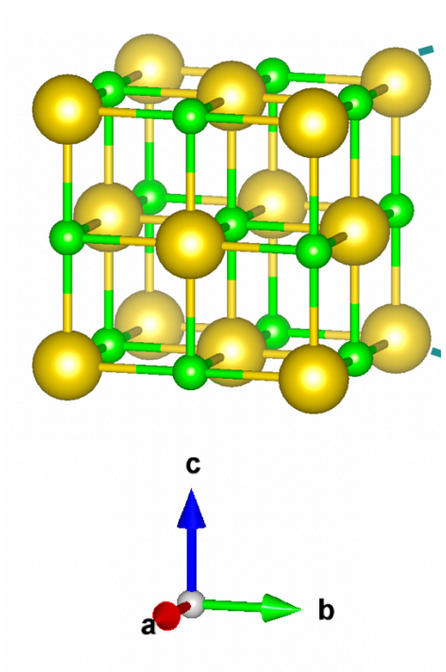
Predicting Material Properties using Invertible Real-Space Crystallographic Representation (IRCR)

Crystal Structure Representation



Any crystal structure can be represented by a repeating unit that tiles the entire 3D space. This repeating unit is called the unit cell.

The unit cell itself contains a number of atoms that are arranged inside of it.

Thus a material can be represented as

M = (A, X, L) , where

A - atomic species [a_1, a_2, …, a_n]
X - positions of each atom in the unit cell $[\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}]$
L - three lattice vectors $[\overrightarrow{l_1}, \overrightarrow{l_2}, \overrightarrow{l_3}]$, can also be expressed by $[a, b, c, \alpha, \beta, \gamma]$

The positions can be either fractional coordinates or cartesian coordinates.

We can convert between fractional coordinates X and cartesian coordinates $\tilde{X}$ as follows:

$$\tilde{X} = LX \quad \& X = L^{-1}\tilde{X}$$

As an example, if an atom has fractional coordinate $(0.25,0.25,0.5)^T$, then its cartesian coordinate will be $x = 0.25\overrightarrow{l_1} + 0.25\overrightarrow{l_2} + 0.5\overrightarrow{l_3}$.

Crystal Structure File :

1. **POSCAR**
2. **Crystallographic Information File (CIF)**

NaCl POSCAR File

```
Na4 Cl4
1.0
   5.5881264354399347    0.0000000000000000    0.0000000000000003
  -0.0000000000000003    5.5881264354399347    0.0000000000000003
   0.0000000000000000    0.0000000000000000    5.5881264354399347
Na Cl
4 4
direct
   0.0000000000000000    0.0000000000000000    0.0000000000000000 Na+
   0.0000000000000000    0.5000000000000000    0.5000000000000000 Na+
   0.5000000000000000    0.0000000000000000    0.5000000000000000 Na+
   0.5000000000000000    0.5000000000000000    0.0000000000000000 Na+
   0.0000000000000000    0.0000000000000000    0.5000000000000000 Cl-
   0.0000000000000000    0.5000000000000000    0.0000000000000000 Cl-
   0.5000000000000000    0.0000000000000000    0.0000000000000000 Cl-
   0.5000000000000000    0.5000000000000000    0.5000000000000000 Cl-
```

NaCl CIF File

```
# generated using pymatgen
data_NaCl
_symmetry_space_group_name_H-M   'P 1'
_cell_length_a   5.58812644
_cell_length_b   5.58812644
_cell_length_c   5.58812644
_cell_angle_alpha   90.00000000
_cell_angle_beta   90.00000000
_cell_angle_gamma   90.00000000
_symmetry_Int_Tables_number   1
_chemical_formula_structural   NaCl
_chemical_formula_sum   'Na4 Cl4'
_cell_volume   174.50130186
_cell_formula_units_Z   4
loop_
 _symmetry_equiv_pos_site_id
 _symmetry_equiv_pos_as_xyz
  1  'x, y, z'
loop_
 _atom_type_symbol
 _atom_type_oxidation_number
  Na+  1.0
  Cl-  -1.0
loop_
 _atom_site_type_symbol
 _atom_site_label
 _atom_site_symmetry_multiplicity
 _atom_site_fract_x
 _atom_site_fract_y
 _atom_site_fract_z
 _atom_site_occupancy
  Na+  Na0  1  0.00000000  0.00000000  0.00000000  1
  Na+  Na1  1  0.00000000  0.50000000  0.50000000  1
  Na+  Na2  1  0.50000000  0.00000000  0.50000000  1
  Na+  Na3  1  0.50000000  0.50000000  0.00000000  1
  Cl-  Cl4  1  0.00000000  0.00000000  0.50000000  1
  Cl-  Cl5  1  0.00000000  0.50000000  0.00000000  1
  Cl-  Cl6  1  0.50000000  0.00000000  0.00000000  1
  Cl-  Cl7  1  0.50000000  0.50000000  0.50000000  1
```

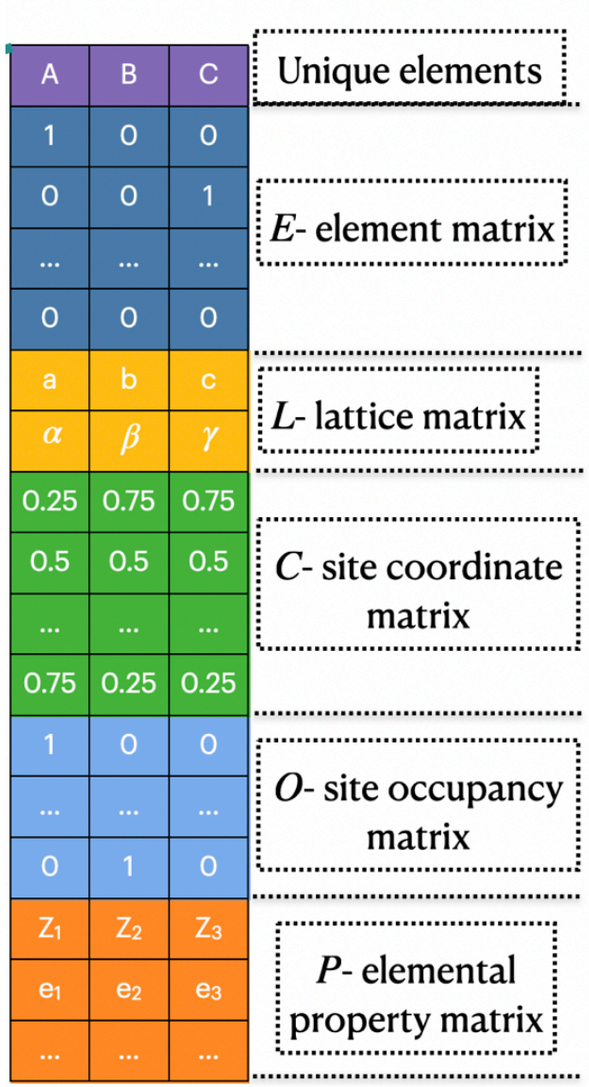
Feature Engineering : Composition-Weighted elemental properties, sine matrix elements or its eigenvalues, ACSF, SOAP etc.

But it is also possible to predict material properties with raw crystal structure data, where the neural network models will extract meaningful features from that raw data.

Invertible Real-Space Crystallographic Representation (IRCR)

Invertible : We can invert it back to CIF/ POSCAR with no information loss. Hence it is used in generative models.

Let us consider upto ternary materials.



Matrix	Shape	Description
E (Element)	**(Z_{max}×3)**	One-hot encoded vectors representing unique elements (up to ternary). Z_{max} is the highest atomic number in the database.
L (Lattice)	**(2×3)**	Encodes lattice geometry in two rows: constants (a,b,c) and angles (α,β,γ).
C (Site Coord.)	**(n_{sites}×3)**	Fractional coordinates of lattice sites. Uses zero padding for empty sites, where nsites is the max atoms in a unit cell.
O (Occupancy)	**(n_{sites}×3)**	One-hot encoded elements at specific lattice sites. Corresponds one-to-one with matrices E and O.
P (Property)	**(6×3)**	Encodes 6 chemical properties: Atomic Number (Z), Electronegativity (e), Period (p), Group (g), Fraction (s), and Valence (n_v).

$Z_{max} = 94$ (Pu), $n_{sites} \leq 20$: IRCR shape is 144x3

Follow this repo to construct IRCR from CIF/POSCAR files :
https://github.com/SouravMal/MagGen

Mal et al., MagGen (JPCL, 2024)
Ren et al., FTCP (Matter, 2022)

Task1 : Formation Energy Prediction with Neural Network

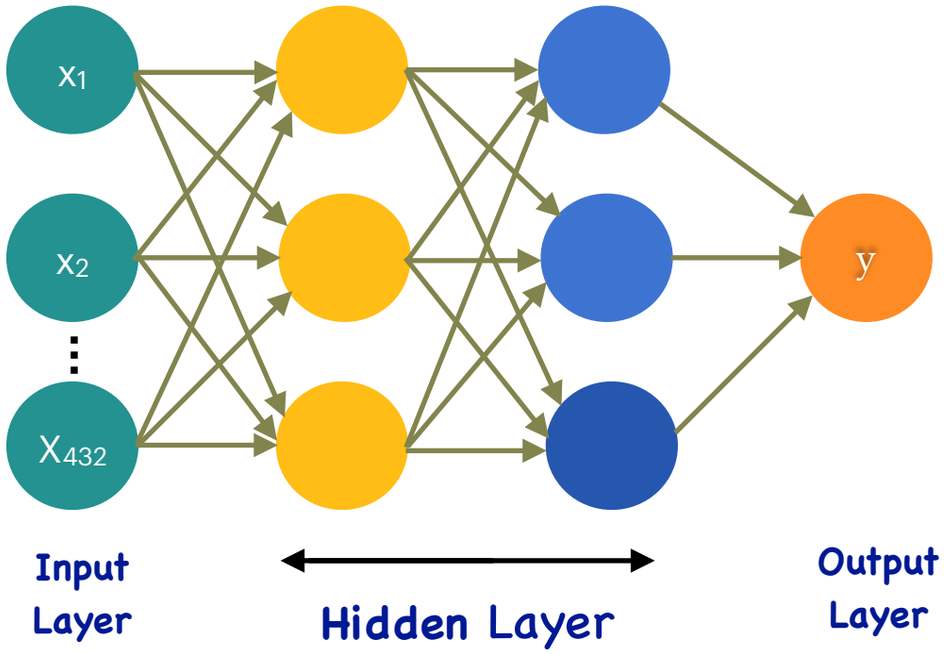
A	B	C
1	0	0
0	0	1
...	...	...
0	0	0
a	b	c
α	β	γ
0.25	0.75	0.75
0.5	0.5	0.5
...	...	...
0.75	0.25	0.25
1	0	0
...	...	...
0	1	0
Z_1	Z_2	Z_3
e_1	e_2	e_3
...	...	...

Matrix (144x3)

Flatten

Vector (432)

Multi-layer Perceptron (MLP)



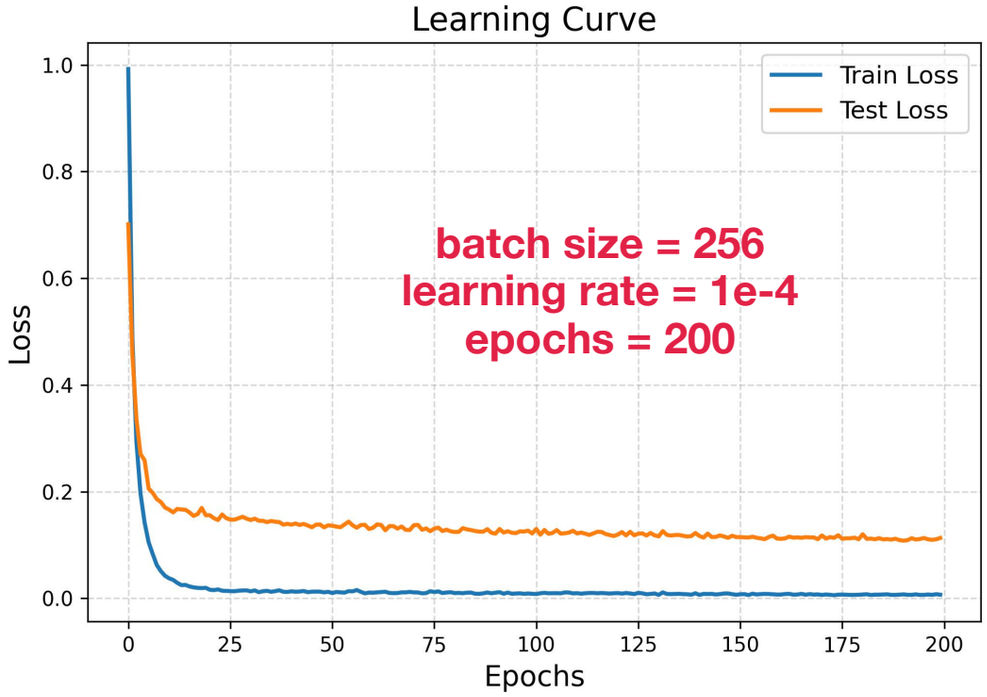
Predicted Property Value

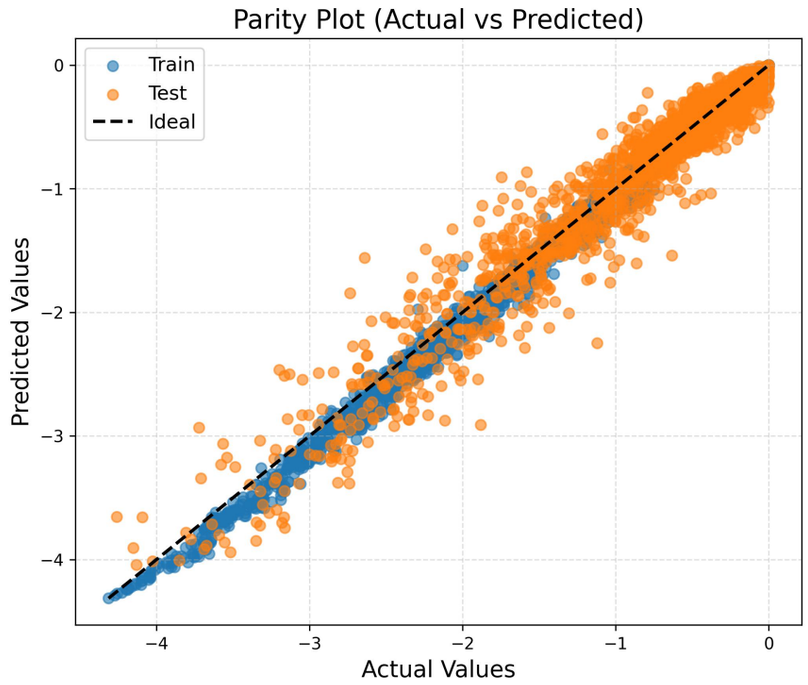
Neural Network

Training Dataset : 15130 materials, taken from Alexandria database

```python
import torch
import torch.nn as nn

class RegressorANN(nn.Module):
    def __init__(self,
                 input_size=432,
                 hidden_size_1=1024,
                 hidden_size_2=256,
                 hidden_size_3=128,
                 hidden_size_4=32):
        super(RegressorANN, self).__init__()

        self.fc1 = nn.Linear(input_size, hidden_size_1)
        self.bn1 = nn.BatchNorm1d(hidden_size_1)

        self.fc2 = nn.Linear(hidden_size_1, hidden_size_2)
        self.bn2 = nn.BatchNorm1d(hidden_size_2)

        self.fc3 = nn.Linear(hidden_size_2, hidden_size_3)
        self.bn3 = nn.BatchNorm1d(hidden_size_3)

        self.fc4 = nn.Linear(hidden_size_3, hidden_size_4)
        self.bn4 = nn.BatchNorm1d(hidden_size_4)

        self.fc5 = nn.Linear(hidden_size_4, 1)

        self.act = nn.ReLU()

    def forward(self, x):
        # Standard order: Linear -> BatchNorm -> Activation
        x = self.act(self.bn1(self.fc1(x)))
        x = self.act(self.bn2(self.fc2(x)))
        x = self.act(self.bn3(self.fc3(x)))
        x = self.act(self.bn4(self.fc4(x)))
        out = self.fc5(x)
        return out
```

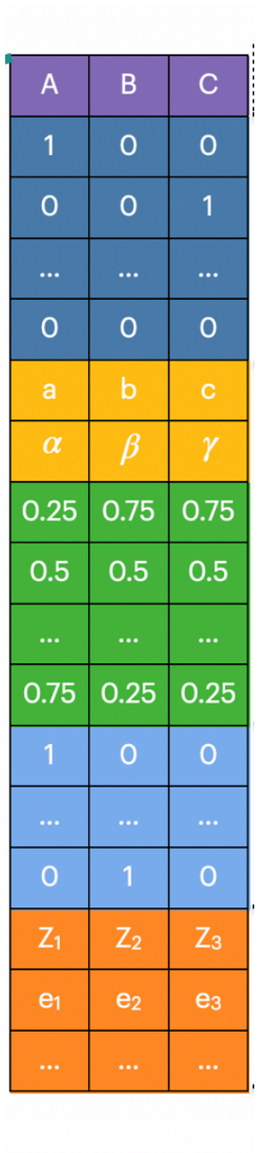
A Neural Network with four hidden layers.





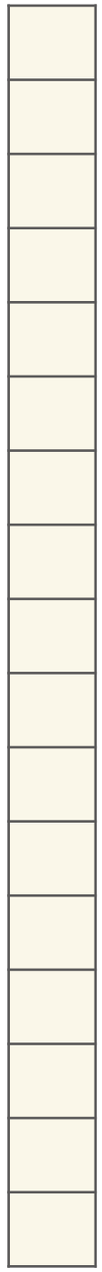
Dataset	MAE (eV/atom)	R2 Score
Training Set	0.026	0.995
Test Set	0.101	0.946

Limitation of MLP :

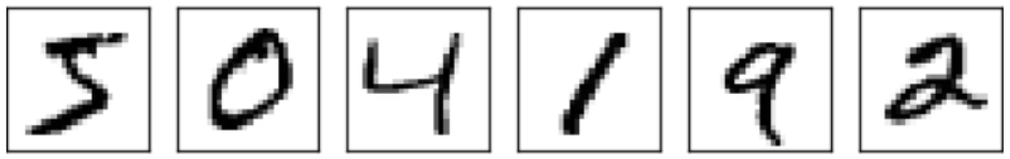


Matrix (144x3)

Flatten



Vector (432)



Flattening destroys spatial structure.

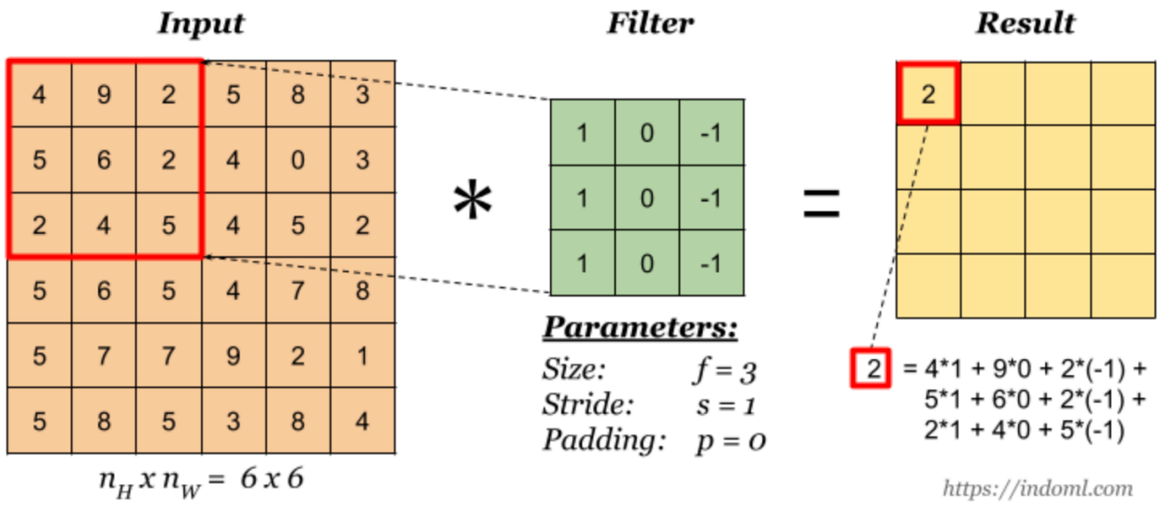
A pixel at (x,y) loses its connection to its neighbours, destroying local context like shapes or textures.

Better approach is using convolution network on top of our MLP.

Convolutional Neural Network (CNN)

CNNs automatically learn spatial structures of features from data using *convolutions.*

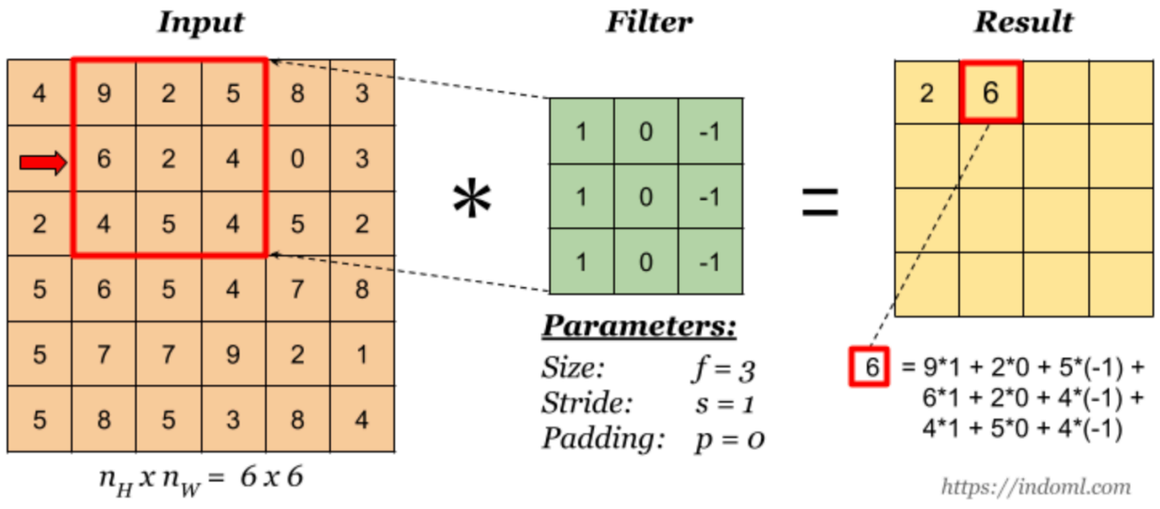
Convolution Layers : Apply learnable filters that slide over the input to detect edges, textures, shapes or any complex patterns.



1. **A 3x3 filter scans the image, and at each position, it performs element-wise multiplication with the selected patch and sums the result to produce one output value.**

2. **With stride=1, the filter moves one step at a time across the input, computing the next value in the feature map.**

3. **This example show how the first and second output values are obtained, demonstrating how convolution extracts local patterns from the image.**



After scanning the whole image , we will get a feature map.

By increasing the stride, the filter jumps more pixels at each step, which reduces the spatial size of the output. In this way, controlling the stride effectively downsamples the image.

Stride : The number of pixels by which the filter moves after each operation.

Architecture of a convolutional neural network

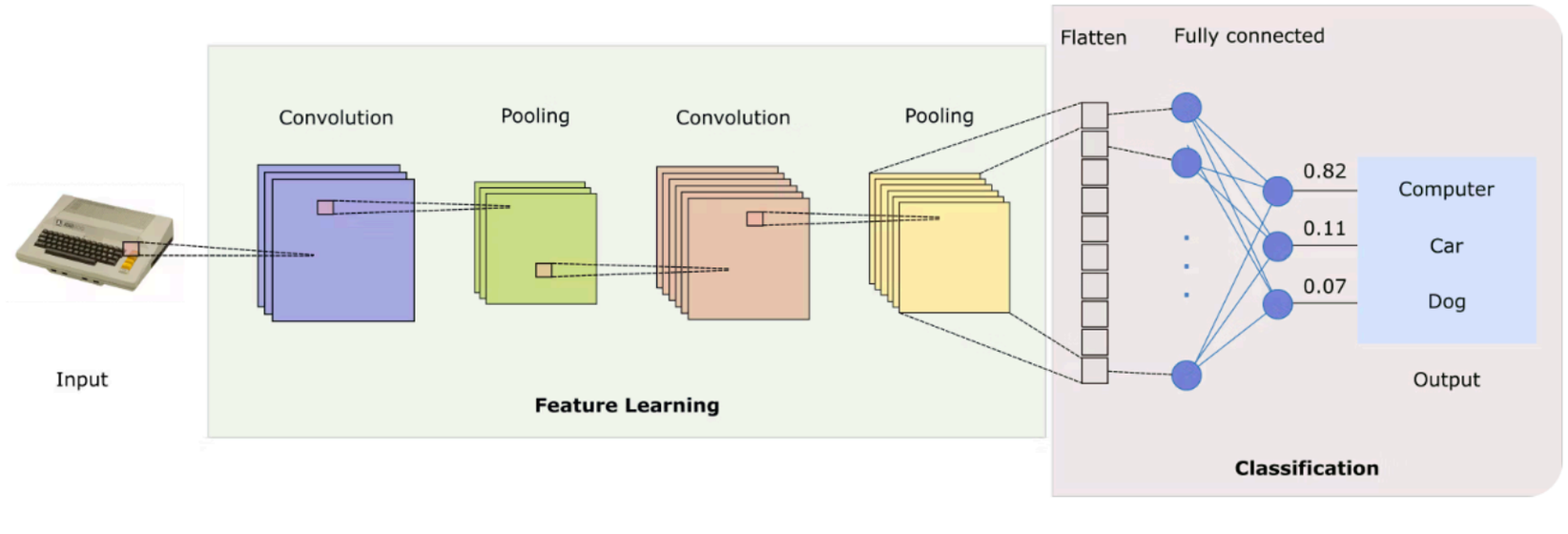


Image Source : https://domino.ai/blog/gpu-accelerated-convolutional-neural-networks-with-pytorch

Pooling reduces the spatial size of feature maps while keeping key information.

Max pooling selects the largest value in each small region (e.g., 2×2), capturing the strongest activation.

It **downsamples the feature map**, making the model more efficient and more robust to small shifts.

Task2 : Formation Energy Prediction with Convolution Neural Network

A	B	C
1	0	0
0	0	1
...	...	...
0	0	0
a	b	c
α	β	γ
0.25	0.75	0.75
0.5	0.5	0.5
...	...	...
0.75	0.25	0.25
1	0	0
...	...	...
0	1	0
Z_1	Z_2	Z_3
e_1	e_2	e_3
...	...	...

Convolution

Matrix (144x3)

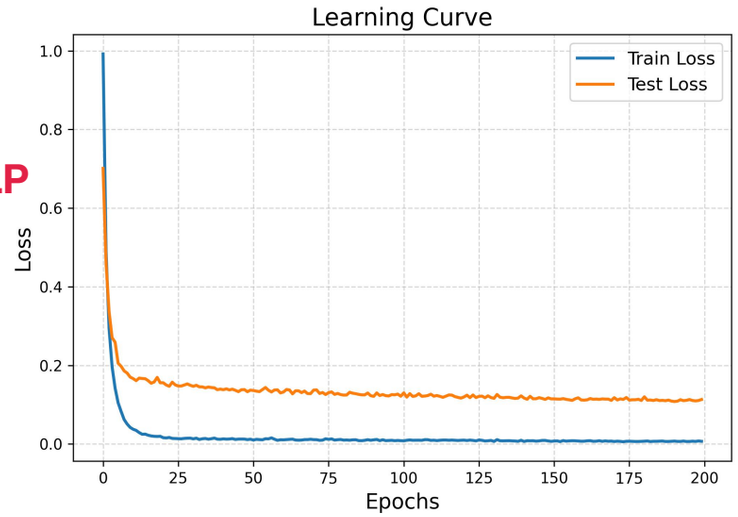
```python
class RegressorANN(nn.Module):
    def __init__(self,
                 hidden_size_1=1024,
                 hidden_size_2=256,
                 hidden_size_3=128,
                 hidden_size_4=32):
        super(RegressorANN, self).__init__()

        # convolution
        self.conv1 = conv_block(3, 64, 5, 2, 2)     72x64
        self.conv2 = conv_block(64, 128, 5, 2, 2)  36x128
        self.conv3 = conv_block(128, 256, 5, 2, 2) 18x256
        self.conv4 = conv_block(256, 256, 3, 1, 1) 18x256

        # flatten
        self.flatten = nn.Flatten()

        # fully connected layer
        self.fc1 = nn.Linear(256*18, hidden_size_1)
        self.bn1 = nn.BatchNorm1d(hidden_size_1)

        self.fc2 = nn.Linear(hidden_size_1, hidden_size_2)
        self.bn2 = nn.BatchNorm1d(hidden_size_2)

        self.fc3 = nn.Linear(hidden_size_2, hidden_size_3)
        self.bn3 = nn.BatchNorm1d(hidden_size_3)

        self.fc4 = nn.Linear(hidden_size_3, hidden_size_4)
        self.bn4 = nn.BatchNorm1d(hidden_size_4)

        self.fc5 = nn.Linear(hidden_size_4, 1)

        self.act = nn.ReLU()
```

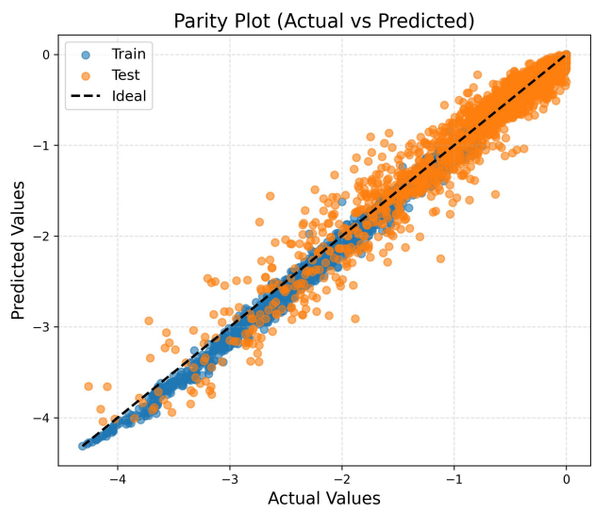
```python
class conv_block(nn.Module):
    def __init__(self, in_c, out_c, kernel_size, stride, padding):
        super(conv_block, self).__init__()

        self.conv = nn.Conv1d(in_c, out_c, kernel_size, stride, padding)
        self.activation = nn.LeakyReLU(0.2)
        self.normalize = nn.BatchNorm1d(out_c)

    def forward(self, x):
        out = self.conv(x)
        out = self.activation(out)
        out = self.normalize(out)
        return out
```

```python
def forward(self, x):
    # convolution
    x = self.conv4(self.conv3(self.conv2(self.conv1(x))))

    # flattening
    x = self.flatten(x)

    # fully-connected layer
    x = self.act(self.bn1(self.fc1(x)))
    x = self.act(self.bn2(self.fc2(x)))
    x = self.act(self.bn3(self.fc3(x)))
    x = self.act(self.bn4(self.fc4(x)))
    out = self.fc5(x)
    return out
```

Comparing Model Performances

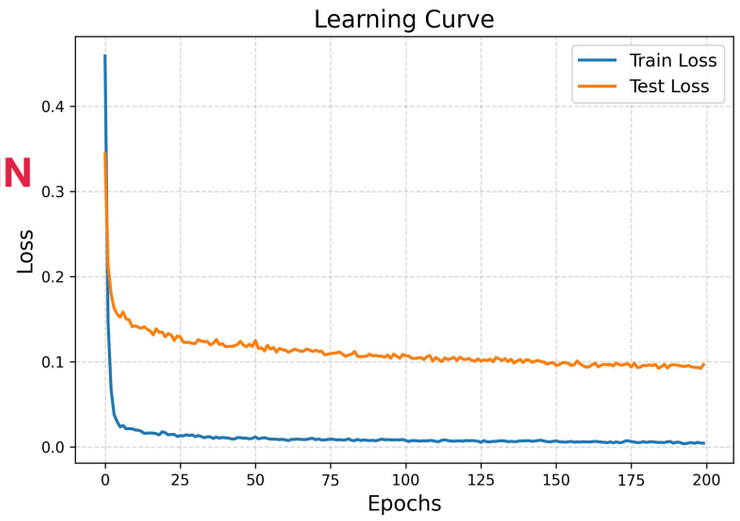
MLP

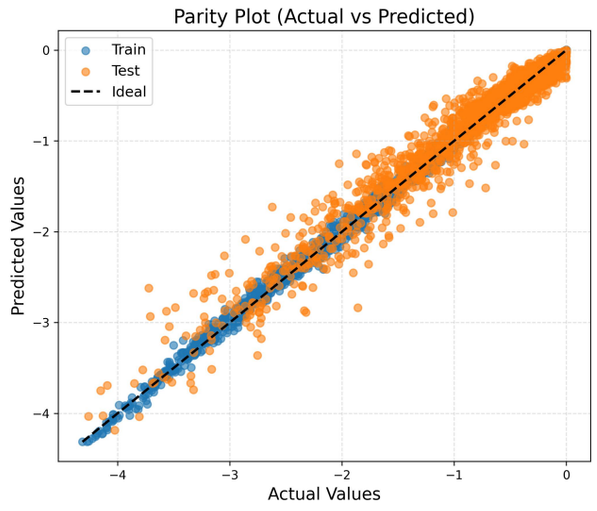




Dataset	MAE (eV/atom)	R2 Score
Training Set	0.026	0.995
Test Set	0.101	0.946

CNN





Dataset	MAE (eV/atom)	R2 Score
Training Set	0.016	0.998
Test Set	0.086	0.958