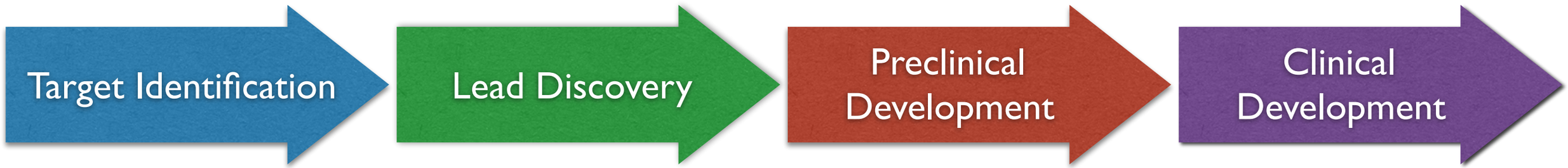
Machine Learning in Drug Discovery: Overview

Apr 26, 2019
Sunhwan Jo
SilcsBio, LLC
8 Market Street, Baltimore, MD, USA

Objective

- Show machine learning is not hard
- Share how machine learning is used in drug discovery

Machine Learning in Drug Discovery



- Gene-disease association
- Target druggability prediction

- Compound design
- Reaction path design
- Ligand based screening

- Biomarker identification
- Drug response classification

- Phenotype classification

Source: Vamathevan, J. et al. Nat. Rev. Drug Discov. 9, 242 (2019).

What is machine learning, anyway?

A quote from Geoffrey Hinton:

"… **you tell the computer exactly what to do in excruciating detail so it will perform the task you want -- and that is called programing.**

But there's now a new way of getting computers to do things. … **for any particular thing you wanted to do you just show it examples. You show what the inputs look like, you show what the correct outputs are for those inputs.** And to begin with, it gets it wrong. And you tell it that it got it wrong. And after awhile it starts getting it right. …"

Fizzbuzz

The "Fizz-Buzz test" is an interview question designed to help filter out programming job candidates.

> *Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".*

Fizzbuzz

The "Fizz-Buzz test" is an interview question designed to help filter out programming job candidates.

> *Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".*

```
for i in range(1, 101):
    if i % 15 == 0:
        print("fizzbuzz")
    elif i % 5 == 0:
        print("buzz")
    elif i % 3 == 0:
        print("fizz")
    else:
        print(i)
```

How about machine learning?

http://joelgrus.com/2016/05/23/fizz-buzz-in-tensorflow/

Fizzbuzz

Input	Output
1	1
2	2
3	fizz
4	4
5	buzz
6	fizz
7	7
8	8
9	fizz
10	buzz
11	11
12	fizz
13	13
14	14
15	fizz buzz
16	16
…	…

- Show inputs and correct outputs to machine learning model.

- Encode data
- Define model
- Train model
- Evaluate prediction

Fizzbuzz

Input	Output	Encoded Input	Encoded Output
1	1	0 0 0 0 1	0
2	2	0 0 0 1 0	0
3	fizz	0 0 0 1 1	1
4	4	0 0 1 0 0	0
5	buzz	0 0 1 0 1	2
6	fizz	0 0 1 1 0	1
7	7	0 0 1 1 1	0
8	8	0 1 0 0 0	0
9	fizz	0 1 0 0 1	1
10	buzz	0 1 0 1 0	2
11	11	0 1 0 1 1	0
12	fizz	0 1 1 0 0	1
13	13	0 1 1 0 1	0
14	14	0 1 1 1 0	0
15	fizz buzz	0 1 1 1 1	3
16	16	1 0 0 0 0	0
…	…	…	…

Fizzbuzz

Model: Single layer neural net with 100 hidden unit

```
NUM_DIGIT = 10      # number of binary digit
NUM_HIDDEN = 100    # number of hidden unit

# Define the model
model = torch.nn.Sequential(
    torch.nn.Linear(NUM_DIGITS, NUM_HIDDEN),
    torch.nn.ReLU(),
    torch.nn.Linear(NUM_HIDDEN, 4)
)
```

Fizzbuzz

Train: Train with numbers from 101-1023 (10 binary digits).

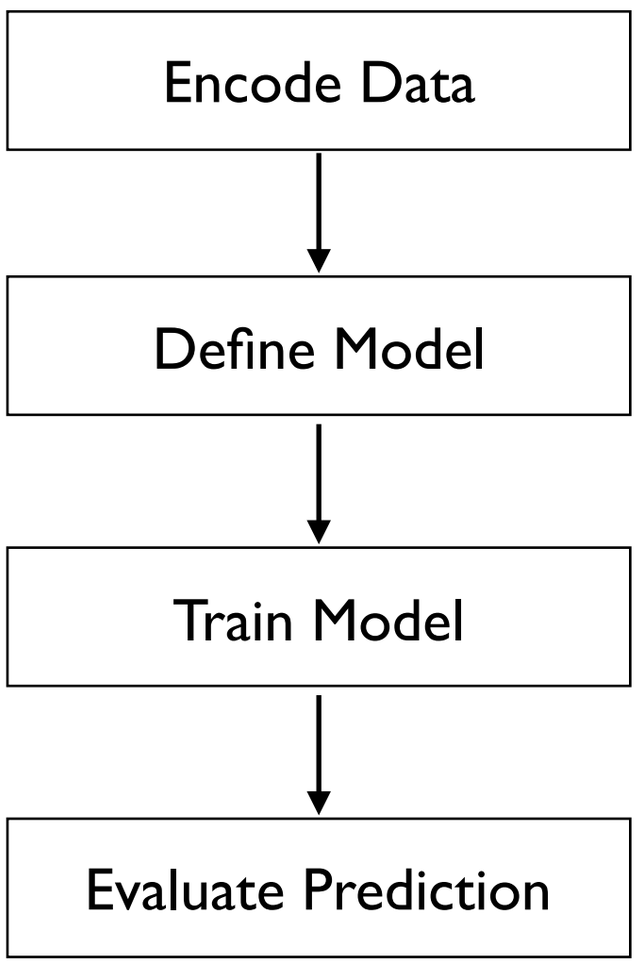
```
BATCH_SIZE = 128

# create validation set by randomly select 10% of train data
# the train set is also shuffled as we split the data.
(trainX, trainY), (valX, valY) = \
            split_train_test(X[100:], y[100:], test_size=0.1)

for start in range(0, len(trainX), BATCH_SIZE):
end = start + BATCH_SIZE
batchX = trainX[start:end]
batchY = trainY[start:end]

y_pred = model(batchX)
loss = loss_fn(y_pred, batchY)

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Fizzbuzz

Evaluate prediction:

```
# number from 1-100 (in binary digits)
testX, testY = X[:100], y[:100]

# model produce “probability” of 4 category
# pick the category with highest probability
y_pred = model(testX).argmax(1)

# decode the category to human readable output
for i in range(100):
    output = fizz_buzz_decode(i + 1, y_pred[i])
    print(output)
```

Four Steps in Machine Learning



Four Steps in Machine Learning

Encode Data

↓

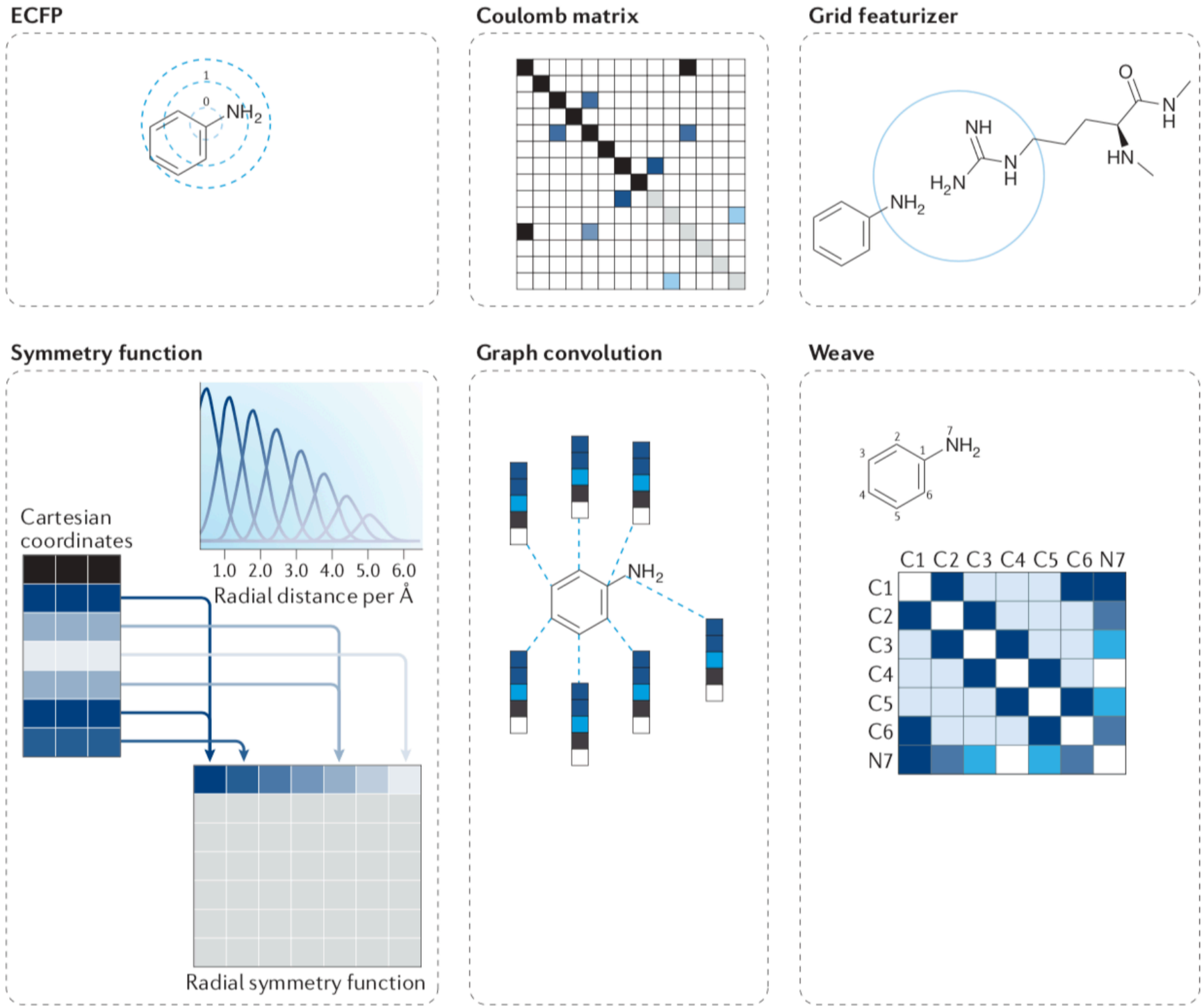
Define Model

↓

Train Model

↓

Evaluate Prediction



Source: Vamathevan, J. et al. Nat. Rev. Drug Discov. 9, 242 (2019).

Four Steps in Machine Learning

Encode Data

↓

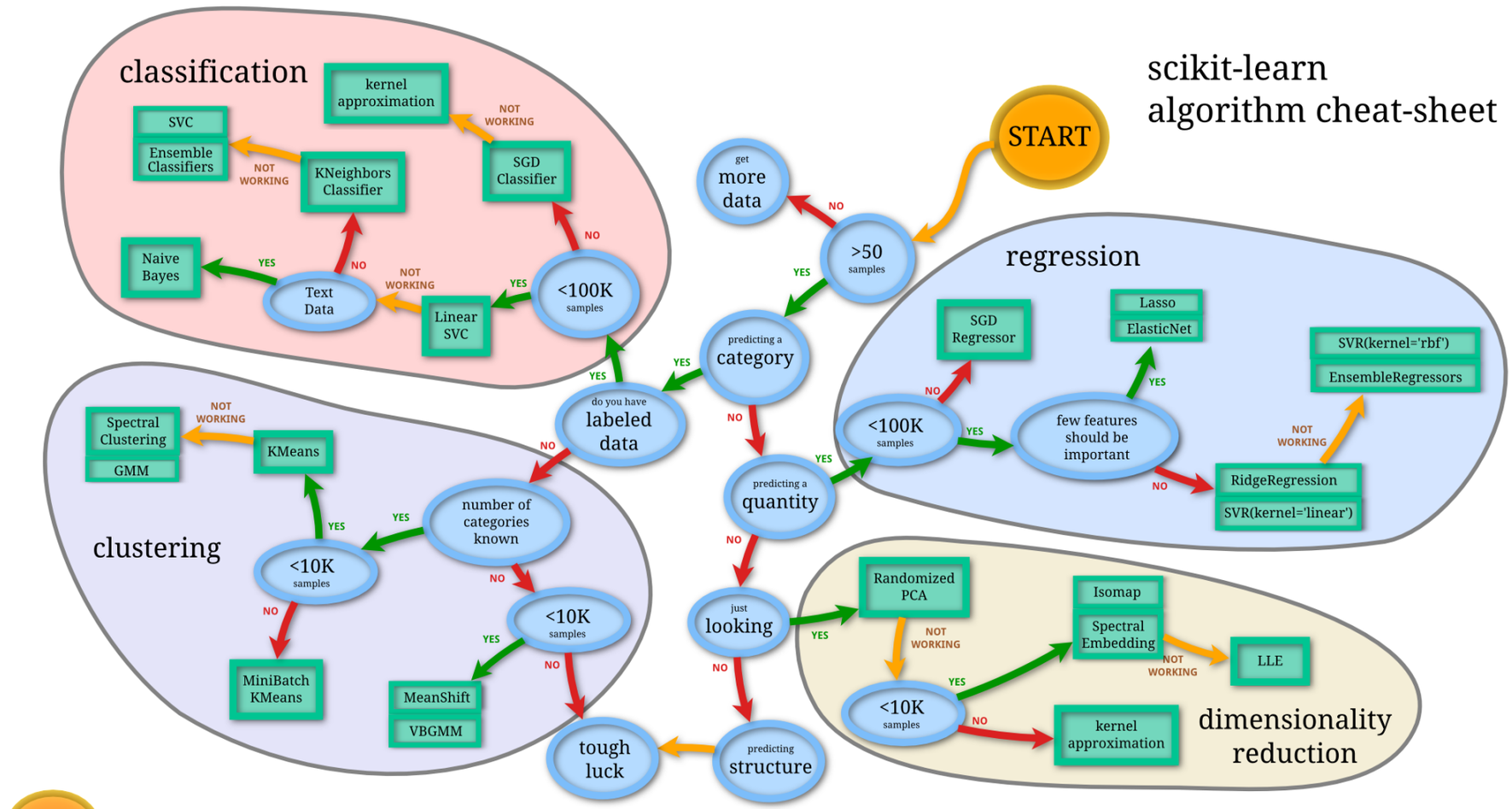
Define Model

↓

Train Model

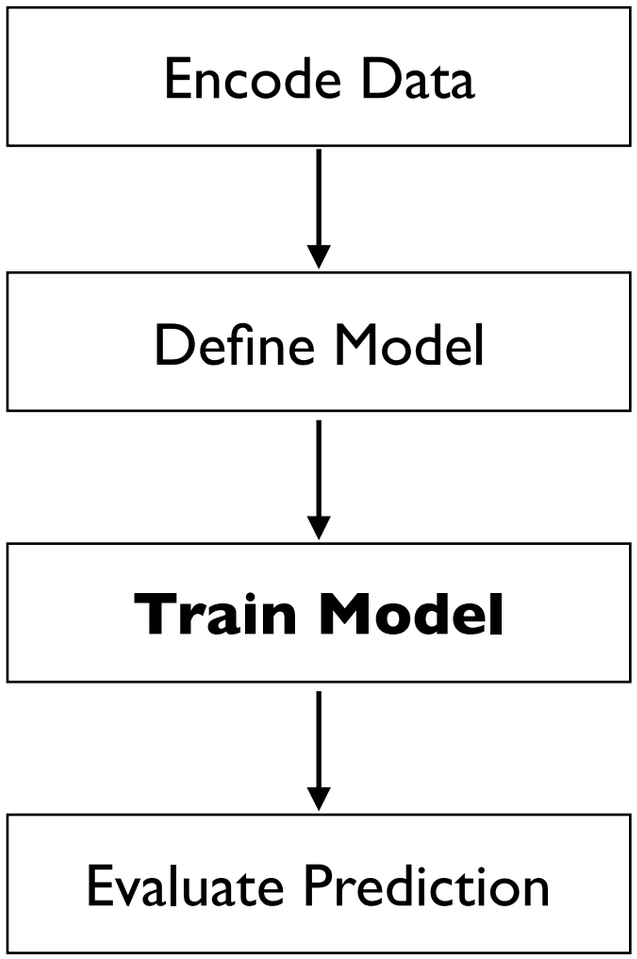
↓

Evaluate Prediction



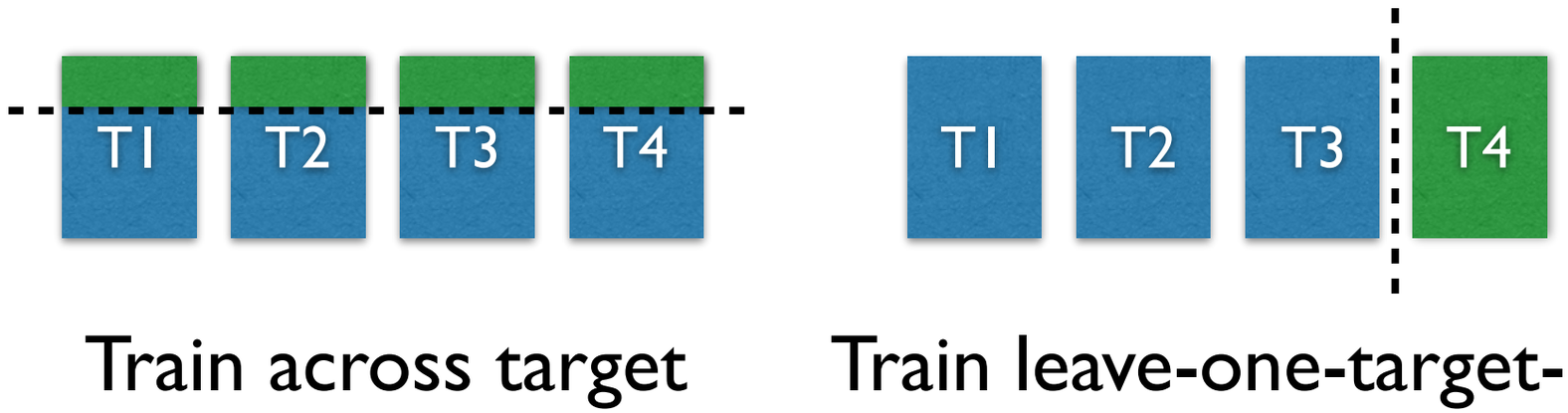
Source: scikit-learn.org

Four Steps in Machine Learning



- Needs to be mindful about train/test split.
- Unintended bias can be introduced by train data selection.

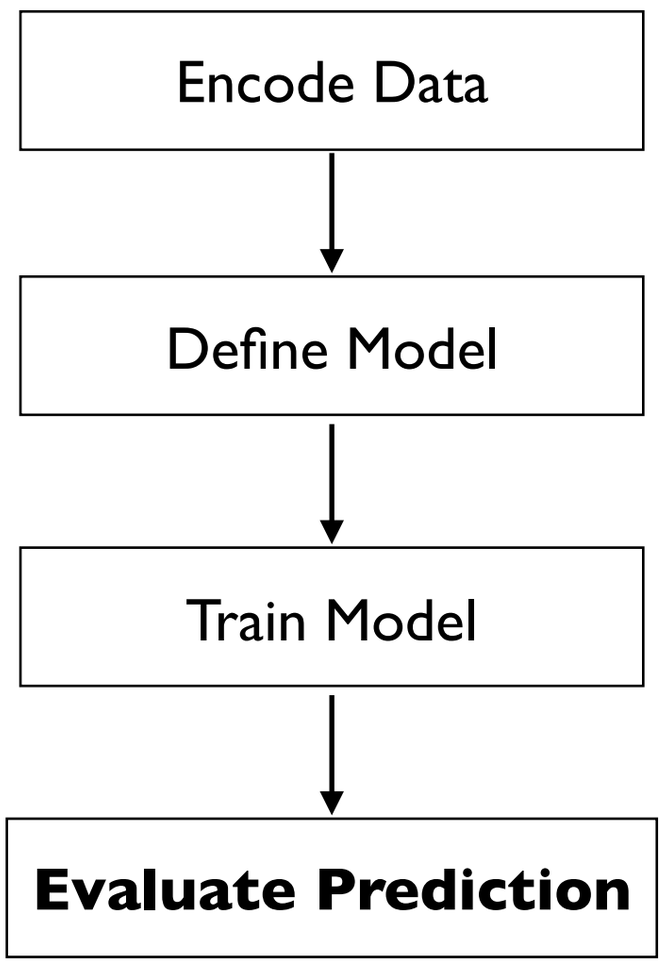
- Chen L., et. al., Hidden Bias in the DUD-E Dataset Leads to Misleading Performance of Deep Learning in Structure-Based Virtual Screening, ChemRxiv (2019)
- Sieg J., et. al., In Need of Bias Control: Evaluating Chemical Data for Machine Learning in Structure-Based Virtual Screening, JCIM (2019), 59 (3), pp 947–961



Train across target

Train leave-one-target-out

Four Steps in Machine Learning



- Test prediction in real, unseen data
- Evaluate the performance of model and provide feedback to entire process

Merck Molecular Activity Challenge

- 15 target data:
  - 3A4, CB1, DPP4, HIVINT, HIVPROT, LOGD, METAB, NK1, OX1, OX2, PGP, PPB, RAT_F, TDI, THROMBIN

```
In [21]: train.head()
```

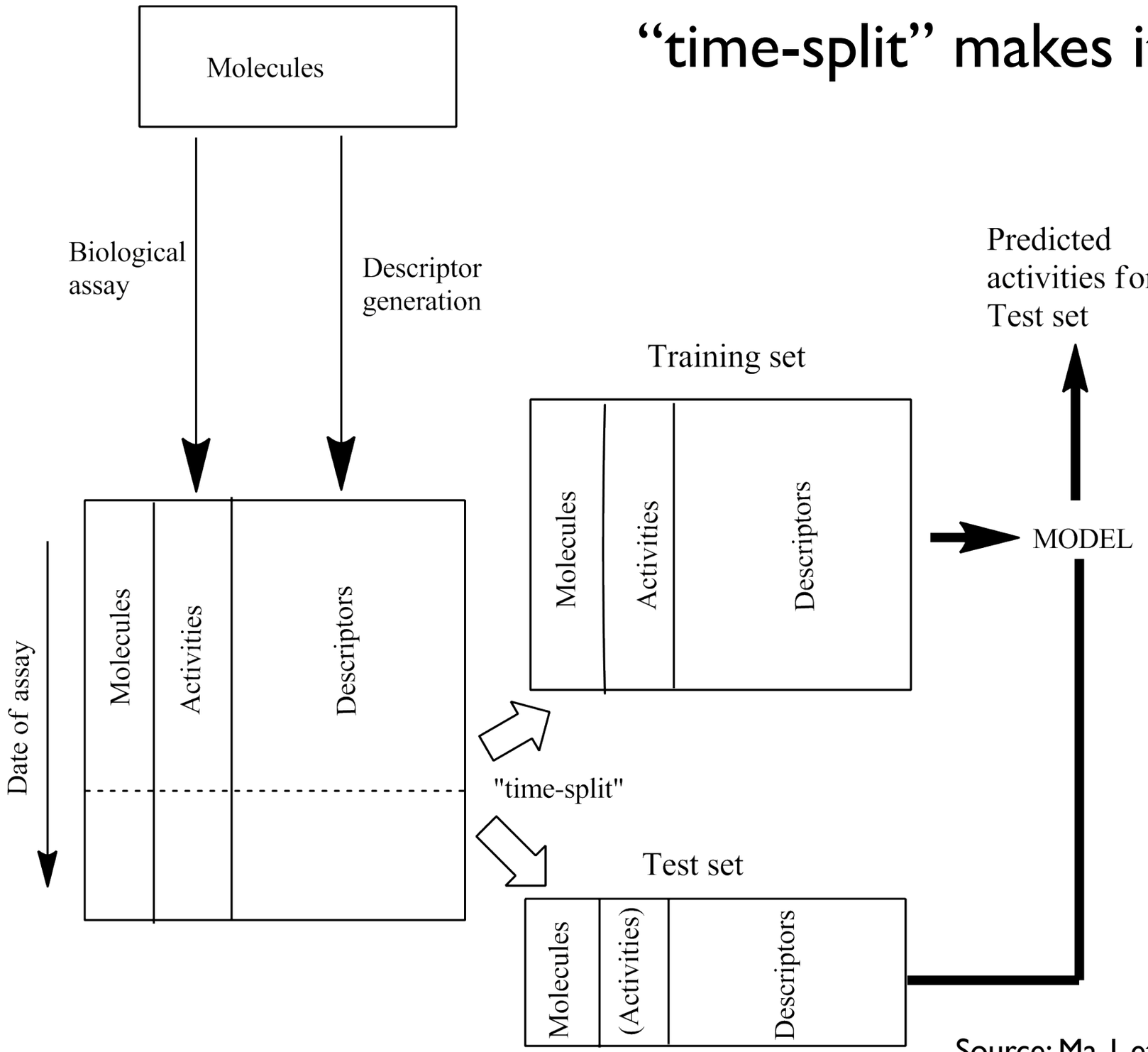
Out[21]:

	MOLECULE	Act	D_36	D_37	D_38	D_39	D_41	D_42	D_43	D_44	...	D_10736	D_10737	D_10738	D_10740	D_10741
0	M_5058	70.00	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	M_6406	74.68	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	M_12634	2.00	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	M_17594	0.00	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	M_17627	73.00	0	0	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 4374 columns

- Data set composed of molecule ID, activity, and descriptors.

Merck Molecular Activity Challenge



"time-split" makes it more challenging

Source: Ma, J. et. al., J. Chem. Inf. Model. 55, 263–274 (2015).

Merck Molecular Activity Challenge

Model: Multilayer neural net with 4000/2000/1000 hidden units.

```
input_dim = train_x.shape[1]
model = Sequential([
    Dense(4000, input_dim=input_dim),
    Activation('relu'),
    Dropout(0.25),

    Dense(2000),
    Activation('relu'),
    Dropout(0.1),

    Dense(1000),
    Activation('relu'),
    Dropout(0.1),

    Dense(1000),
    Activation('relu'),
    Dropout(0.1),

    Dense(1)
])
```

Merck Molecular Activity Challenge

- Implementing a CNN model is straightforward.
- Bulk of time is spent on finding a good hyper parameters (number of CNN layers, number of atom types to use, so on)
- The authors tested ~50 combinations of hyper parameters

- Each of three options of data preprocess (i.e., (1) no transformation, (2) logarithmic transformation, and (3) binary transformation) was selected.
- The number of hidden layers ranged from 1 to 4.
- The number of neurons in each hidden layer ranged from 100 to 4500.
- Each of the two activation functions (i.e., (1) sigmoid and (2) ReLU) was selected.
- DNNs were trained both (1) separately from an individual QSAR data set and (2) jointly from a data set combining all 15 data sets.
- The input layer had either no dropouts or 10% dropouts. The hidden layers had 25% dropouts.
- The network parameters were initialized as random values, and no unsupervised pretraining was used.
- The size of mini-batch was chosen as either 128 or 300.
- The number of epochs ranged from 25 to 350.
- The parameters for the optimization procedure were fixed as their default values. That is, learning rate is 0.05, momentum strength is 0.9, and weight cost strength is 0.0001.

Source: Ma, J. et. al., J. Chem. Inf. Model. 55, 263–274 (2015).

Merck Molecular Activity Challenge

Evaluation:

Table 2. Comparing Test R^2s of Different Models

data set	random forest	DNN with recommended parameter values	DNNs trained with arbitrarily selected, but reasonable, parameters: median R^2	worst case	best case	My Trial
3A4	0.469	**0.521**	*0.515*	0.476	0.550	0.514
CB1	0.283	**0.345**	*0.331*	0.294	0.377	**0.383**
DPP4	**0.232**	0.194	0.224	0.186	0.259	**0.252**
HIVINT	0.353	**0.403**	0.316	0.263	0.393	0.306
HIVPROT	0.530	**0.543**	0.524	0.467	0.581	0.461
LOGD	0.685	**0.822**	*0.808*	0.769	0.836	**0.829**
METAB	0.631	**0.687**	*0.661*	0.603	0.690	NaN
NK1	0.403	**0.448**	*0.414*	0.373	0.449	0.411
OX1	0.501	**0.657**	*0.584*	0.484	0.628	0.589
OX2	0.580	**0.707**	*0.621*	0.540	0.695	0.607
PGP	0.550	**0.616**	*0.592*	0.521	0.622	0.576
PPB	0.420	**0.598**	*0.561*	0.508	0.615	
RAT_F	0.098	**0.168**	*0.138*	0.065	0.187	0.089
TDI	**0.381**	0.350	0.339	0.319	0.377	0.327
THROMBIN	0.222	**0.375**	*0.351*	0.315	0.380	0.321
mean	***0.423***	***0.496***	***0.465***	***0.412***	***0.509***	

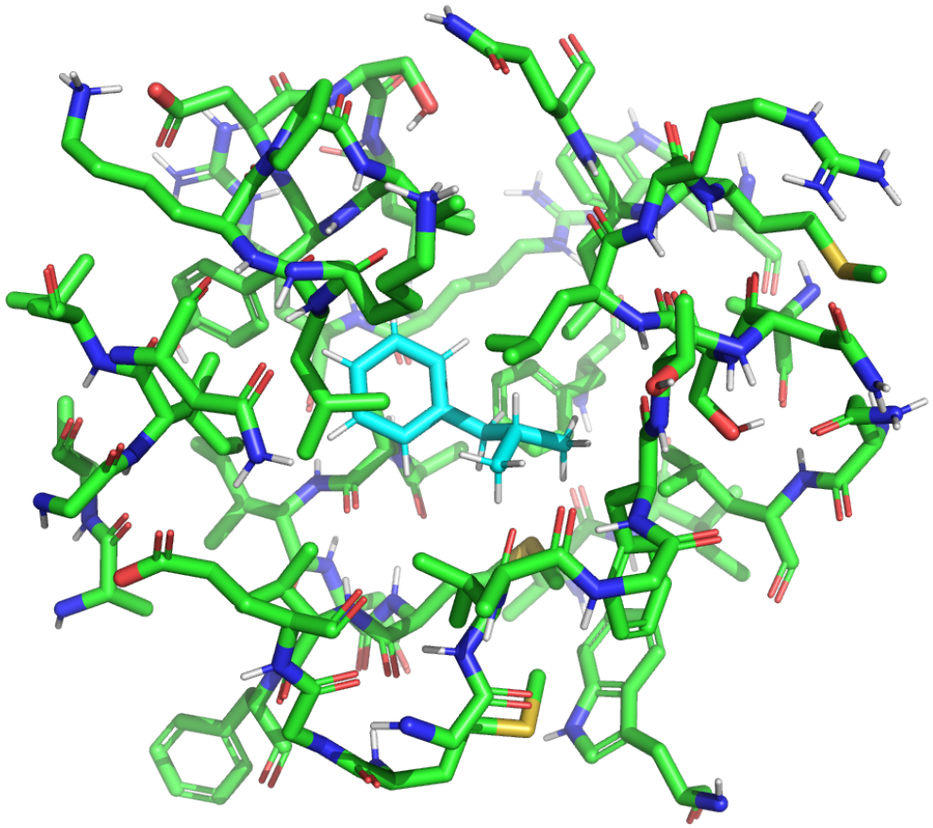
Source: Ma, J. et. al., J. Chem. Inf. Model. 55, 263–274 (2015).

CNN Scoring

- CNN (convolutional neural network)-based protein-ligand affinity prediction has gained popularity:
  - GNINA (Koes et. al.)
  - KDeep (De Fabrittis et. al.)
  - AtomWise

- Typically use voxelized atomic features

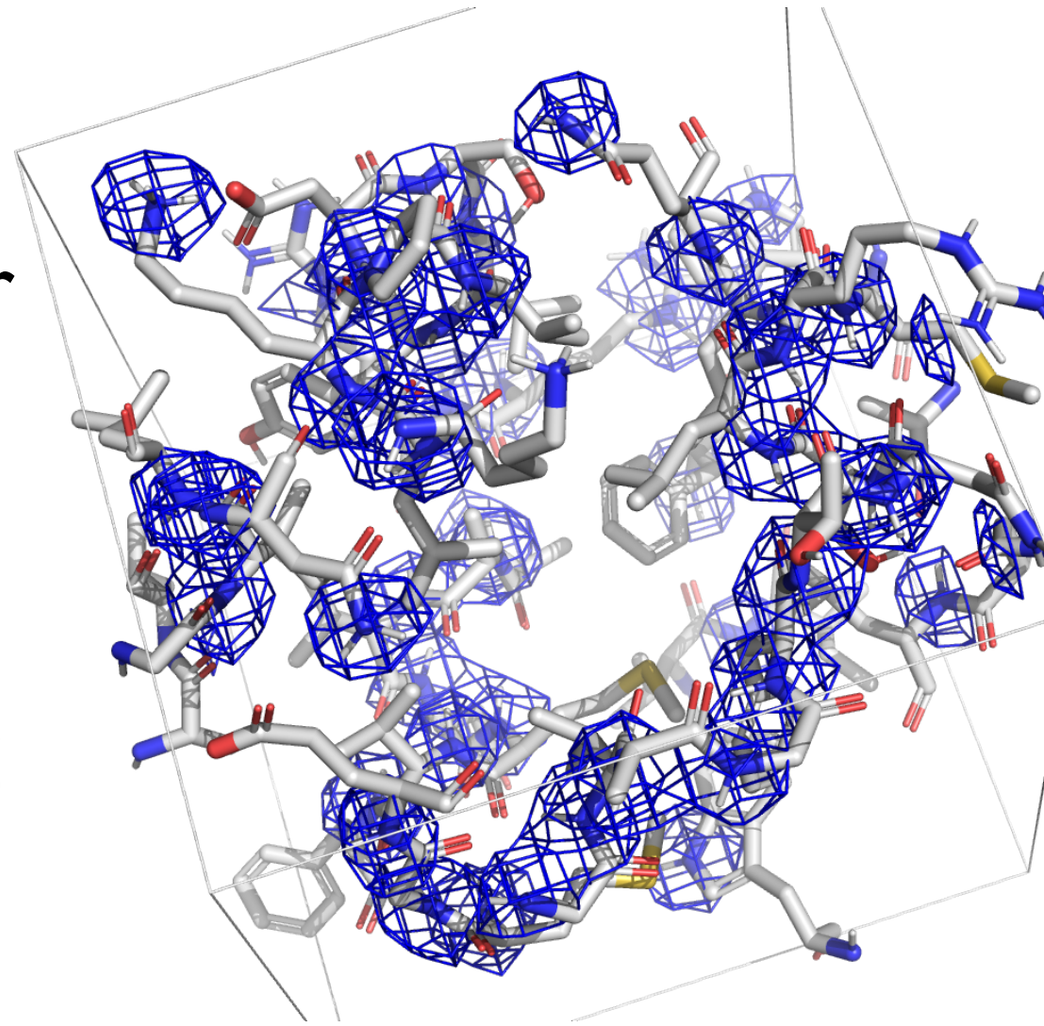
CNN Scoring



Atom Filter

Grid Featurizer

$$n(r) = 1 - \exp\left(-\left(\frac{r_{\text{vdw}}}{r}\right)^{12}\right)$$



Nitrogen Volxel Map (Channel)

CNN Scoring

Gnina

Table 1
The 35 atom types used in gnina. Carbon atoms are distinguished by aromaticity and adjacency to polar atoms ("NonHydrophobe"). Polar atoms are distinguished by hydrogen bonding propensity.

Receptor Atom Types	Ligand Atom Types
AliphaticCarbonXSHydrophobe	AliphaticCarbonXSHydrophobe
AliphaticCarbonXSNonHydrophobe	AliphaticCarbonXSNonHydrophobe
AromaticCarbonXSHydrophobe	AromaticCarbonXSHydrophobe
AromaticCarbonXSNonHydrophobe	AromaticCarbonXSNonHydrophobe
Calcium	Bromine
Iron	Chlorine
Magnesium	Fluorine
Nitrogen	Nitrogen
NitrogenXSAcceptor	NitrogenXSAcceptor
NitrogenXSDonor	NitrogenXSDonor
NitrogenXSDonorAcceptor	NitrogenXSDonorAcceptor
OxygenXSAcceptor	Oxygen
OxygenXSDonorAcceptor	OxygenXSAcceptor
Phosphorus	OxygenXSDonorAcceptor
Sulfur	Phosphorus
Zinc	Sulfur
	SulfurAcceptor
	Iodine
	Boron

16 protein + 19 ligand =
Total 35 channel (grid)

Kdeep

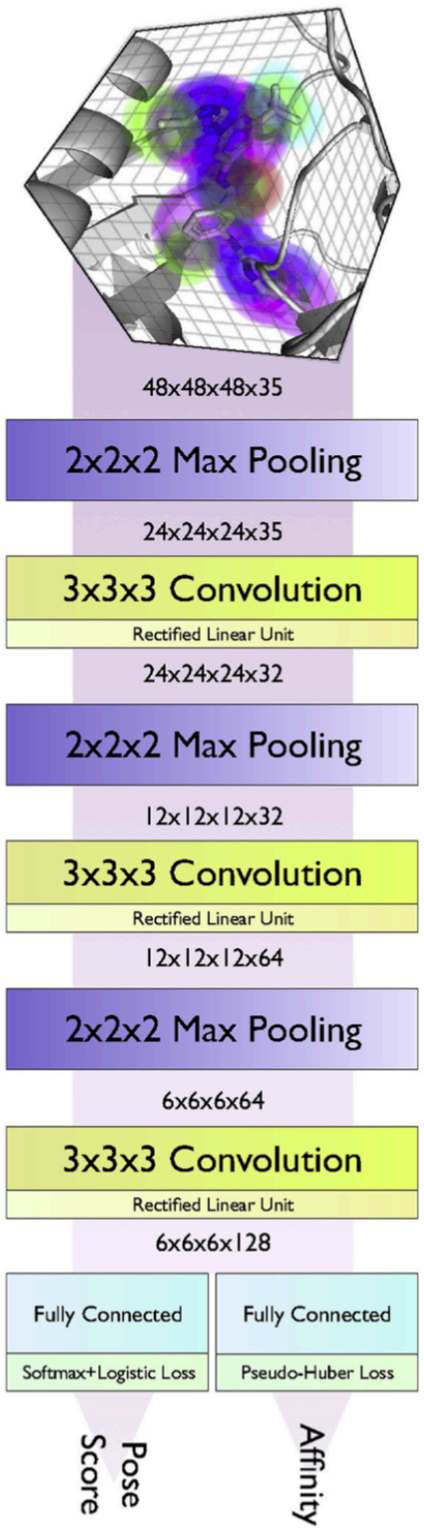
Table 2. Rules Defined for Three-Dimensional Descriptors Described in This Work

Property	Rule
Hydrophobic	Aliphatic or aromatic C
Aromatic	Aromatic C
Hydrogen bond acceptor	Acceptor 1 H-bond or S Spherical N; Acceptor 2 H-bonds or S Spherical O; Acceptor 2 H-bonds S
Hydrogen bond donor	Donor 1 H-bond or Donor S Spherical H with either O or N partner
Positive ionizable	Gasteiger positive charge
Negative ionizable	Gasteiger negative charge
Metallic	Mg, Zn, Mn, Ca, or Fe
Excluded volume	All atom types

8 protein + 8 ligand =
Total 8 channel (grid)

CNN Scoring

Gnina

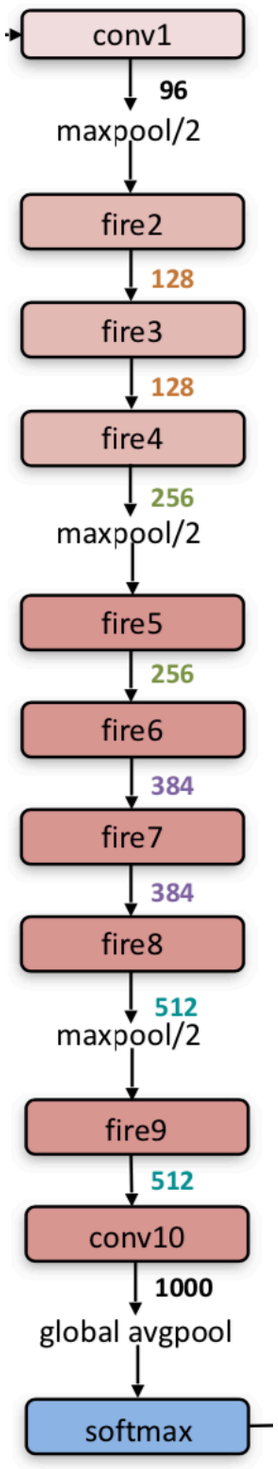



```
class GninaBlock(nn.Module):
    def __init__(self, in_size, in_channel, out_channel):
        self.pool = nn.AdaptiveMaxPool3d((out_size, out_size, out_size))
        self.conv = nn.Conv3d(in_channel, out_channel, kernel_size=3,
stride=1, padding=1)
        self.activation = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.conv(self.pool(x))
        return self.activation(x)

class GninaNetworkGenerator(nn.Module):
    def __init__(self):
        super(GninaNetworkGenerator, self).__init__()
        self.gpu_ids = gpu_ids
        features = [GninaBlock(48, 34, 32),
                    GninaBlock(24, 32, 64),
                    GninaBlock(12, 64, 128)]
        head = [Flatten(),
                nn.Linear(27648, 1)]
        model = features + head
        self.model = nn.Sequential(*model)

    def forward(self, input):
        if len(self.gpu_ids) and isinstance(input.data,
torch.cuda.FloatTensor):
            return nn.parallel.data_parallel(self.model, input,
self.gpu_ids)
        else:
            return self.model(input)
```

~ 30 lines of code!

CNN Scoring

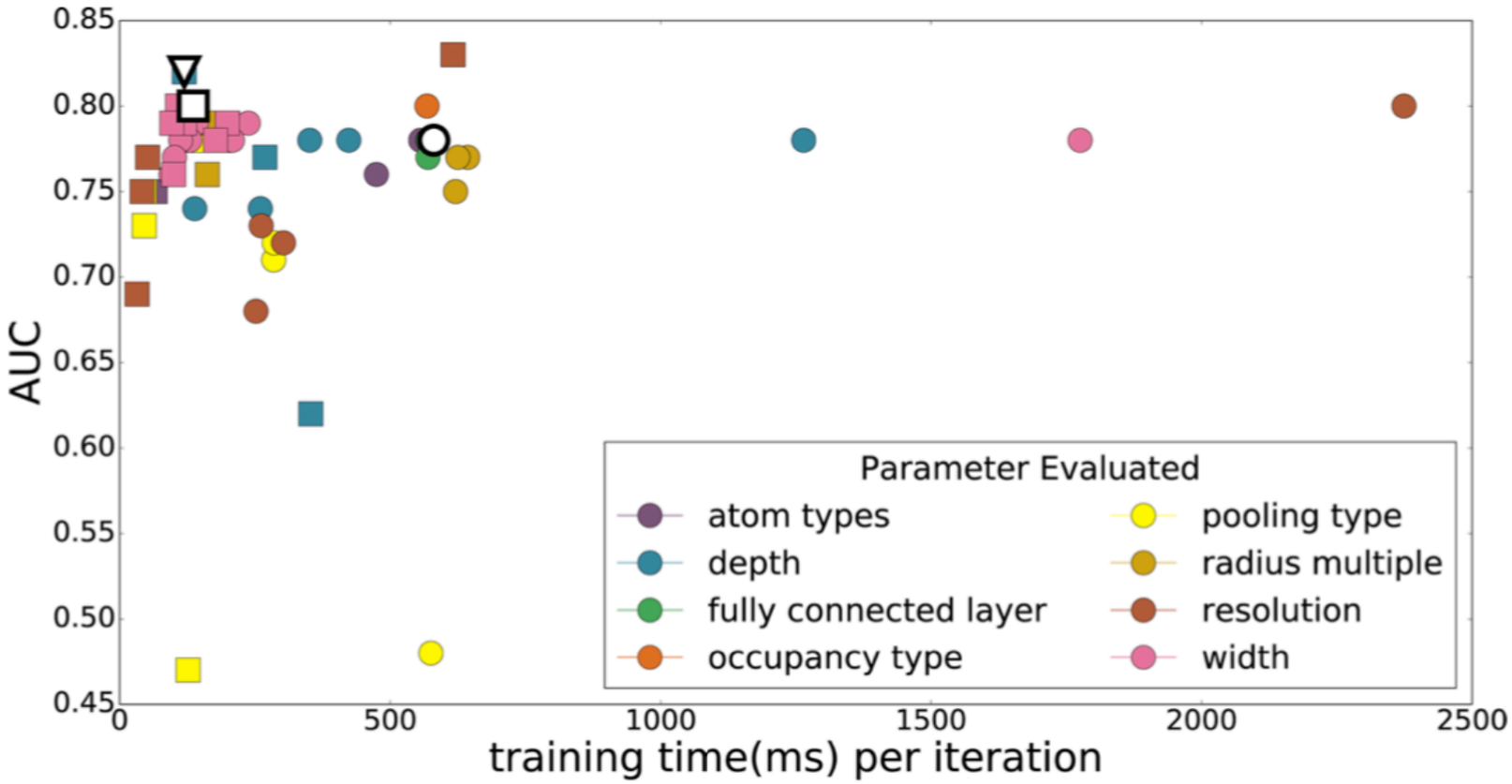
Kdeep




```
class Fire(nn.Module):
    def __init__(self, inplanes, squeeze_planes,
                 expand1x1_planes, expand3x3_planes):
        super(Fire, self).__init__()
        self.inplanes = inplanes
        self.squeeze = nn.Conv3d(inplanes, squeeze_planes, kernel_size=1)
        self.squeeze_activation = nn.ReLU(inplace=True)
        self.expand1x1 = nn.Conv3d(squeeze_planes, expand1x1_planes,
                                   kernel_size=1)
        self.expand1x1_activation = nn.ReLU(inplace=True)
        self.expand3x3 = nn.Conv3d(squeeze_planes, expand3x3_planes,
                                   kernel_size=3, padding=1)
        self.expand3x3_activation = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.squeeze_activation(self.squeeze(x))
        return torch.cat([
            self.expand1x1_activation(self.expand1x1(x)),
            self.expand3x3_activation(self.expand3x3(x))
        ], 1)


class Flatten(nn.Module):
    def forward(self, input):
        return input.view(input.size(0), -1)

class KDeepNetworkGenerator(nn.Module):
    def __init__(self, input_nc, norm_layer=nn.BatchNorm3d, use_dropout=False, gpu_ids=[], padding_type='reflect'):
        super(KDeepNetworkGenerator, self).__init__()
        self.gpu_ids = gpu_ids
        features = [nn.Conv3d(input_nc, 96, kernel_size=3, stride=2, padding=1),
                    nn.ReLU(inplace=True),
                    Fire(96, 16, 64, 64),
                    Fire(128, 16, 64, 64),
                    Fire(128, 32, 128, 128),
                    nn.MaxPool3d(kernel_size=3, stride=2, ceil_mode=True),
                    Fire(256, 32, 128, 128),
                    Fire(256, 48, 192, 192),
                    Fire(384, 48, 192, 192),
                    Fire(384, 64, 256, 256)]
        head = [nn.AdaptiveAvgPool3d((2, 2, 2)),
                Flatten(),
                nn.Linear(4096, 1)]
        model = features + head
        self.model = nn.Sequential(*model)

    def forward(self, input):
        if len(self.gpu_ids) and isinstance(input.data, torch.cuda.FloatTensor):
            return nn.parallel.data_parallel(self.model, input, self.gpu_ids)
        else:
            return self.model(input)
```

~ 55 lines of code!

CNN Scoring

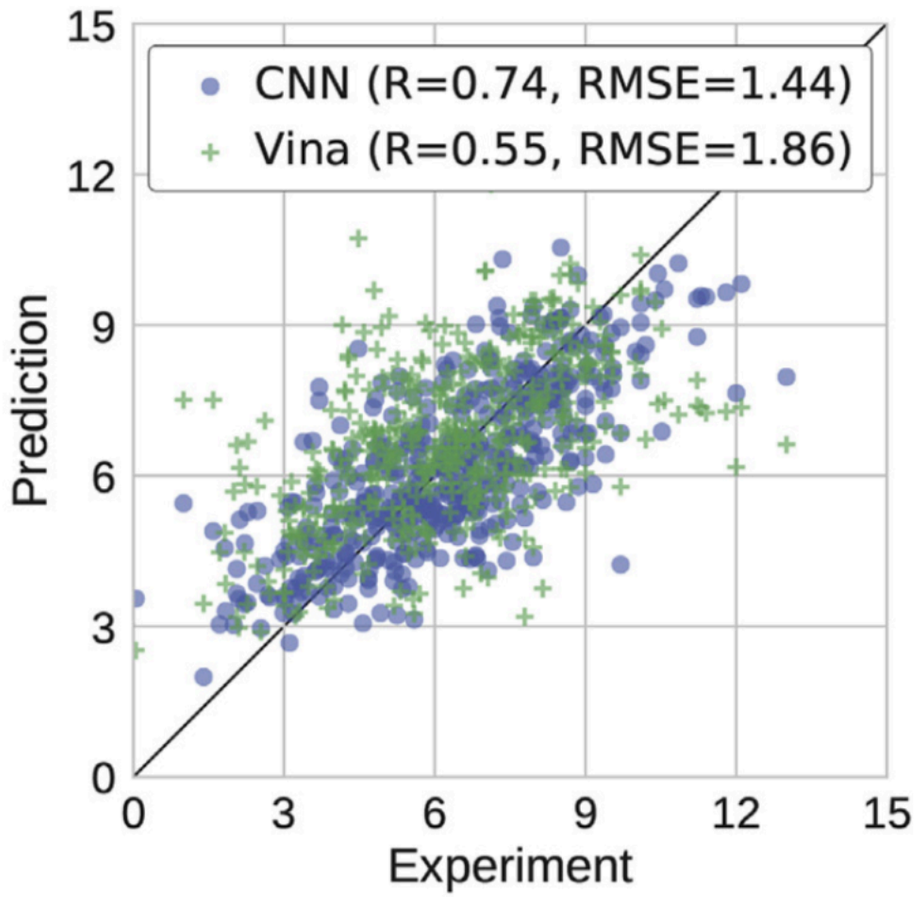


- Implementing a CNN model is not challenging
- Bulk of time is spent on finding a good hyper parameters (number of CNN layers, number of atom types to use, so on)
- Having a good validation data set that give objective performance evaluation is important.

Source: Ragoza, M. et. al., J. Chem. Inf. Model. 57, 942–957 (2017).

CNN Scoring

Evaluation:



- Implementing a CNN model is not challenging
- Bulk of time is spent on finding a good hyper parameters (number of CNN layers, number of atom types to use, so on)
- Having a good validation data set that give objective performance evaluation is important.

Source: Ragoza, M. et. al., J. Chem. Inf. Model. 57, 942–957 (2017).

Conclusion

- Machine Learning Algorithm:
  - Encoding
  - Model
  - Train
  - Evaluation

- Building a model is an easy part.
- Finding a good hyper parameters and building training/ evaluation protocol is harder and requires a lot of trial and error.
- Having a good (and large) training/validation is important for successful ML project.