**# create initial DOE matrix**

import numpy as np

import pandas as pd

n1 = pd.DataFrame([8,32,128,512,2048])

n1.columns=['n1']

n1['key'] = 0

k1 = pd.DataFrame([3,5,7])

k1.columns=['k1']

k1['key'] = 0

p1 = pd.DataFrame([2,4])

p1.columns=['p1']

p1['key'] = 0

a1 = pd.DataFrame(['relu','tanh'])

a1.columns=['a1']

a1['key'] = 0

n2 = pd.DataFrame([8,32,128,512,2048])

n2.columns=['n2']

n2['key'] = 0

k2 = pd.DataFrame([3,5,7])

k2.columns=['k2']

k2['key'] = 0

p2 = pd.DataFrame([2,4])

p2.columns=['p2']

p2['key'] = 0

a2 = pd.DataFrame(['relu','tanh'])

a2.columns=['a2']

a2['key'] = 0

d = pd.DataFrame([.25,.5])

d.columns=['d']

d['key'] = 0

doe\_df = n1.merge(k1, on='key')

doe\_df = doe\_df.merge(p1, on='key')

doe\_df = doe\_df.merge(a1, on='key')

doe\_df = doe\_df.merge(n2, on='key')

doe\_df = doe\_df.merge(k2, on='key')

doe\_df = doe\_df.merge(p2, on='key')

doe\_df = doe\_df.merge(a2, on='key')

doe\_df = doe\_df.merge(d, on='key')

doe\_df.drop(labels=['key'], axis=1,inplace = True)

**# remove trails where n2 > n1. It keeps 2,880 trails out of 7,200**

doe\_df = doe\_df[doe\_df.n1>doe\_df.n2]

**# remove trails where k2 => k1. It keeps 1,920 trails**

doe\_df = doe\_df[doe\_df.k1>=doe\_df.k2]

doe\_df = doe\_df.reset\_index()

**Models**

Architectures of the applied basic neural network models used in this research, pseudo code, and finalized hyperparameters have been as follows.

b-VAE network (both classical full factorial 24 and MNIST tuning examples):

Encoder: Flatten input with 4 (full factorial 24) or 9 (MNIST tuning) channels; Dense (512, activation='relu'); Dense (32, activation='relu'); two Dense (latent\_dim=2)

Decoder: Dense (32, input\_dim=2, activation='relu'), Dense (512, activation='relu'), Dense (original\_dim=4 (full factorial 24) or 11 (MNIST tuning)), activation='sigmoid')

Loss=binary\_crossentropy+beta\*KL; beta=0.3; optimizer=adam; batch\_size=256; epochs=500; data sets duplications: full factorial 24: train x50 and test x30; MNIST tuning: train x5 and test x3

Classification convolutional neural network:

Input 784 (28x28); Conv2D (n1, kernel\_size=(k1, k1), padding='same', activation=a1); MaxPool2D (pool\_size=(p1,p1)

Conv2D (n2, kernel\_size=(k2, k2), padding='same', activation=a2); MaxPool2D (pool\_size=(p2, p2)); Flatten (); Dropout (d); Dense (num\_classes=10), activation='sigmoid')

Loss= categorical\_crossentropy; optimizer=adam; metric=accuracy; batch\_size = 64; epochs = 10

Sequential neural network (importance evaluation model):

Dense (16, kernel\_initializer='normal', activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=1e-5, l2=1e-4),

bias\_regularizer=regularizers.l2(1e-4), activity\_regularizer=regularizers.l2(1e-5))

Dense (4, kernel\_initializer='normal', activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=1e-5, l2=1e-4),

bias\_regularizer=regularizers.l2(1e-4), activity\_regularizer=regularizers.l2(1e-5))

Dense (1, kernel\_initializer='normal')

Loss='mean\_squared\_error'; optimizer='adam'; batch\_size = 128; epochs = 100; data sets duplications: square and polar grids, random: train x50 and test x30; initial design: train x5 and test x3