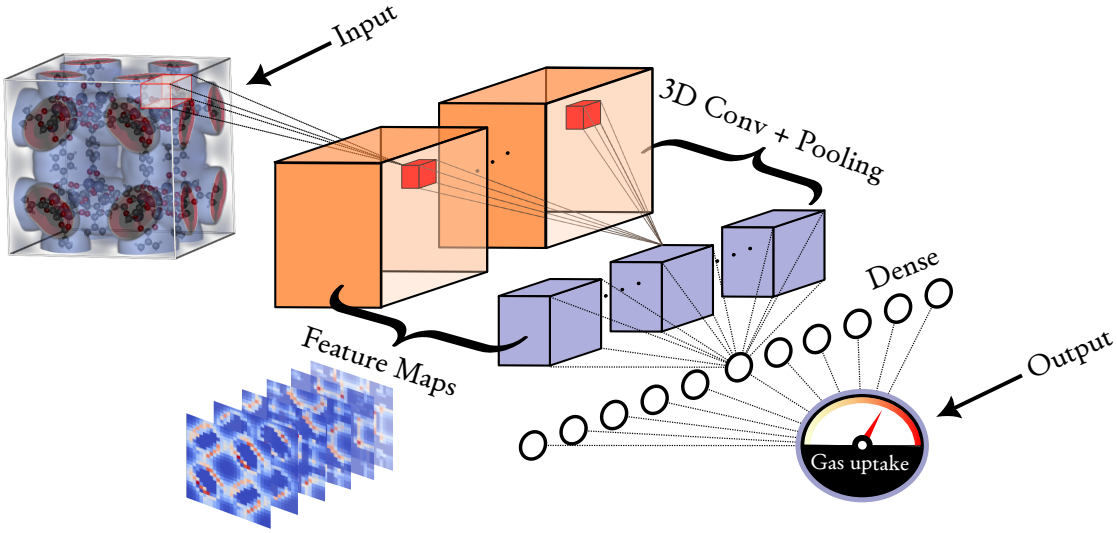


Figure 1.3: Proposed scheme to predict gas adsorption properties, starting from the PES as raw input. A 3D CNN extracts its features from the PES, and then uses them to predict the adsorption property of interest. The IRMOF-1 structure and PES were visualized with the iRASPA software (Dubbeldam et al. 2018).

1.4 Thesis Statement

In the aforedescribed works, a general pattern can be recognized with regards to the building of the ML models. Starting from the potential energy surface (PES) or an approximation thereof, energetic fingerprints are manually handcrafted based on some heuristic, and these fingerprints are then used to train a ML algorithm. However, a lot of information has been lost during the conversion of the PES into fingerprints, as a 3D object is converted into an 1D object. *Since gas adsorption comes down to the PES, it is reasonable to question whether one can use the PES itself as descriptor.* By doing this: i). The information content that goes into a ML algorithm is increased ii). The computational cost remains the same relative to the previously described works iii). It is no longer necessary to manually handcraft fingerprints.

In this thesis, ***a generalized framework to predict gas adsorption properties is proposed, using the PES as raw input****. In order to be machine understandable, the PES is first voxelized—the voxelized PES is essentially a 3D energy image—and then, it is processed by a 3D convolutional neural network (CNN), known for its ability to process image-like data. The proposed scheme is schematically presented* in Figure 1.3.

Chapter 2

Theoretical Background

DEEP LEARNING, *a class of ML algorithms based on neural networks (NNs)*, has revolutionized the way we tackle a problem from a ML perspective and is one if not the most important factor for recent ML achievements. Solving complex tasks such image classification or language translation, that for years have bedevilled traditional ML algorithms, constitutes the signature of deep learning (DL). Admittedly, *the advent of a deep CNN*, the AlexNet (Krizhevsky et al. 2012) on September 30 of 2012, signified the "modern birthday" of this field. On this day, AlexNet not only won the ImageNet (Deng et al. 2009) Large Scale Visual Recognition Challenge (ILSVRC), but dominated it, achieving a top-5 accuracy of 85 %, surpassing the runner-up which achieved a top-5 accuracy of 75 %. AlexNet showed that NNs are not merely a pipe-dream, but they can be applied in real-world problems. It is worth to notice that ideas of NNs trace back to 1943, but it was until recently that these ideas got materialized. The reason for this recent breakthrough of DL (and ML) is twofold. First, the availability of large datasets—the era of big data—such as ImageNet. Second, the increase in computational power, mainly of GPUs for DL, accelerating the training of deep NNs and traditional ML algorithms.

2.1 Machine Learning Preliminaries

Since DL is a subfield of ML, it is necessary to familiarize with the later before diving into the former. In this section, the minimum theoretical background and jargon of ML is presented. Machine learning can be defined as *"the science and (art) of programming computers so they can learn from the data"* (Géron 2017). A more technical definition is the following:

Definition 2.1 (Machine learning, Mitchell 1997)**.** *A computer program is said to learn from experience **E** with respect to some class of tasks **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**.*

For instance, a computer program that classifies emails into spam and non-spam (the task **T**), can improve its accuracy, i.e. the percentage of correctly classified emails (the performance **P**), through examples of spam and non-spam emails (the experience **E**). But in order to take advantage of the experience aka *data*, it must be written in such a way that *adapts to the patterns in the data*. Certainly, a *traditional spam filter can not learn from experience*, since the latter does not affect the classification rules of the former and as such, its performance. For a traditional spam filter to adapt to new patterns and perform better, it must change its hard-wired rules, but by then it will be a different program. In contrast, a *ML-based filter can adapt to new patterns, simply because it has been programmed to do so*. In other words, *in traditional programming we write rules for solving **T** whereas **in ML we write rules***

to learn the rules *for solving* ***T***. This subtle but essential difference is what gives ML algorithms the ability to take advantage of the data.

2.1.1 Learning paradigms

Depending on the type of experience they are allowed to have during their *training phase* (Goodfellow et al. 2016), ML approaches are divided into three main *learning paradigms*: ***unsupervised learning***, ***supervised learning*** *and* ***reinforcement learning***. The following definitions are not by any means formal, but merely serve as an intuitive description of the different paradigms.

Definition 2.2 (Unsupervised learning). *The experience comes in the form* $\mathcal{D}_{train} = \{\boldsymbol{x}_i\}$*, where* $\boldsymbol{x}_i \sim p(\boldsymbol{x})$ *is the input of the* i*-th training instance or sample. In this paradigm we are interested in learning useful properties of the underlying structure captured by* $p(\boldsymbol{x})$ *or* $p(\boldsymbol{x})$ *itself.*

For example, suppose we are interested in generating images that look like Picasso paintings. In this case, the input is just the pixel values, i.e. $\boldsymbol{x} \in \mathbb{R}^{W \times H \times 3}$. The latter follow a distribution $p(\boldsymbol{x})$, so all we have to do is to train an unsupervised learning algorithm with many Picasso paintings to get a *model*, that is $\hat{p}(\boldsymbol{x})$. Assuming the estimation of the original distribution is good enough, new realistically looking paintings (with respect to original Picasso paintings) can be "drawn" by just sampling from $\hat{p}(\boldsymbol{x})$. In the ML parlance, this task is known as *generative modeling* while inputs are also called *features, predictors or descriptors*.

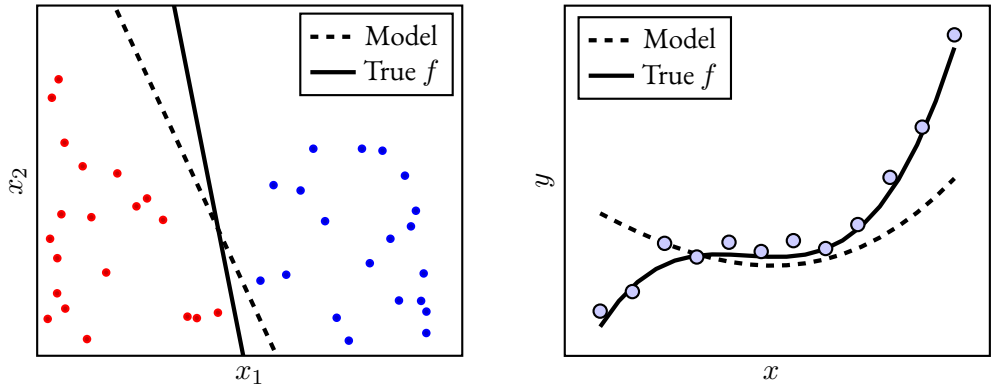
Definition 2.3 (Supervised learning). *The experience comes in the form* $\mathcal{D}_{train} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$*, where* $(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim p(\boldsymbol{x}, \boldsymbol{y})$ *and* $\boldsymbol{y}_i$ *is the output aka label of the* i*-th training instance. In this paradigm we are usually interested in learning a function* $f : X \to Y$*.*

This paradigm comes mainly under two flavors: *regression and classification*, which are schematically depicted in Figure 2.1. In regression the interest is in predicting a continuous value given an input, i.e. $y \in \mathbb{R}$, such as a molecular property given a mathematical representation of a molecule. In classification, the interest is to predict in which of k classes an input belongs to, i.e. $y \in \{1, \ldots, k\}$, such as predicting the breed of a dog image given the raw pixel values of the image. The term "supervised" is coined due to the "human supervision" the algorithm receives during its training phase, through the presence of the correct answer (the label) in the experience. In a sense, in this paradigm we "teach" the learning algorithm aka *learner*. It should be emphasized that the label is not constrained to be single-valued, but can also be multi-valued. In this case, one talks about *multi-label regression or classification* (Read et al. 2009).

A more exotic form of supervised learning is *conditional generative modelling*, where the interest is in estimating $p(\boldsymbol{x} \mid \boldsymbol{y})$. For example, one may want to build a model that generates images of a specific category or a *model that designs molecules/materials with tailored properties* (K. Kim et al. 2018; Yao et al. 2021; Gebauer et al. 2022). This is one approach of how ML can tackle the *inverse design problem* in chemistry.

Definition 2.4 (Reinforcement learning). *The experience comes from the interaction of the learner, called agent in this context, with its environment. In other words, there is a feedback loop between the learner and its environment. In this paradigm we are interested in building an agent that can take suitable actions in a given situation.*

The agent observes its *environment*, selects and performs *actions* and gets *rewards or penalties* in return. The goal is to learn an optimal strategy, called a *policy*, that *maximizes the long-term reward*



(A) Classification. (B) Regression.

FIGURE 2.1: Main tasks of supervised learning.

(Géron 2017). A policy simply defines the action that the agent should choose in a given situation. In contrast to supervised learning, where the correct answers are provided to the learner, *in reinforcement learning the learner must find the optimal answers by trial and error* (Bishop 2007). Reinforcement learning techniques find application in fields such as gaming (AlphaGo is a well known example), robotics, autonomous driving and recently chemistry (H. Li et al. 2018; Gow et al. 2022). Since in the present thesis only supervised learning techniques were employed, the remaining of this chapter is devoted to this learning paradigm.

2.1.2 Formulating the problem of supervised learning

The general setting of supervised learning is as follows: we assume that there is some relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{2.1}$$

and we want to estimate f from the data. The function f represents the *systematic information* that $\boldsymbol{x}$ gives about $\boldsymbol{y}$ while $\boldsymbol{\epsilon}$ is a random *error term* independent of $\boldsymbol{x}$ and with zero mean. More formally, we have an input space X, an output space Y and we are interested in learning a function $\hat{h}\colon X \to Y$, called the *hypothesis*, which produces an output $\boldsymbol{y} \in Y$ given an input $\boldsymbol{x} \in X$. At our disposal we have a collection of input-output pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, forming the ***training set*** $\mathcal{D}_{\text{train}}$, with the pairs drawn i.i.d from $p(\boldsymbol{x}, \boldsymbol{y})$.

Ideally, we would like to learn a hypothesis that minimizes the ***generalization error or loss***:

$$\mathcal{L} \coloneqq \int_{X \times Y} \ell(h(\boldsymbol{x}), \boldsymbol{y}) p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{x} d\boldsymbol{y} \tag{2.2}$$

that is, the expected value of some *loss function* ℓ over all possible input-output pairs. A loss function just measures the discrepancy of the prediction $h(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ from the true value $\boldsymbol{y}$ and as such, the best hypothesis is the one that minimizes this integral. Obviously, it is impossible to evaluate the integral in Equation 2.2, since we don't have access to infinite data.

The idea is to use the *training error or loss*:

$$\mathcal{L}_{\text{train}} \coloneqq \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i \in \mathcal{D}_{\text{train}}} \ell(h(\boldsymbol{x}_i), \boldsymbol{y}_i) \tag{2.3}$$

as an approximation for the generalization loss, and *choose the hypothesis that minimizes the training loss*, a principle known as *empirical risk minimization*. In other words, to get a hypothesis aka *model* $\mathcal{M}$ from the data, we need to solve the following optimization problem:

$$\hat{h} \leftarrow \arg\min_{h \in \mathcal{H}} \mathcal{L}_{\text{train}} \tag{2.4}$$

which is achieved, by just feeding the training data into the learning algorithm $\mathcal{A}$:

$$\mathcal{M} \leftarrow \mathcal{A}(\mathcal{D}_{\text{train}}) \tag{2.5}$$

2.1.3 Components of a learning algorithm

By breaking down Equation 2.4, i.e. the optimization problem the learner needs to solve, the components of a learner can be revealed. The latter is comprised of the following three "orthogonal" components: a ***hypothesis space***, a ***loss function*** and an ***optimizer***. We now look into each of them individually and describe the contribution of each one to the solution of the optimization problem. For the ease of notation and clarity, in the remaining of this chapter we will stick to examples from simple (single-valued) regression and binary classification.

Definition 2.5 (Hypothesis space)**.** *The set of hypotheses (functions), denoted as $\mathcal{H}$, from which the learner is allowed to pick the solution of Equation 2.4.*

A simple example of a hypothesis space, is the one used in univariate *linear regression*:

$$\hat{y} = \beta_0 + \beta x \tag{2.6}$$

where $\mathcal{H}$ contains all lines (or hyperplanes in the multivariate case) defined by Equation 2.6. Of course, one can get *a more expressive* hypothesis space, by including polynomial terms, e.g.:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{2.7}$$

The more expressive the hypothesis space, the larger the ***representational capacity*** of the learning algorithm. For a formal definition of representational capacity, the interested reader can look at *Vapnik-Chervonenkis Dimension* (Hastie et al. n.d.).

Definition 2.6 (Loss function)**.** *A function that maps a prediction into a real number, which intuitively represents the quality of a candidate hypothesis.*

For example, a typical loss function used in regression is the *squared loss*:

$$\ell(\hat{y}, y) \coloneqq (\hat{y} - y)^2 \tag{2.8}$$

where $y, \hat{y} \in \mathbb{R}$. A typical loss function for binary classification is the *binary cross entropy loss*:

$$\ell(\hat{y}, y) \coloneqq y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}) \tag{2.9}$$

where $y \in \{0, 1\}$, indicating the correct class, and $\hat{y} \in [0, 1]$ which corresponds to the predicted probability for class-1. Notice that in both cases the loss is minimum when the prediction is equal to the ground truth. For the cross entropy loss, if $y = 1$ is the correct class, then the model must predict $\hat{y} = 1$ for the loss to be minimized.

Usually, we are not only penalizing a hypothesis for its mispredictions, but also for its *complexity*. This is done in purpose, since a learner with a *very rich hypothesis space* can easily memorize the training set but fail to generalize well to new unseen examples. *Every modification that is made to a learner in order to reduce its generalization loss but not its training loss, is called* ***regularization*** (Goodfellow et al. 2016).

A common—but not the only—way to achieve that, is by including another penalty term called *regularization term or regularizer*, denoted as $\mathcal{R}$, in Equation 2.3:

$$\mathcal{L}_{\text{train}} := \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i \in \mathcal{D}_{\text{train}}} \ell(h(\boldsymbol{x}_i), \boldsymbol{y}_i) + \lambda \mathcal{R} \tag{2.10}$$

The λ factor controls the strength of regularization and it is an ***hyperparameter***, i.e. a parameter that is not learned during training but *whose value is used to control the training phase.* In order to see how λ penalizes model complexity, assume we perform univariate polynomial regression of degree k:

$$\hat{y} = \beta_0 + \sum_{i=1}^{k} \beta_i x^i \tag{2.11}$$

combining mean squared loss (MSL) and *Lasso regularization* as training loss:

$$\mathcal{L}_{\text{train}} = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i \in \mathcal{D}_{\text{train}}} \ell(\hat{y}_i - y_i)^2 + \lambda \sum_{i=1}^{k} |\beta_i| \tag{2.12}$$

Lets apply a very strong regularization by setting $\lambda \to \infty$ (in practice we set λ to a very large value) and observe what happens to the *weights* β_i. By setting $\lambda \to \infty$, the regularization term dominates the MSL and as such, the only way to minimize the training loss is by setting $\beta_i = 0$. This leave us with a very simple model—only the *bias* β_0 survives—which always predicts the mean value of y in the training set.

Applying a regularizer, is also useful when we need to select between two (or more) competing hypotheses that are equally good. For example, assuming two hypotheses achieve the same (unregularized) training loss, the inclusion of a regularization term help us decide between the two, *by favoring the simplest one*. This is reminiscent of the *Occam's razor aka principle of parsimony*, which advocates that between two competing theories with equal explanatory power, one should prefer the one with the fewest assumptions.

Definition 2.7 (Optimizer). *An algorithm that searches through $\mathcal{H}$ for the solution of Equation 2.4.*

Having defined the set of candidate models (the hypothesis space) and a measure that quantifies the quality of a given model (the loss function), all that is remaining is a tool to scan the hypothesis space and pick the model that minimizes the training loss (the optimizer). A naive approach is to check all hypotheses in $\mathcal{H}$ and then pick the one that achieves the lowest training loss. This approach can

work if $\mathcal{H}$ is finite, but obviously doesn't scale in the general case where $\mathcal{H}$ is infinite[1]. More efficient approaches are needed if we are aiming to solve Equation 2.4 in finite time.

One optimizer that is frequently used in ML and is the precursor of more refined ones, is ***gradient descent***. With this method, the exploration of hypothesis space[2] involves the following steps:

Algorithm 1: Gradient descent

```
θ ← random initialization;
while stopping criterion not met do
    θ ← θ − η∇_θ L_train(θ);
end
```

where η is a small number called the *learning rate*. Gradient descent is based on the idea that if a multivariate function is defined and differentiable at a point $\boldsymbol{x}$, then $f(\boldsymbol{x})$ *decreases fastest if one takes a small step from* $\boldsymbol{x}$ *in the direction of negative gradient at* $\boldsymbol{x}$, $-\nabla f(\boldsymbol{x})$.

The motivation becomes clear if we look at the differential of $f(\boldsymbol{x})$ in direction $\boldsymbol{u}$:

$$f(\boldsymbol{x} + \delta\boldsymbol{u}) - f(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) \cdot \delta\boldsymbol{u} \tag{2.13}$$

Equation 2.13 says that this differential is minimized[3] when $\delta\boldsymbol{u}$ is anti-parallel to $\nabla f(\boldsymbol{x})$ and that is why we subtract the gradient in Algorithm 1, i.e. move in direction anti-parallel to the gradient. The fact that Equation 2.13 holds only locally (magnitude of $\delta\boldsymbol{u}$ must be small) explains why η must be a small number. It should be added that gradient descent can be trapped to (potential) local minima of the training loss and therefore fail to solve Equation 2.4. As it will be discussed later, this is not a problem, because *the ultimate purpose is to find a hypothesis that generalizes well, not necessarily the one that minimizes the training loss*[4]. Optimizers are discussed in further detail in the section 2.2.3.

Before moving on, it is worth to add that both the regularization and the optimizer can effect the "true" or ***effective capacity*** (Goodfellow et al. 2016) of the learner, *which might be less than the representational capacity of the hypothesis space*. For example a regularizer penalizes the complexity of an hypothesis, effectively "shrinking" the representational capacity of the hypothesis space. The effect of the optimizer can be understood by looking on its contribution to the solution of Equation 2.4. As described previously, the optimizer searches through the hypothesis space. If this "journey" is not long enough, then this "journey" is practically equivalent to a long "journey" in a shortened version of the original hypothesis space. In the rest of this chapter, by the term ***complexity*** *or capacity of a learner, we mean its effective capacity, which is affected by all its three components.*

2.1.4 Performance, complexity and experience

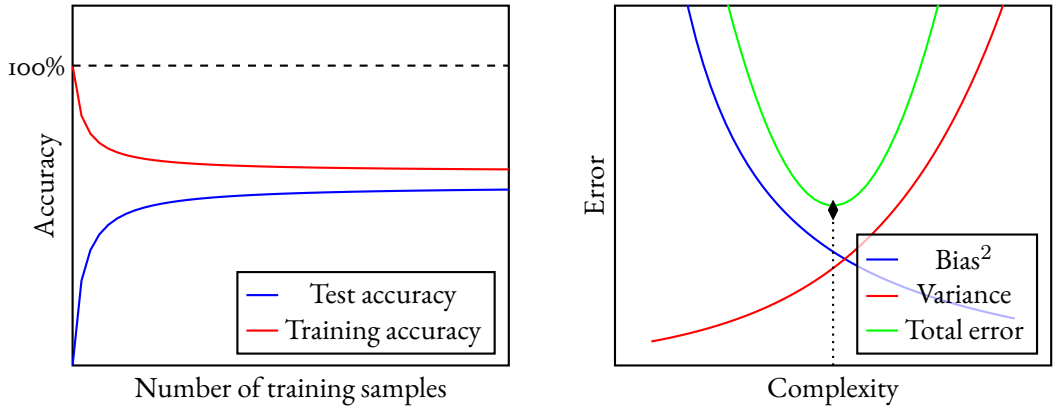
Suppose that we have trained our learner, and finally we get our model, as stated by Equation 2.5. *How can we assess its performance?* Remember, we can't calculate the generalization loss, since we are

[1] It is not uncommon for $\mathcal{H}$ to be infinite. Even for simple learners like linear regression $\mathcal{H}$ is infinite, since there infinite lines defined by Equation 2.6.

[2] We have implicitly assumed that $\mathcal{H}$ can be parameterized, i.e. $\mathcal{H} = \{h(\boldsymbol{x}; \boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}$, where Θ denotes the parameter space, the set of all values the parameter $\boldsymbol{\theta}$ can take. This allows us to write the training loss as function of model parameters and optimize them with gradient descent.

[3] The right hand side of Equation 2.13 is a dot product.

[4] Remember we use the training loss (see Equation 2.3) as a proxy for the generalization loss (see Equation 2.2).



(A) Learning curve of learner with low complexity. **(B)** Learning curve of learner with high complexity.

FIGURE 2.2: Relation between performance and experience.

not given an infinite amount of data. First of all, *we should not report the training loss, because it is optimistically biased*, as it is evaluated on the same data that has been trained on[5]. What we should is to collect new input-output pairs, forming the ***test set*** $\mathcal{D}_{\text{test}}$, and then *estimate the generalization loss* as following:

$$\mathcal{L}_{\text{test}} := \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i \in \mathcal{D}_{\text{test}}} \ell(h(\boldsymbol{x}_i), \boldsymbol{y}_i) \tag{2.14}$$

The *test error or loss* is evaluated on new—unseen to the learner during the training phase—samples and as such, it is an *unbiased estimate* of the generalization loss. Usually, since many times is not even possible to collect new samples, *we split the initial dataset into training and test sets*.

The general recipe for building and evaluating the performance of a ML model have already been presented. What is missing is how we can improve its performance, or to put it differently, the factors that affect the quality of the returned model. There are two main factors that determine the performance of the model: *complexity and experience*. In general, the larger the experience—the training set— the better the performance, just like we humans perform improve on a task by keep practicing. With regards to the complexity, *learners of low complexity might fail to capture the patterns in the data*, meaning that the resulting model will fail to generalize. In contrast, *learners of higher complexity would be able to capture these patterns*, and as such return models of higher quality. However, *as the complexity of the learner keeps increasing, the latter is more sensitive to noise, i.e. there is a higher chance that the learner will simply memorize its experience* and as such, fail to generalize.

In other words, there is a trade-off between the complexity of the learner and its performance. The learner should be not too simple but also not too complex, in order to generalize well. This in turn implies that we need to find a way to "tune" the complexity. A common way to achieve that is by using another set of instances, called ***validation set***

THEOREM 1 (Bias-variance decomposition). *From Equation 2.1 and assuming* $\epsilon \sim \mathcal{N}(0, 1)$*, the ex-*

[5]Intuitively, this is like assessing students' performance based on problems they have already seen before. They can easily achieve zero error, just by recalling their memory.

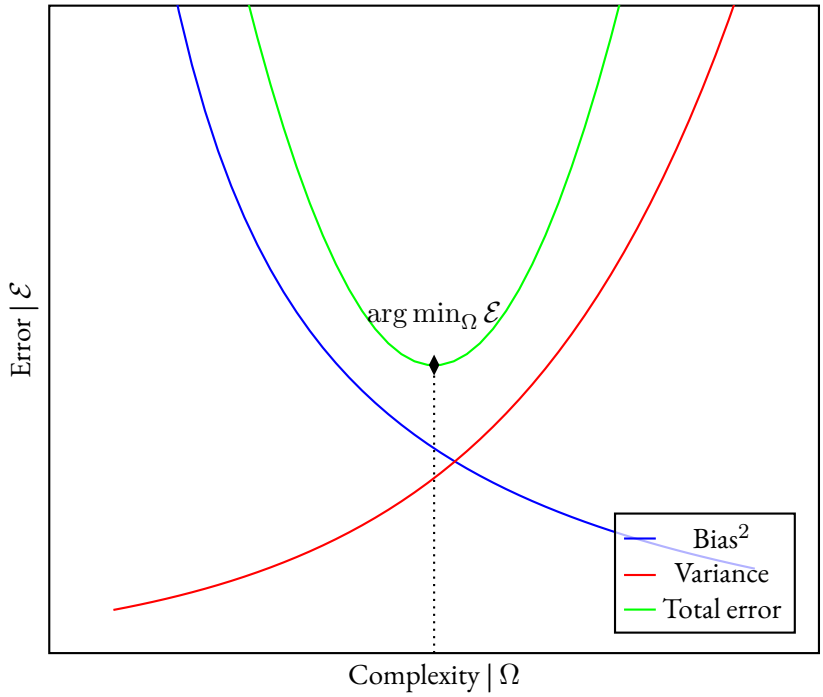


Figure 2.3: The bias-variance trade-off.

pected squared loss at $\boldsymbol{x}^*$, can be decomposed as following:

$$\mathbb{E}\left[\left(y^* - \hat{f}(\boldsymbol{x}^*)\right)^2\right] = \left(f(\boldsymbol{x}^*) - \mathbb{E}\left[\hat{f}(\boldsymbol{x}^*)\right]\right)^2 + \mathbb{E}\left[\left(\hat{f}(\boldsymbol{x}^*) - \mathbb{E}\left[\hat{f}(\boldsymbol{x}^*)\right]\right)^2\right] + \sigma_\epsilon^2 \quad (2.15)$$

The expected squared loss refers to the average squared loss we would obtain by repeatedly estimating f using different training sets, each tested at $\boldsymbol{x}^*$. The overall expected squared loss can be computed by averaging the left hand side of Equation 2.15 over all possible values $\boldsymbol{x}^*$ in the test set.

2.2 Fundamentals of Deep Learning

2.2.1 Convolutional neural networks

2.2.2 Regularizing neural networks

2.2.3 Training neural networks

Algorithm 2: Batch, mini-batch and stochastic gradient descent algorithms

Input: $\mathcal{D}_{\text{train}}$, loss function ℓ, model parameters $\boldsymbol{\theta}$, learning rate η, batch size $|\mathcal{B}|$

$\boldsymbol{\theta} \leftarrow$ random initialization;
while *stopping criterion not met* **do**
  $\mathcal{B} \leftarrow$ sample $|\mathcal{B}|$ datapoints from $\mathcal{D}_{\text{train}}$;
  $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \ell_i(\boldsymbol{\theta})$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$;
end
return *optimized parameters* $\boldsymbol{\theta}$

Algorithm 3: Momentum algorithm, needs citation

Input: Model parameters $\boldsymbol{\theta}$, momentum β, learning rate η

$\boldsymbol{\theta} \leftarrow$ random initialization;
while *stopping criterion not met* **do**
  $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m}$;
end
return *optimized parameters* $\boldsymbol{\theta}$

Algorithm 4: Nesterov momentum algorithm, needs citation

Input: Model parameters $\boldsymbol{\theta}$, momentum β, learning rate η

$\boldsymbol{\theta} \leftarrow$ random initialization;
while *stopping criterion not met* **do**
  $\boldsymbol{m} \leftarrow \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta} + \beta \boldsymbol{m})$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m}$;
end
return *optimized parameters* $\boldsymbol{\theta}$

Algorithm 5: AdaGrad algorithm, needs citation

Input: Model parameters $\boldsymbol{\theta}$, momentum β, learning rate η, smoothing term ϵ

$\boldsymbol{\theta} \leftarrow$ random initialization;
while *stopping criterion not met* **do**
  $\boldsymbol{s} \leftarrow \boldsymbol{s} + \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \epsilon}$;
end
return *optimized parameters* $\boldsymbol{\theta}$

Algorithm 6: RMSProp algorithm, needs citation

Input: Model parameters $\boldsymbol{\theta}$, decay rate β, learning rate η, smoothing term ϵ

$\boldsymbol{\theta} \leftarrow$ random initialization;
while *stopping criterion not met* **do**
  $\boldsymbol{s} \leftarrow \boldsymbol{s} + (1-\beta)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \epsilon}$;
end
return *optimized parameters* $\boldsymbol{\theta}$

Algorithm 7: Adam algorithm, needs citation

Input: Model parameters $\boldsymbol{\theta}$, learning rate η, smoothing term ϵ, momentum decay β_1, scaling decay β_2

$\boldsymbol{\theta} \leftarrow$ random initialization;
while *stopping criterion not met* **do**
  $\boldsymbol{m} \leftarrow \beta_1\boldsymbol{m} - (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$;
  $\boldsymbol{s} \leftarrow \beta_2\boldsymbol{s} + (1-\beta_2)\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$;
  $\widehat{\boldsymbol{m}} \leftarrow \frac{\boldsymbol{m}}{1-\beta_1^t}$;
  $\widehat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\beta_2^t}$;
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\widehat{\boldsymbol{m}} \oslash \sqrt{\widehat{\boldsymbol{s}} + \epsilon}$;
end
return *optimized parameters* $\boldsymbol{\theta}$

Algorithm 8: Backpropagation algorithm, needs citation

Input: Computational graph $\mathcal{G}$ where nodes u_i follow a topological ordering[a]

`/* Forward pass */`
for $i = 1$ **to** N **do**
  Compute u_i as a function of $\mathrm{Pa}_{\mathcal{G}}(u_i)$;
end
$u_N = 1$;
`/* Backward pass */`
for $i = N-1$ **to** 1 **do**
  $\bar{u}_i = \sum_{j \in \mathrm{Ch}_{\mathcal{G}}(u_i)} \bar{u}_j \frac{\partial u_j}{\partial u_i}$;
end
return *derivatives* $\bar{u}_i$

[a] Any ordering such that parents come before children.

Chapter 3

Methodology

NEURAL NETWORKS are notorious for being "data hungry", requiring a relatively large amount of training data, in order to unleash their full potential. As such, to get a representative picture of the capabilities of the proposed DL framework, two large datasets are employed to train the 3D CNN. The first one, is a subset of the University of Ottawa (UO) database (Boyd et al. 2019), and is used to verify the applicability of the proposed pipeline, examining CO_2 uptake. The second one, is the COFs database generated by (Mercado et al. 2018), and is employed to demonstrate the transferability of the approach, examining CH_4 uptake. Please note, that these datasets are already labeled, and as such no molecular simulations were performed in this study to generate the labels (gas uptakes) of the materials. Information regarding the Grand Canonical Monte Carlo (GCMC) calculations that were performed to produced the labels, can be found in the original works.

3.1 Datasets

3.1.1 MOFs dataset

The UO database is composed of 324 426 hypothetical MOFs. Randomly selected subsets of size 32 432, 5000 and 27 438, served as the training, validation[1] and test sets, respectively. The absolute CO_2 uptake at 298 K and 0.15 bar was examined and the following eight textual properties were used as input for the conventional models: unit cell's mass and volume, gravimetric surface area, void fraction, void volume, largest free sphere diameter, largest included sphere along free sphere path diameter and largest included sphere diameter. For producing the learning curves shown in Figure 4.3a, the training set size was varied and the following training set sizes were considered:

$$\{100, 500, 1000, 2000, 5000, 10\,000, 15\,000, 20\,000, 32\,432\} \tag{3.1}$$

The energy voxels of MOFs are publicly available in figshare.

3.1.2 COFs dataset

The COFs database contains 69 839 and provides data for five textual properties and CH_4 uptake at different thermodynamic conditions. A randomly selected subset of 55 871 materials served as the training set, whereas the remaining 13 698 correspond to the test set. In this work, CH_4 uptake at

[1] The validation set was used to select the number of epochs, see Section 3.3.2.

298 K and 5.8 bar was examined. The following five textual properties were used to build the conventional models: density, gravimetric surface area, void fraction, pore limiting diameter and largest cavity diameter. For producing the learning curves shown in Figure 4.3b, the training set size was varied and the following training set sizes were considered:

$$\{5000, 10\,000, 15\,000, 20\,000, 35\,000, 55\,871\} \tag{3.2}$$

3.2 Voxelized PES

In order to calculate the voxelized PES, first a 3D grid of size $n \times n \times n$ is overlayed over the unit cell of the material. Second, at each voxel centered at grid point $\boldsymbol{r}_i$, the interaction of the guest molecule with the framework atoms $\mathcal{V}(\boldsymbol{r}_i)$ is calculated, and this energy value "colorizes" the corresponding voxel. The workflow to construct the voxelized PES is schematically depicted in Figure 3.1. The grid size n and the type of the potential control the "trade-off" between information content and computational cost. The greater the grid size n, the greater the resolution of the energy image and as such, the information content. However, this comes at the cost of increased computational cost which is by no means negligible, since voxelization scales up as $\mathcal{O}(n^3)$. Similarly, the more accurate the modeling of interactions, the greater the information content, but again, extra computational burden is required. *The voxelized PES converges to the exact one as $n \to \infty$ and when the voxels are filled with energy values derived from ab-initio calculations.*

In this work we strived for minimal computational cost, setting $n = 25$ and modeling all interactions with the Lennard-Jones (LJ) potential, using a spherical probe molecule as guest. The interaction energy $\mathcal{V}(\boldsymbol{r}_i)$ between the spherical probe and the framework atoms was calculated as following:

$$\mathcal{V}(\boldsymbol{r}_i) = \sum_{\substack{j=1 \\ r_{ij} \leq r_c}}^{N} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \tag{3.3}$$

where N is the number of framework atoms, r_c is the cutoff radius which was set to 10 Å, r_{ij} is the distance between the j-th framework atom and the probe molecule and ϵ_{ij} and σ_{ij} combine the ϵ and σ values of the probe molecule and the j-th framework atom using the Lorentz-Berthelot mixing rules:

$$\sigma_{ij} = \frac{\sigma_i + \sigma_j}{2} \quad \wedge \quad \epsilon_{ij} = \sqrt{\epsilon_i + \epsilon_j} \tag{3.4}$$

If there is geometric overlap between a grid point and the position of a framework atom, the interaction energy can be extremely repulsive, leading to very large, even infinite values, which can hamper or not allow the training of a NN at all. For this reason, each voxel was filled with $e^{-\beta \mathcal{V}(\boldsymbol{r}_i)}$, which tends to 0 as $\mathcal{V}(\boldsymbol{r}_i) \to \infty$, where $\beta = \frac{1}{k_B T}$ is the Boltzmann constant and T is the temperature, which was set at 298 K. The Python package MOX$\epsilon\lambda$ was introduced to facilitate the calculation of energy voxels. In the remaining of this thesis, the terms "voxelized PES" and "energy voxels", are used interchangeably.

3.3 Machine Learning Details

For the conventional ML models, the RF algorithm as implement in the scikit-learn (Pedregosa et al. 2011) package (version 1.2.2) was used, while the PyTorch (Paszke et al. 2019) framework (version

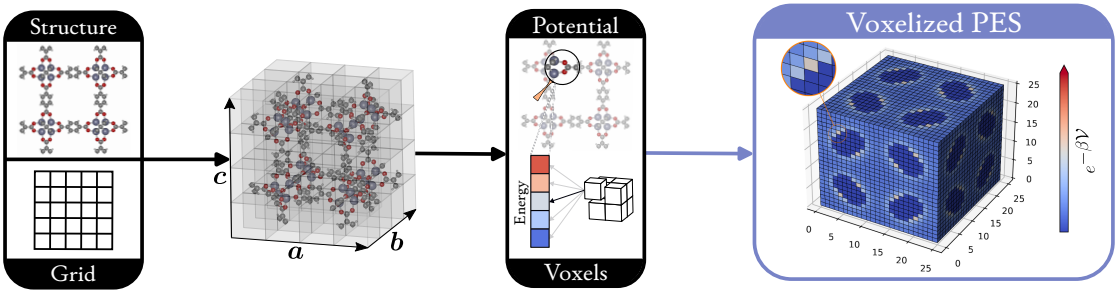


Figure 3.1: Workflow to construct the voxelized PES. The grid size and the type of the potential control the "trade-off" between information content and computational cost. The IRMOF-1 structure was visualized with the iRASPA software (Dubbeldam et al. 2018).

2.0.1+cu118) was employed for the CNN models. The performance, i.e. the generalization ability of the models, was assesed by the coefficient of determination R^2:

$$R^2 := 1 - \frac{\sum_{i=1}^{N_{\text{test}}} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N_{\text{test}}} (y_i - \bar{y})^2} \tag{3.5}$$

where N_{test} is the number of samples in test set, $\bar{y}$ is the mean value of y in the test set and y_i, $\hat{y}_i$ are the ground truth and predicted values of the i-the sample, respectively. In all cases where confidence interval (CI) are presented, they were calculated using the percentile bootstrap method (Efron et al. 1994), with 10 000 bootstrapped samples from the test set.

3.3.1 CNN architecture

The architecture of the 3D CNN is presented in Figure 4.1, whereas a PyTorch implementation is publicly available in: RetNet. Kernel size is set to 3 for Conv1, Conv2 and 2 for Conv3, Conv4 and Conv5 layers. Stride equals 1 for all Conv layers and only Conv1 layer is padded, with "same" padding and "periodic" mode. For both MaxPool layers, kernel size and stride are both set to 2. For the Dropout layer, the dropout rate p equals 0.3, while the negative slope is set to 0.01 for all LeakyReLU layers.

3.3.2 Preprocessing & CNN training details

Prior to entering the CNN the energy voxels are standardized "on the fly" based on the training set statistics—this transformation is applied both during training and inference—which are computed channel wise[2]. The voxelized PES of a material $\boldsymbol{x}$, enters the CNN as following:

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - \mu_{\text{train}}}{\sigma_{\text{train}}} \tag{3.6}$$

Regarding CNN training, MSL is used as loss function and weights are initialized according to the He scheme (He et al. 2015). The Adam optimizer (Kingma et al. 2017) is employed, with $|\mathcal{B}| = 64$, $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$. The CNN training lasts for 50 epochs with

[2] The voxelized PES is essentialy a single channel, i.e. grayscale, image.

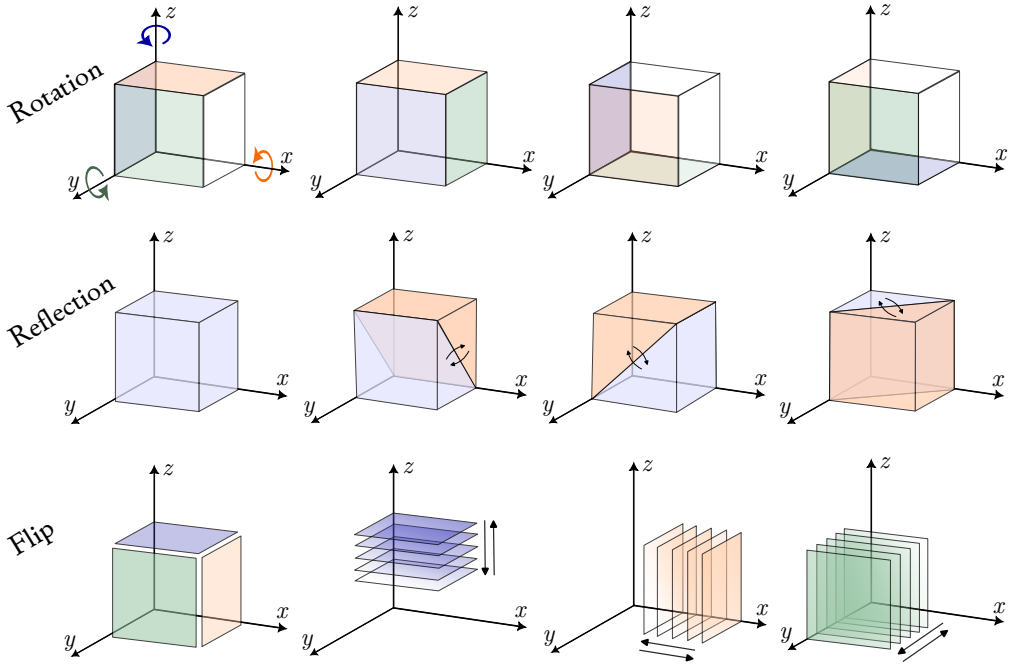


Figure 3.2: Geometric transformations for data augmentation.

the learning rate being decaybed by 0.5 every 10 epochs. RetNet was trained in the MOFs dataset with the largest training set size (32 432 training samples), for a different number of epochs, namely 10, 20 and 50. The latter value was selected, since it showed the greatest performance in the validation set.

3.3.3 Data augmentation

With this technique, the training set is artifically increased, by applying transformations on the input that leave the label unchanged. With regards to gas adsorption, this amounts to applying geometric transformations on the voxelized PES, that leave the gas uptake value of the material unchanged. Data augmentation, helps the CNN to combat overfitting—e.g. memorizing specific orientations of the voxelized PES—and focus on the underlying patterns.

In this work, four types of geometric transformations are applied (including the identity one), as shown in Figure 3.2. At each training iteration, the samples in the batch undergo one of these transformations, with all transformations having the same probability to be applied. For instance, at one training iteration, the voxelized PES can be rotated 90° around the x-axis, while at another iteration, it might be flipped along the z-axis. Rotation is performed either clockwise or counterclockwise, around one the three axes. The voxelized PES can also be viewed as a stack of 2D slices. In this view, reflection corresponds to transposing each slice, whereas flip reversed the order of the slices. Reflection takes place along one of the xy, xz, yz planes, whereas flip is performed along one of the three axes. Figure 3.3 illustrates the performance difference when the CNN is trained on the MOFs dataset with and without data augmentation, for training set sizes:

$$\{5000, 10\,000, 15\,000, 20\,000, 32\,432\} \tag{3.7}$$

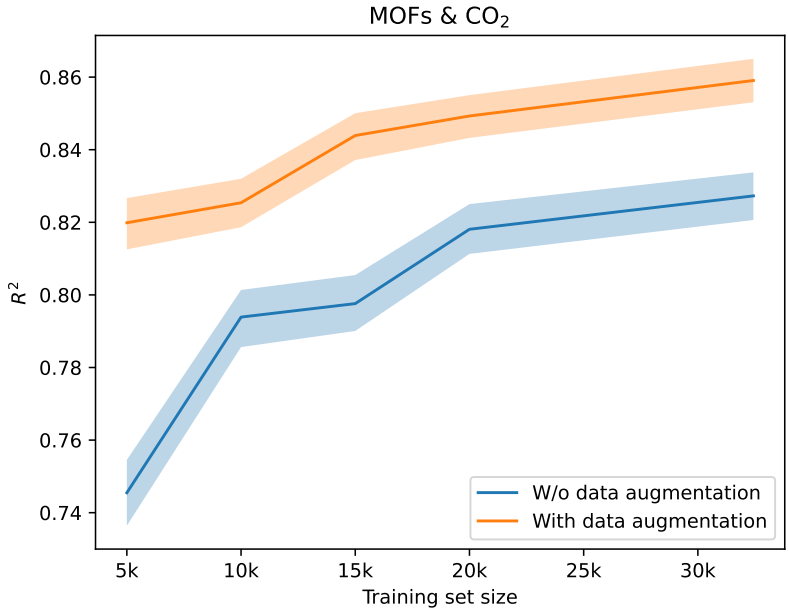


Figure 3.3: CNN performance (R^2 score) on test set with and without data augmentation. Shaded areas correspond to the 95 % CI.

Chapter 4

Results & Discussion

THE PROPOSED FRAMEWORK is tested on the UO database, for predicting CO_2 uptake in MOFs, the gas that mainly "triggered" the development of energy-based descriptors. In order to evaluate the transferability of the approach, a different host-guest system is also examined. We apply the suggested approach in the database created by Mercado et al. (2018), for predicting CH_4 uptake in COFs. In both cases, the resulting ML models are compared with conventional ones, built upon geometric descriptors. In the rest of this chapter, results from these comparisons are presented, followed by discussion for improvements of the proposed framework. Before delving into the results, we first take a look at RetNet, the 3D CNN under the hood, that takes as input a voxelized PES and outputs a prediction for a gas adsorption property, hereon gas uptake.

4.1 Visualizing RetNet

Figure 4.1 illustrates the processing a voxelized PES undergoes, as it is passing through RetNet. For the purpose of this visualization, we use the model trained on the MOFs dataset with the largest training set size (see Section 3.1). Moreover, for the ease of visualization, only some feature maps of RetNet are visualized. Please note, that each feature map of a given layer, combines all the feature maps of the precedent layer. The only exception are the pooling layers, which just downsample the feature maps from the previous layers.

For example, each feature map of the Conv2 layer takes into account all the twelve feature maps of Conv1 layer. In contrast, the feature maps of the MaxPool1 layer, are just downsampled versions of the corresponding feature maps in Conv2 layer. Although feature maps of CNNs are not meant to be interpreted by humans—especially the ones found deeper in the network—it is worth noticing that early Conv layers (i.e. Conv1 and Conv2) emphasize the texture of the structure. For instance, the third feature map of Conv1 layer delineates the skeleton of the framework.

Moving towards the output layer, the alternation of MaxPool and Conv layers continues until the Flatten layer, which just flattens out and concatenates[1] all feature maps from Conv2 layer into a single vector of size 3240. This vector is then processed by a fully connected neural network (FCNN)—i.e. the stack of Dense and Output layers—to give the final prediction. Since the Output layer is really nothing more than a linear layer, all that RetNet does is the following:

[1] Given m feature maps of size $n \times n \times n$, a Flatten layer converts them into a vector of size mn^3.

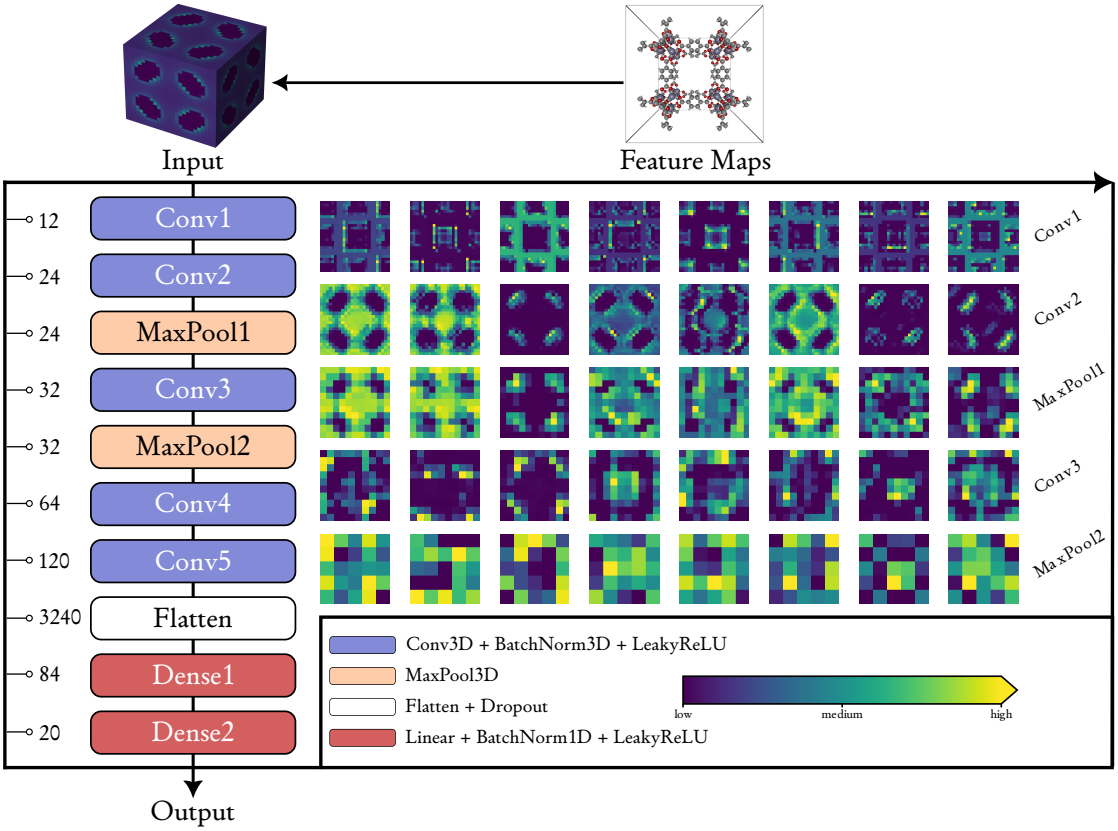
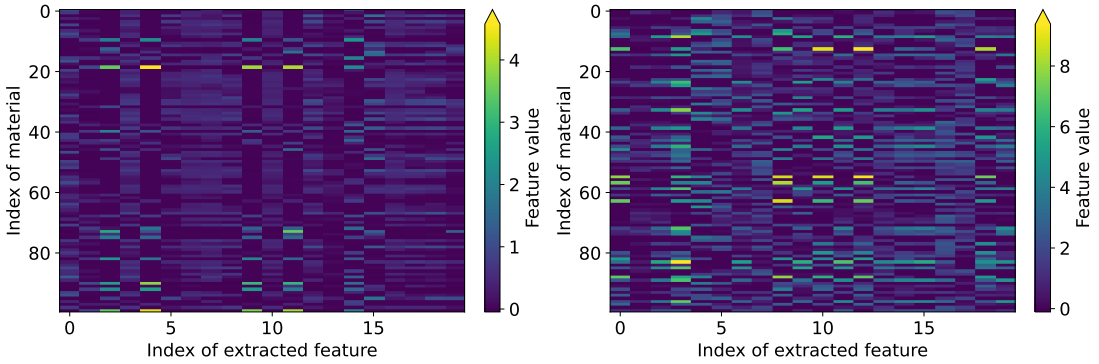


Figure 4.1: Forward pass of IRMOF-1 through RetNet. For the sake of visualization, only slices (feature maps are 3D matrices) of eight feature maps from the first five layers are visualized. For Conv1 layer, the fifth slice is presented, while for the remaining layers, the first slice is presented. The IRMOF-1 structure was visualized with the iRASPA software (Dubbeldam et al. 2018).

$$\underbrace{\boldsymbol{x}}_{\text{input}} \longrightarrow \underbrace{\phi(\boldsymbol{x};\boldsymbol{\theta})}_{\text{feature extraction}} \longrightarrow \underbrace{\boldsymbol{\beta}^\top \phi(\boldsymbol{x};\boldsymbol{\theta}) + \beta_0}_{\text{output}} \tag{4.1}$$

Equation 4.1 says that RetNet, *starting from the PES, extracts a fingerprint—that is, a high level representation of the PES—and then predicts the gas uptake by using a linear model on top of this fingerprint*. All intermediate layers between Input and Output layer participate in this feature extraction step, with the Dense2 layer determining the size of the fingerprint, which is a vector of size 20, i.e. $\phi(\boldsymbol{x}) \in \mathbb{R}^{20}$ (see Figure 4.2). The fact that *this fingerprint extraction step is learnable*—the parameters $\boldsymbol{\theta}$ of ϕ are learned during the training phase—*is what fundamentally distinguishes the proposed approach from methods that use hand-crafted fingerprints* (see Section 1.3). *In these methods the fingerprint or extraction step is fixed*, and based on some heuristic, such as energy histograms (Bucior, Bobbitt, et al. 2019) or average interactions (Fanourgakis, Gkagkas, Tylianakis, and Froudakis 2020). *Hereon, feature ex-*



(A) Fingerprints extracted from the MOFs dataset. (B) Fingerprints extracted from the COFs dataset.

Figure 4.2: Output of the last LeakyReLU layer of RetNet trained on MOFs (left) and COFs (right) datasets, with the corresponding maximum training set size. The fingerprints of the first 100 materials in the training set are depicted.

traction from the PES is no longer fixed, but is an essential part of the training phase.

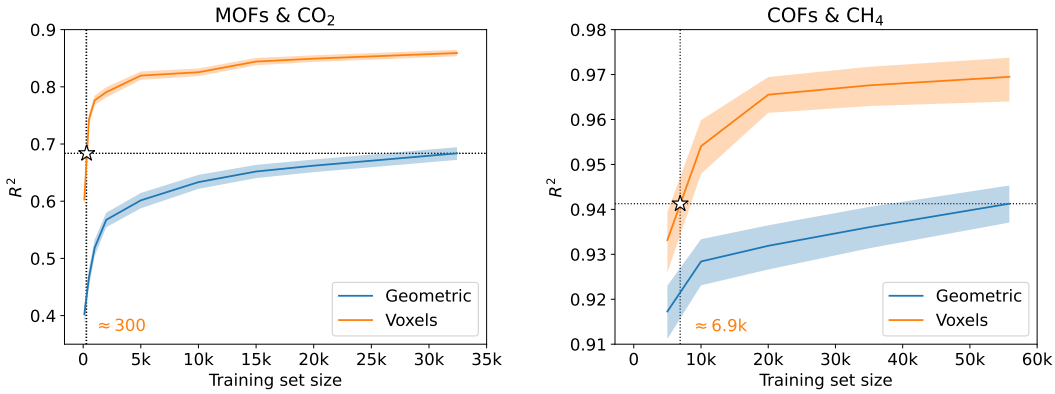
4.2 Learning Curves

The learning curves of the conventional models—built upon geometric descriptors—and the proposed ones–built upon energy voxels—are shown in Figure 4.3. As it can be seen from Figure 4.3a, in the MOF-CO_2 case, the CNN model achieves an R^2 score of 0.859, outperforming the conventional model, which shows an R^2 score of 0.690. This amounts to around 25 % increase in accuracy, even with such a coarse approximation of the PES[2]. Moreover, from the same figure, one can notice that the proposed model reaches the peak performance of the conventional one—that is, the performance when trained with the maximum training set size—by requiring two orders of magnitude less training data, around 300.

Analogous results are observed when examining the COF-CH_4 case. Again the CNN model performs better, showing an R^2 score 0.969 compared to 0.941 for the conventional one. Similar to the previous case, a substantially smaller amount of training data are required—one order of magnitude less training, around 6900—for the CNN model to match the performance of the conventional model.

The fact that in both cases, the learning curves of the proposed models lie above the corresponding ones of the conventional models, should be credited to the following factors: i). The increased informativeness of the voxelized the voxelized PES—in comparison to geometric descriptors. ii). The ability of CNNs to handle images and image-like data, such as the voxelized PES, which is essentially a single channel 3D image. iii). The data augmentation technique, which was applied during the CNN training (see Section **??**).

[2] In this work, all host-guest interactions were modeled with the LJ potential (see Section 3.2), which neglects electrostatic interactions.



(A) Learning curves for MOFs & CO_2.

(B) Learning curves for COFs & CH_4.

FIGURE 4.3: Performance (R^2 score) on test set as function of the training set size for conventional and CNN models. Shaded areas correspond to the 95 % CI. The x-coordinate of the white star denotes the trainig set size where the CNN model reaches the performance of the conventional one, the y-coordinate. "Geometric" stands for geometric descriptors, while "Voxels" stands for energy voxels.

4.3 Discussion

It is worth mentioning the increase in performance, approximately 13 %, of the CNN model in the COFs-CH_4 case ($R^2 = 0.969$) compared to the MOFs-CO_2 case ($R^2 = 0.859$). In contrast to CO_2, which exhibits strong electrostatic interactions with the framework atoms, CH_4 lacks dipole or quadrupole moment. Given that the same resolution—i.e. the same grid size—was used in both cases and that the LJ potential doesn't account for electrostatic interactions, this performance gap should be attributed to the absence of the latter in the voxelized PES. *In other words, the extra "contrast" that such strong interactions add to the energy image of the masterial, is missing from the voxelized PES. As such, a straightforward approach to improve the performance of the proposed approach, especially for adsorbates like CO_2, H_2 and H_2S, is to include this type of interactions into the voxelized PES*. Of course, there is no free lunch, since these refinements require the assignment of partial charges to each framework atom, which is a computationally expensive task. Luckily, ML-based approaches have already been developed (Bleiziffer et al. 2018; Raza et al. 2020; Kancharlapalli et al. 2021), which can assign partial charges rapidly and with high fidelity, enabling the efficient construction of a more accurate voxelized PES.

Improving the input, and as such, the performance of the suggested pipeline is a major concern, but not the only one. *What about the data efficiency of the pipeline?* Imagine that we are asked to predict CH_4 uptake at various thermodynamic conditions. A naive approach would be to collect training data and retrain from scratch the CNN for every thermodynamic condition, which is of course a laborious task. *Can we do something smarter?* Well, the fact that the proposed framework uses a DL algorithm under the hood, opens the door for applying ***transfer learning*** *techniques*. In a nutshell, transfer learning(Zhuang et al. 2019; R. Ma et al. 2020; Kang et al. 2023) is based on the following idea: *a violist can learn to play piano faster than others, since both the piano and the violin are musical instruments, and*

may share some common knowledge. Translating this to NNs, a pre-trained NN on an original task—known as the *source task*—may require less training data to perform well on a new task—known as the *target task*—if there is some *similarity between the tasks*. Coming back to our "imaginary" scenario, all we have to do is to train the CNN once in a specific thermodynamic condition[3] and then fine-tune this pre-trained model on the other conditions.

Throughout this work we focused on gas adsorption, but of course this doesn't mean we are not interested in predicting other properties of reticular materials. *What if we are asked to predict properties such as band gap or bulk modulus?* In that case, quantities such as *electron density* are more informative over host-guest interactions with regards to the aforementioned properties. This entails that the *voxelized electron density* should substitute the voxelized PES, as input to the 3D CNN. Nevertheless, ***wouldn't be great if all properties could be predicted from one and only one input?*** If our aim is to predict *different properties for the same structure, shouldn't the structure itself be used as input?* Currently, the approaches to tackle this challenge are based on *text representations* (Bucior, Rosen, et al. 2019; Cao et al. 2023) and *crystal graphs* (Xie et al. 2018; Chen et al. 2019). The main drawback of these approaches, is their inability to represent exactly the structure, that is the *exact arrangement of the atoms in the 3D space*. ***Point clouds*** (Qi et al. 2016; Bello et al. 2020) are a natural way to solve this problem, since they are just *a set of coordinates and associated features*. In our context, the coordinates are the coordinates of the atoms, and the associated features are the types of the atoms. It should be emphasized, that *a point cloud is not another mathematical representation of a material—in the sense of a descriptor—it is the material itself*[4]. Therefore, an answer to the original question is to *couple point clouds with a neural network that can handle such kind of input*. Such an approach might has to overcome the current immaturity of DL over point clouds—especially regarding materials and molecules—but from a chemical perspective, it is the only one that truly respects the 3D nature of chemistry and of course, reticular chemistry.

[3] Preferably, the one where we have more labeled training data.

[4] Same ideas apply for molecules and in general, for any chemical system.

Index

INDEX

INDEX

Acronyms

ANG adsorbed natural gas.

CI confidence interval.

CNG compressed natural gas.

CNN convolutional neural network.

COF covalent organic framework.

CoRE Computation-Ready Experimental.

CSD Cambridge Structural Database.

DAC direct air capture.

DL deep learning.

FCNN fully connected neural network.

GCMC Grand Canonical Monte Carlo.

GPU graphics processing unit.

i.i.d independent and identically distributed.

ILSVRC Large Scale Visual Recognition Challenge.

LJ Lennard-Jones.

LNG liquefied natural gas.

MAE mean absolute error.

ML machine learning.

MOF metal-organic framework.

MSL mean squared loss.

NN neural network.

Acronyms

PES potential energy surface.

RF Random Forest.

UO University of Ottawa.