ASSUMPTION UNIVERSITY

Vincent Mary School of Science and Technology

Department of Information Technology

IT4292 SENIOR PROJECT II

Final Report

Neural Network Learning Generalization with Customized Gaussian Noise Injection

Project Advisor:

Asst. Prof. Dr. Thitipong Tanprasert

Project Committee:

Asst. Prof. Dr. AnilKumar Kothalil Gopalakrishnan

Asst. Prof. Dr. Benjawan Srisura

Submitted By: Mr.Chaitach Vadhanachai 5915132

Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science and Technology in Information Technology and Computer Science

Abstract

This report provides a study of Neural Network Learning Generalization with Gaussian Noise Injection. Though such injections have been studied when applied to data. There have been few studied on understanding the Neural Network Generalization when applied to network activation.

Here we derive the generalization of the Neural Network with gaussian noise injection, obtained by scaling the injection noise to original data. we show analytically and empirically that such generalization produces the accuracy with different datasets

Table of content

Objective

This project will aim to uncover these mechanisms to better understand the generalization in neural network with GNIs. We aim to get the result from train the neural network model with the difference datasets that have been injected with the various of noise data. Also, to determine that is noise adding technique could improving generalization in neural network.

Introduction

Gaussian noise injections (GNIs) are a simple and widely-used regularization method for training neural networks, noise injection techniques involve multiplying samples from a noise distribution to the weights of a neural network during training. Such techniques have numerous demonstrated advantages. Models that have been trained with noise frequently generalize to new data better and are less likely to overfit. [1] Explicit Regularization in Gaussian Noise Injections Research

Numerous research has been conducted on the advantages of noise data, but the exact processes by these injections work remain unclear. Early results suggest that to reduce overfitting by training the network on noise data and by changing the complexity of the model network. Most research said, introducing noise to a dataset can indeed have a positive influence on the model.[2] Train Neural Network With noise [3] Noise Affect generalization

Concretely our studies are:

- We derive an analytic form for a generalizer with GUIs
- We scaling the value of noise to inject to the original data
- Finally, we show analytically and empirically about the result of generalization accuracy value when finish training the model

Method

How to do

Datasets

Firstly, we research the datasets from various sources and try to get the best suitable datasets which contained many kinds of attributes with the output which are Boolean type (0,1). we choose the appropriate datasets for this experiment which the data must be the binary classification data. Which most data used in this experiment retrieve from Kaggle and Google Data Finder. [4] Kaggle [5] Data finder

Secondly, after we get all datasets that we will use in our experiment, we balance the data by using out sprit train-test program and get the equal output of data and generate in to new csv file for training. This method we sprit train-test data and prepare the data to train in the model.

Dataset that we use:

- Breast Cancer Dataset (570 of Data Row)
- Diabetes Dataset (769 of Data Row)
- Smoking Dataset (55693 of Data Row)
- Wine Quality Dataset (1600 of Data Row)
- Heart Disease Dataset (1026 of Data Row)

Breast Cancer Dataset

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (local variation in radius lengths)

f) compactness (perimeter^2 / area - 1.0)

g) concavity (severity of concave portions of the contour)

h) concave points (number of concave portions of the contour)

i) symmetry

j) fractal dimension ("coastline approximation" - 1)

k) diagnosis (0 or 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image,

resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius. [6] Breast Cancer Dataset

Diabetes Dataset

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. [7] Diabetes Dataset

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age (years)

Outcome: Class variable (0 or 1)

Smoking Dataset

This dataset is a collection of basic health biological signal data

To determine the presence or absence of smoking through bio-signals

Columns contained

age : 5-years gap

height(cm)

weight(kg)

waist(cm) : Waist circumference length

eyesight(left)

eyesight(right)

hearing(left)

hearing(right)

systolic : Blood pressure

relaxation : Blood pressure

fasting blood sugar

Cholesterol : total

triglyceride

HDL : cholesterol type

LDL : cholesterol type

hemoglobin

Urine protein

serum creatinine

AST : glutamic oxaloacetic transaminase type

ALT : glutamic oxaloacetic transaminase type

Gtp : γ-GTP

oral : Oral Examination status

Dental caries

Smoking (1 or 0)

To prepare smoking dataset, we remove other Boolean data type such as hearing, urine protein, and dental caries. Which could interrupt the training process and get the inaccurate accuracy. [8] Smoking Dataset

Wine Quality Dataset

The dataset is related to red and white variants of the Portuguese "Vinho Verde" wine.

Input variables (based on physicochemical tests):

1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar

5 - chlorides
6 - free sulfur dioxide

7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
Output variable (based on sensory data) [9] Wine Quality Dataset

Heart Disease Dataset

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V [10] Heart Disease Dataset

Attribute Information:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 0 = normal; 1 = fixed defect; 2 = reversable defect
    The names and social security numbers of the patients were recently removed from the database, replaced with dummy values.

Example of Diabetes Original Dataset

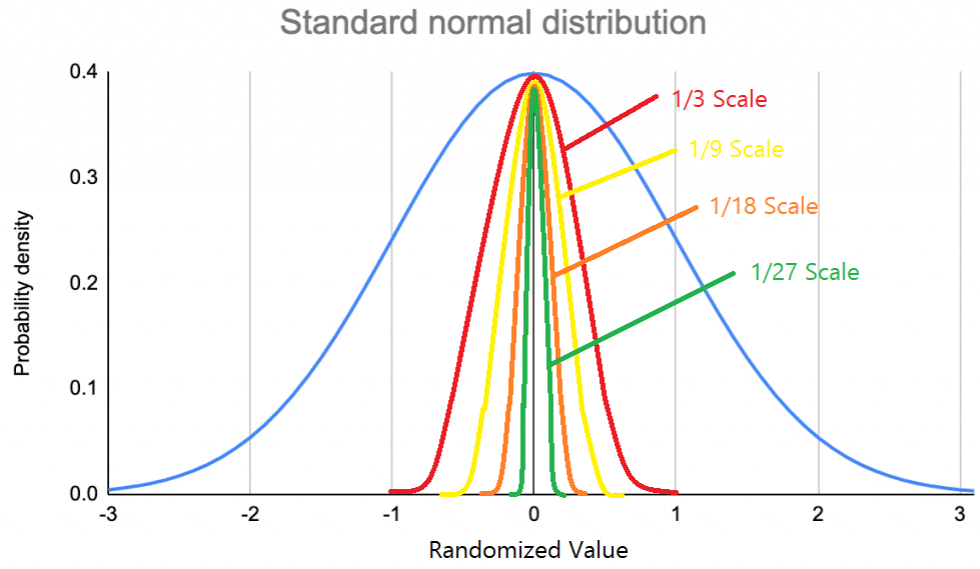
	A	B	C	D	E	F	G	H	I
1	Pregnancie	Glucose	BloodPress	SkinThickn	Insulin	BMI	DiabetesPe	Age	Outcome
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1

Example of Diabetes Train-Test Sprit Dataset (Balance Data)

	A	B	C	D	E	F	G	H	I
200	1	147	94	41	0	49.3	0.358	27	1
201	3	187	70	22	200	36.4	0.408	36	1
202	6	162	62	0	0	24.3	0.178	50	1
203	4	136	70	0	0	31.2	1.182	22	1
204	0	181	88	44	510	43.3	0.222	26	1
205	8	154	78	32	0	32.4	0.443	45	1
206	1	128	88	39	110	36.5	1.057	37	1
207	0	123	72	0	0	36.3	0.258	52	1
208	6	190	92	0	0	35.5	0.278	66	1
209	1	126	60	0	0	30.1	0.349	47	1
210	1	93	70	31	0	30.4	0.315	23	0
211	1	97	70	40	0	38.1	0.218	30	0
212	1	143	74	22	61	26.2	0.256	21	0
213	2	123	48	32	165	42.1	0.52	26	0
214	4	110	66	0	0	31.9	0.471	29	0
215	6	103	66	0	0	24.3	0.249	29	0
216	0	95	80	45	92	36.5	0.33	26	0
217	9	124	70	33	402	35.4	0.282	34	0
218	0	98	82	15	84	25.2	0.299	22	0
219	1	84	64	23	115	36.9	0.471	28	0
220	0	152	82	39	272	41.5	0.27	27	0

Generate noise data

When all datasets are ready to use for training, we contribute the program to generate and inject noise data into original data. In this method, firstly, we need to get the scale value for each data column by getting the max value of each column and calculate the average value from each column. Then, we calculate the scale value by subtract the max value with average value from each column then divide by 9 18 and 27. By using the value of 9,18, and 27 to divide the scale value, we will get the noise value which is not getting out of range from the original value range. Lastly, we generate new csv files for noise data by each divided value.



Example of Wine Quality original Dataset

	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acidit	volatile acic	citric acid	residual su	chlorides	free sulfur	total sulfur	density	pH	sulphates	alcohol	Output
2	7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	1
3	7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	1
4	8.5	0.28	0.56	1.8	0.092	35	103	0.9969	3.3	0.75	10.5	1
5	7.5	0.52	0.16	1.9	0.085	12	35	0.9968	3.38	0.62	9.5	1
6	5.4	0.835	0.08	1.2	0.046	13	93	0.9924	3.57	0.85	13	1
7	9.6	0.32	0.47	1.4	0.056	9	24	0.99695	3.22	0.82	10.3	1
8	12.8	0.3	0.74	2.6	0.095	9	28	0.9994	3.2	0.77	10.8	1
9	12.8	0.3	0.74	2.6	0.095	9	28	0.9994	3.2	0.77	10.8	1
10	5.2	0.48	0.04	1.6	0.054	19	106	0.9927	3.54	0.62	12.2	1

Example of Wine Quality Noise Dataset (Scale value divided by 9)

	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acidit	volatile acic	citric acid	residual su	chlorides	free sulfur	total sulfur	density	pH	sulphates	alcohol	Output
2	7.289359	0.586244	-0.02283	1.156651	0.039566	27.35481	20.06437	0.993387	3.359803	0.647777	10.1227	1
3	7.299499	0.481061	-0.0278	2.740377	0.103765	4.829714	10.6962	0.996654	3.359883	0.568431	9.109432	1
4	7.87117	0.3808	0.778657	1.683766	0.071076	35.56585	99.47238	0.99673	3.263912	0.764074	10.43612	1
5	7.125909	0.499908	0.17468	2.347046	0.045759	2.545786	32.45266	0.996347	3.350827	0.50897	9.506654	1
6	6.033784	0.750643	0.133922	0.479668	-0.0449	19.92279	106.3148	0.993029	3.576785	0.804763	12.71043	1
7	9.875736	0.228714	0.416344	1.283957	0.018147	9.216195	40.4453	0.997788	3.226386	0.791144	10.51945	1
8	13.97483	0.39274	0.703913	3.54455	0.070242	14.00067	63.24729	0.999525	3.236643	0.593108	11.38567	1
9	11.7685	0.220953	0.800272	3.149721	0.081328	14.31498	53.50992	0.998048	3.196158	0.807592	10.79055	1
10	4.505171	0.471147	-0.04106	1.090979	0.036213	15.50103	99.03278	0.993005	3.480286	0.620443	12.01256	1

Example of Wine Quality Noise Dataset (Scale value divided by 18)

	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acidit	volatile acic	citric acid	residual su	chlorides	free sulfur	total sulfur	density	pH	sulphates	alcohol	Output
2	7.23266	0.648194	-0.03041	0.878667	0.046832	17.26747	18.31819	0.994585	3.378037	0.479579	9.48439	1
3	8.013714	0.627386	0.07059	1.940863	0.079448	8.887416	11.62253	0.99651	3.356075	0.636948	9.644892	1
4	8.025107	0.335786	0.573804	1.681127	0.100888	37.20855	81.76243	0.996364	3.342258	0.710782	10.84573	1
5	7.649961	0.571846	0.126745	2.44509	0.012978	9.69612	61.79469	0.996911	3.388004	0.546073	9.361493	1
6	4.420228	0.796067	0.085209	1.620216	0.027177	9.300208	89.10192	0.992657	3.590751	0.925333	12.91732	1
7	10.28965	0.352402	0.491927	1.667198	0.074255	8.093578	2.641376	0.997063	3.240683	0.842195	10.05799	1
8	12.03098	0.244449	0.74507	2.503434	0.121203	6.026997	16.48323	0.999192	3.220905	0.722808	10.76294	1
9	12.80865	0.241654	0.816129	2.114581	0.063811	11.53594	18.27364	0.998798	3.189684	0.725475	10.89252	1
10	5.518082	0.46101	0.047497	2.155941	-0.01676	17.80566	83.98495	0.992348	3.535247	0.669276	12.52235	1

Example of Wine Quality Noise Dataset (Scale value divided by 27)

	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acidit	volatile acic	citric acid	residual su	chlorides	free sulfur	total sulfur	density	pH	sulphates	alcohol	Output
2	7.500168	0.663481	0.106526	1.640128	-0.00049	17.62071	37.85311	0.993768	3.37608	0.451486	10.00638	1
3	8.003707	0.632885	0.016555	1.500765	0.097227	10.42899	9.649266	0.99704	3.398064	0.584311	9.50194	1
4	8.804072	0.183357	0.548992	2.477573	0.109659	36.49119	117.2292	0.997279	3.311569	0.688119	10.43795	1
5	7.671727	0.464798	0.147022	1.867424	0.048987	14.64263	20.52622	0.996441	3.391972	0.621618	9.447839	1
6	5.096564	0.830186	0.083913	1.143174	0.037604	9.116466	98.15966	0.992505	3.5748	0.77624	13.14514	1
7	9.203519	0.305283	0.417191	1.467583	0.049761	8.709502	17.50714	0.996364	3.240998	0.790509	10.80039	1
8	12.75454	0.353287	0.77519	2.146982	0.125567	8.553717	27.77431	0.999633	3.192729	0.781253	10.53925	1
9	13.58056	0.270918	0.728974	2.785656	0.128338	13.1092	33.42671	0.999841	3.147018	0.724916	10.49091	1
10	5.858719	0.550476	0.038529	1.286851	0.038481	21.88365	129.4676	0.993223	3.559491	0.701353	12.12303	1

Build Model

We build the difference NN model that suitable for each dataset. To be more clearly, each datasets have the difference input size, hence for the input dimension in dense layer must be difference for each dataset and some datasets need to add more dense layers to handle the data.

Train model

As we already prepare the original data and noise data in csv file. We could import the data and train each dataset with the suitable model that we have created for each dataset. We train the all data with epochs 500 and use early stopping method to declare the best epoch value with the highest accuracy for each data. We also plot the accuracy graph in every training.

Code and Experiments

Main Code

Generate noise to original data

```
]: def noise9_data_generator(data):
       generate_data =[]
       for col in data.columns:
           scale = (data[col].max()-data[col].mean())/9
           noises = []
           for idx in range(len(data[col])):
               random = np.random.normal(loc=0, scale=1, size =(1,1))[0][0]*scale
               noise = data[col][idx]+random
               noises.append(noise)
           generate_data.append(noises)
       gen_df = pd.DataFrame(generate_data).T
       gen_df.columns = data.columns
       return gen_df

]: for idx in range(len(data)):
       new_df = noise9_data_generator(data[idx])
       new_df.to_csv('noise9 {}.csv'.format(data_name[idx]))

]: def noise18_data_generator(data):
       generate_data =[]
       for col in data.columns:
           scale = (data[col].max()-data[col].mean())/18
           noises = []
           for idx in range(len(data[col])):
               random = np.random.normal(loc=0, scale=1, size =(1,1))[0][0]*scale
               noise = data[col][idx]+random
               noises.append(noise)
           generate_data.append(noises)
       gen_df = pd.DataFrame(generate_data).T
       gen_df.columns = data.columns
       return gen_df

]: for idx in range(len(data)):
       new_df = noise18_data_generator(data[idx])
       new_df.to_csv('noise18 {}.csv'.format(data_name[idx]))

]: def noise27_data_generator(data):
       generate_data =[]
       for col in data.columns:
           scale = (data[col].max()-data[col].mean())/18
           noises = []
           for idx in range(len(data[col])):
               random = np.random.normal(loc=0, scale=1, size =(1,1))[0][0]*scale
               noise = data[col][idx]+random
               noises.append(noise)
           generate_data.append(noises)
       gen_df = pd.DataFrame(generate_data).T
       gen_df.columns = data.columns
       return gen_df

]: for idx in range(len(data)):
       new_df = noise27_data_generator(data[idx])
       new_df.to_csv('noise27 {}.csv'.format(data_name[idx]))
```

Train model

```
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor ="loss",
                                        mode ="min", patience = 5,
                                        restore_best_weights = True)
model=None
X_train=None
X_test=None
y_train=None
y_test=None

for idx in range(len(data)):
    print('--------------------{}------------------------------'.format(data_name[idx]))
    prepare_train(data[idx])
    history= model.fit(X_train, y_train,validation_data=(X_test, y_test),epochs=500,callbacks =[earlystopping])
    print(model.evaluate(X_test,y_test))

    #  "Accuracy"
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
 # "Loss"
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

Diabetes Dataset Experiments

Diabetes Data Sprit Train-Test

```
import numpy as np
import random

normal = []
diabet = []

infile = open("diabetes0.csv", "r")

for line in infile:
    x = line.split(',')
    if len(x) > 0:
        x = list(map(float, x))
        if x[8] == 1:
            diabet.append(x)
        else:
            normal.append(x)


infile.close()

prob = 0.2
trainset = []
testset = []

for k in range(len(diabet)):
    p = random.randint(0,10000)/10000.
    if p > 0.2:
        trainset.append(diabet[k][:8]+[diabet[k][8]])
    else:
        testset.append(diabet[k][:8]+[diabet[k][8]])


taken = [False]*len(normal)
setsize = 0
while setsize < len(diabet):
    k = random.randint(0,len(normal)-1)
    if not taken[k]:
        p = random.randint(0,10000)/10000.
        if p > 0.2:
            trainset.append(normal[k][:8]+[normal[k][8]])
        else:
            testset.append(normal[k][:8]+[normal[k][8]])
        setsize += 1
        taken[k] = True

outtrain = open('diabetes-all-train.csv', 'w')
for x in trainset:
    for i in range(8):
        outtrain.write(str(x[i])+',')
    outtrain.write(str(x[8])+'\n')
outtrain.close()

outtest = open('diabetes-all-test.csv', 'w')
for x in testset:
    for i in range(8):
        outtest.write(str(x[i])+',')
    outtest.write(str(x[8])+'\n')
outtest.close()
```

Diabetes Model Maker and Prepare Train

```
def model_maker():
    global model
    global noise_model
    model = Sequential()
    model.add(Dense(32,input_dim=8,activation='tanh',use_bias=True))
    model.add(Dense(1,activation='sigmoid',use_bias=True))
    model.compile(optimizer='adam',
              loss='binary_crossentropy', # Loss
              metrics=['accuracy'])
    model.optimizer.lr.assign(0.001)
```

```
def prepare_train(data):
    global model,X_train,X_test,y_train,y_test
    X_train, X_test, y_train, y_test = split_test(data.iloc[:,:-1],data.iloc[:,-1])
    X_train= X_train.to_numpy()
    y_train= y_train.to_numpy()
    X_test= X_test.to_numpy()
    y_test= y_test.to_numpy()
    print(y_test)


    model_maker()
```

Import Diabetes Data

```
diabetes = pd.read_csv('diabetes-all-train.csv')
diabetes_noise9 = pd.read_csv('noise9 diabetes dataset.csv')
diabetes_noise18 = pd.read_csv('noise18 diabetes dataset.csv')
diabetes_noise27 = pd.read_csv('noise27 diabetes dataset.csv')
```

```
data = [diabetes,diabetes_noise9,diabetes_noise18,diabetes_noise27]
data_name = ['diabetes dataset','diabetes noise9 dataset','diabetes noise18 dataset','diabetes noise27 dataset']
```

Breast Cancer Dataset Experiments

Breast Cancer Data Sprit Train-Test

```
normal = []
Bcancer = []

infile = open("breast-cancer1.csv", "r")

for line in infile:
    x = line.split(',')
    if len(x) > 0:
        x = list(map(float, x))
        if x[30] == 1:
            Bcancer.append(x)
        else:
            normal.append(x)


infile.close()

prob = 0.3
trainset = []
testset = []

for k in range(len(Bcancer)):
    p = random.randint(0,10000)/10000.
    if p > 0.3:
        trainset.append(Bcancer[k][:30]+[Bcancer[k][30]])
    else:
        testset.append(Bcancer[k][:30]+[Bcancer[k][30]])
```

```
taken = [False]*len(normal)
setsize = 0
while setsize < len(Bcancer):
    k = random.randint(0,len(normal)-1)
    if not taken[k]:
        p = random.randint(0,10000)/10000.
        if p > 0.2:
            trainset.append(normal[k][:30]+[normal[k][30]])
        else:
            testset.append(normal[k][:30]+[normal[k][30]])
        setsize += 1
        taken[k] = True

outtrain = open('BreastCancer-all-train.csv', 'w')
for x in trainset:
    for i in range(30):
        outtrain.write(str(x[i])+',')
    outtrain.write(str(x[30])+'\n')
outtrain.close()

outtest = open('BreastCancer-all-test.csv', 'w')
for x in testset:
    for i in range(30):
        outtest.write(str(x[i])+',')
    outtest.write(str(x[30])+'\n')
outtest.close()
```

Breast-Cancer Data Model Maker and prepare Train

```
def model_maker():
    global model
    global noise_model
    model = Sequential()
    model.add(Dense(32,input_dim=30,activation='tanh',use_bias=True))
#   model.add(Dense(10,activation='tanh',use_bias=True))
    model.add(Dense(1,activation='sigmoid',use_bias=True))
    model.compile(optimizer='adam',
              loss='binary_crossentropy', # Loss
              metrics=['accuracy'])
    model.optimizer.lr.assign(0.001)
```

```
def prepare_train(data):
    global model,X_train,X_test,y_train,y_test
    X_train, X_test, y_train, y_test = split_test(data.iloc[:,:-1],data.iloc[:,-1])
    X_train= X_train.to_numpy()
    y_train= y_train.to_numpy()
    X_test= X_test.to_numpy()
    y_test= y_test.to_numpy()
    print(y_test)


    model_maker()
```

Breast-Cancer Import Data

```
breast_cancer = pd.read_csv('BreastCancer-all-train.csv')
breast_cancer_noise9 = pd.read_csv('noise9 Breast Cancer Dataset.csv')
breast_cancer_noise18 = pd.read_csv('noise18 Breast Cancer Dataset.csv')
breast_cancer_noise27 = pd.read_csv('noise27 Breast Cancer Dataset.csv')
```

```
data = [breast_cancer,breast_cancer_noise9,breast_cancer_noise18,breast_cancer_noise27 ]
data_name = ['Breast Cancer Dataset','Breast Cancer noise9 Dataset','Breast Cancer noise18 Dataset','Breast Cancer noise27 Datas
```

Smoking Experiments

Smoking Data Sprit Train-Test

```
import numpy as np
import random

no = []
yes = []

infile = open("Smoking0.csv", "r")

for line in infile:
    x = line.split(',')
    if len(x) > 0:
        x = list(map(float, x))
        if x[23] == 1:
            yes.append(x)
        else:
            no.append(x)


infile.close()

prob = 0.2
trainset = []
testset = []

for k in range(len(yes)):
    p = random.randint(0,10000)/10000.
    if p > 0.2:
        trainset.append(yes[k][:23]+[yes[k][23]])
    else:
        testset.append(yes[k][:23]+[yes[k][23]])
```

```
taken = [False]*len(no)
setsize = 0
while setsize < len(yes):
    k = random.randint(0,len(no)-1)
    if not taken[k]:
        p = random.randint(0,10000)/10000.
        if p > 0.2:
            trainset.append(no[k][:23]+[no[k][23]])
        else:
            testset.append(no[k][:23]+[no[k][23]])
        setsize += 1
        taken[k] = True

outtrain = open('smoking-all-train.csv', 'w')
for x in trainset:
    for i in range(23):
        outtrain.write(str(x[i])+',')
    outtrain.write(str(x[23])+'\n')
outtrain.close()

outtest = open('smoking-all-test.csv', 'w')
for x in testset:
    for i in range(23):
        outtest.write(str(x[i])+',')
    outtest.write(str(x[23])+'\n')
outtest.close()
```

Smoking Data Model Maker and Prepare Train

```
def model_maker():
    global model
    global noise_model
    model = Sequential()
    model.add(Dense(10,input_dim=18,activation='tanh',use_bias=True))
    model.add(Dense(10,activation='tanh',use_bias=True))
    model.add(Dense(1,activation='sigmoid',use_bias=True))
    model.compile(optimizer='adam',
              loss='binary_crossentropy', # Loss
              metrics=['accuracy'])
    model.optimizer.lr.assign(0.001)
```

```
def prepare_train(data):
    global model,X_train,X_test,y_train,y_test
    X_train, X_test, y_train, y_test = split_test(data.iloc[:,:-1],data.iloc[:,-1])
    X_train= X_train.to_numpy()
    y_train= y_train.to_numpy()
    X_test= X_test.to_numpy()
    y_test= y_test.to_numpy()
    print(y_test)


    model_maker()
```

Smoking Import Data

```
smoking = pd.read_csv('Smoking-all-train.csv')
smoking_noise9 = pd.read_csv('noise9 smoking Dataset.csv')
smoking_noise18 = pd.read_csv('noise18 smoking Dataset.csv')
smoking_noise27 = pd.read_csv('noise27 smoking Dataset.csv')
```

```
data = [smoking,smoking_noise9,smoking_noise18,smoking_noise27  ]
data_name = ['smoking Dataset','smoking noise9 Datsets','smoking noise18 Datsets','smoking noise27 Datsets']
```

Wine Quality Experiment

Wine Quality Sprit Train-Test

```
import numpy as np
import random

good = []
bad = []

infile = open("winequality-red1.csv", "r")

for line in infile:
    x = line.split(',')
    if len(x) > 0:
        x = list(map(float, x))
        if x[12] == 1:
            good.append(x)
        else:
            bad.append(x)


infile.close()

prob = 0.2
trainset = []
testset = []

for k in range(len(yes)):
    p = random.randint(0,10000)/10000.
    if p > 0.2:
        trainset.append(good[k][:12]+[good[k][12]])
    else:
        testset.append(good[k][:12]+[good[k][12]])
```

```
taken = [False]*len(no)
setsize = 0
while setsize < len(yes):
    k = random.randint(0,len(no)-1)
    if not taken[k]:
        p = random.randint(0,10000)/10000.
        if p > 0.2:
            trainset.append(bad[k][:12]+[no[k][12]])
        else:
            testset.append(no[k][:12]+[no[k][12]])
        setsize += 1
        taken[k] = True

outtrain = open('winequality-red-all-train.csv', 'w')
for x in trainset:
    for i in range(12):
        outtrain.write(str(x[i])+',')
    outtrain.write(str(x[12])+'\n')
outtrain.close()

outtest = open('winequality-red-all-test.csv', 'w')
for x in testset:
    for i in range(12):
        outtest.write(str(x[i])+',')
    outtest.write(str(x[12])+'\n')
outtest.close()
```

Wine Quality Data Model Maker and Prepare Train

```
from sklearn.model_selection import train_test_split

def split_test(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
    return X_train, X_test, y_train, y_test
```

```
def model_maker():
    global model
    global noise_model
    model = Sequential()
    model.add(Dense(10,input_dim=12,activation='tanh',use_bias=True))
    model.add(Dense(1,activation='sigmoid',use_bias=True))
    model.compile(optimizer='adam',
              loss='binary_crossentropy', # Loss
              metrics=['accuracy'])
    model.optimizer.lr.assign(0.001)
```

```
def prepare_train(data):
    global model,X_train,X_test,y_train,y_test
    X_train, X_test, y_train, y_test = split_test(data.iloc[:,:-1],data.iloc[:,-1])
    X_train= X_train.to_numpy()
    y_train= y_train.to_numpy()
    X_test= X_test.to_numpy()
    y_test= y_test.to_numpy()
    print(y_test)


    model_maker()
```

Wine Quality Import Data

```
wine_quality = pd.read_csv('winequality-red-all-train.csv')
wine_quality_noise9 = pd.read_csv('noise9 Wine-Quality Dataset.csv')
wine_quality_noise18 = pd.read_csv('noise18 Wine-Quality Dataset.csv')
wine_quality_noise27 = pd.read_csv('noise27 Wine-Quality Dataset.csv')
```

```
data = [wine_quality,wine_quality_noise9,wine_quality_noise18,wine_quality_noise27]
data_name = ['Wine-Quality Dataset','Wine-Quality noise9 Dataset','Wine-Quality noise18 Dataset','Wine-Quality noise27 Dataset']
```

Heart Dataset Experiment

Heart Disease Sprit Train-Test

```
import numpy as np
import random

normal = []
heart = []

infile = open("Heart1.csv", "r")

for line in infile:
    x = line.split(',')
    if len(x) > 0:
        x = list(map(float, x))
        if x[13] == 1:
            heart.append(x)
        else:
            normal.append(x)


infile.close()

prob = 0.2
trainset = []
testset = []

for k in range(len(normal)):
    p = random.randint(0,10000)/10000.
    if p > 0.2:
        trainset.append(normal[k][:13]+[normal[k][13]])
    else:
        testset.append(normal[k][:13]+[normal[k][13]])


taken = [False]*len(heart)
setsize = 0
while setsize < len(normal):
    k = random.randint(0,len(heart)-1)
    if not taken[k]:
        p = random.randint(0,10000)/10000.
        if p > 0.2:
            trainset.append(heart[k][:13]+[heart[k][13]])
        else:
            testset.append(heart[k][:13]+[heart[k][13]])
        setsize += 1
        taken[k] = True

outtrain = open('heart-all-train.csv', 'w')
for x in trainset:
    for i in range(13):
        outtrain.write(str(x[i])+',')
    outtrain.write(str(x[13])+'\n')
outtrain.close()

outtest = open('heart-all-test.csv', 'w')
for x in testset:
    for i in range(13):
        outtest.write(str(x[i])+',')
    outtest.write(str(x[13])+'\n')
outtest.close()
```

Heart Disease Data Model Maker and Prepare Train

```
def model_maker():
    global model
    global noise_model
    model = Sequential()
    model.add(Dense(32,input_dim=13,activation='tanh',use_bias=True))
#   model.add(Dense(10,activation='tanh',use_bias=True))
    model.add(Dense(1,activation='sigmoid',use_bias=True))
    model.compile(optimizer='adam',
                  loss='binary_crossentropy', # Loss
                  metrics=['accuracy'])
    model.optimizer.lr.assign(0.001)
```

```
def prepare_train(data):
    global model,X_train,X_test,y_train,y_test
    X_train, X_test, y_train, y_test = split_test(data.iloc[:,:-1],data.iloc[:,-1])
    X_train= X_train.to_numpy()
    y_train= y_train.to_numpy()
    X_test= X_test.to_numpy()
    y_test= y_test.to_numpy()
    print(y_test)


    model_maker()
```

Heart Disease Import Data

```
heart = pd.read_csv('heart-all-train.csv')
heart_noise9 = pd.read_csv('noise9 Heart Dataset.csv')
heart_noise18 = pd.read_csv('noise18 Heart Dataset.csv')
heart_noise27 = pd.read_csv('noise27 Heart Dataset.csv')
```

```
data = [heart,heart_noise9,heart_noise18,heart_noise27]
data_name = ['Heart Dataset','Heart noise9 Dataset','Heart noise18 Dataset','Heart noise27 Dataset']
```

Result

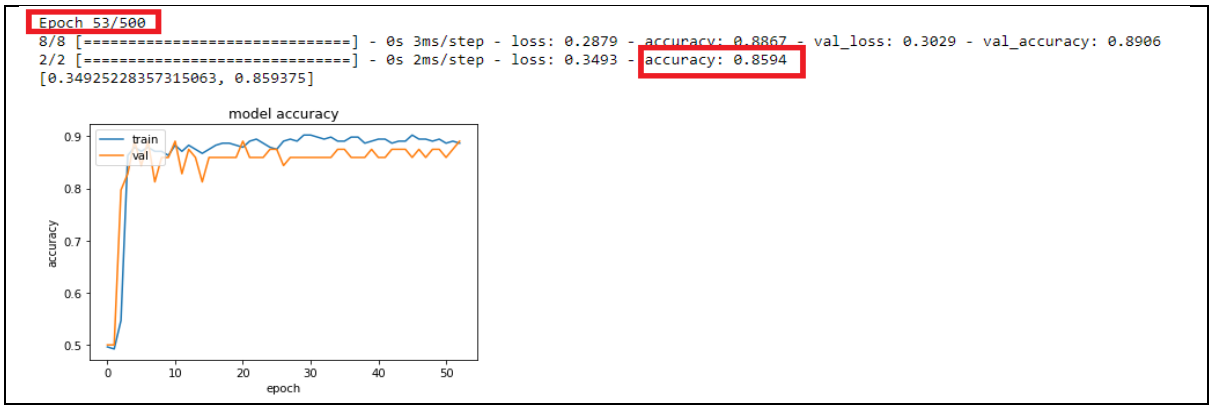
For the result, we intend to train the model for each experiment for several times to ensure that the results of each experiment are invariable.

The accuracy value and accuracy graph of each dataset are shown below.

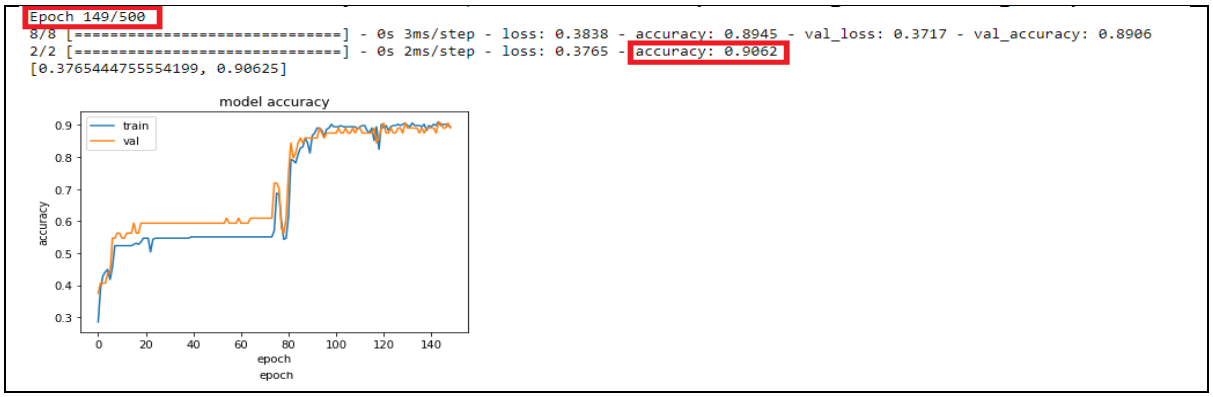
Breast Cancer Experiment Result

Breast Cancer Original Data Result

1st round early stop at epoch 53, accuracy 0.8594

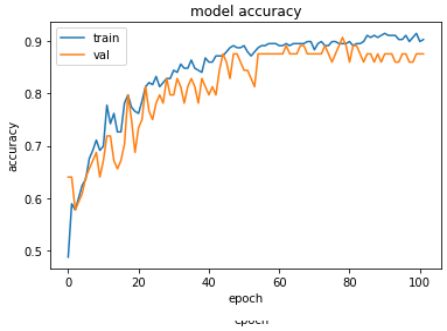


2nd round early stop at epoch 149, accuracy 0.9062



3rd round early stop at epoch 102, accuracy0.8750

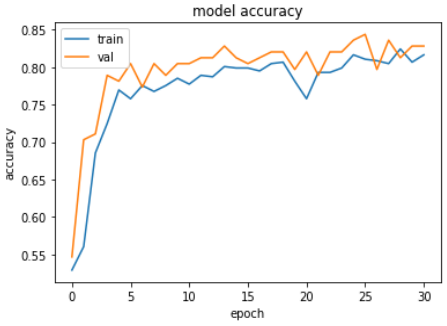
```
Epoch 102/500
8/8 [==============================] - 0s 3ms/step - loss: 0.2729 - accuracy: 0.9023 - val_loss: 0.2981 - val_accuracy: 0.8750
2/2 [==============================] - 0s 2ms/step - loss: 0.2957 - accuracy: 0.8750
[0.29571783542633057, 0.875]
```



Breast Cancer Noise (1/9) Data

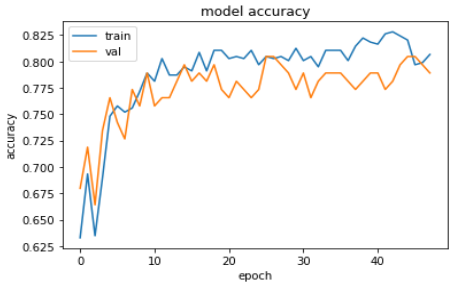
1st round early stop at epoch 31, accuracy 0.8438

```
Epoch 31/500
16/16 [==============================] - 0s 2ms/step - loss: 0.4374 - accuracy: 0.8164 - val_loss: 0.4563 - val_accuracy: 0.828
1
4/4 [==============================] - 0s 1ms/step - loss: 0.4406 - accuracy: 0.8438
[0.4405670464038849, 0.84375]
```



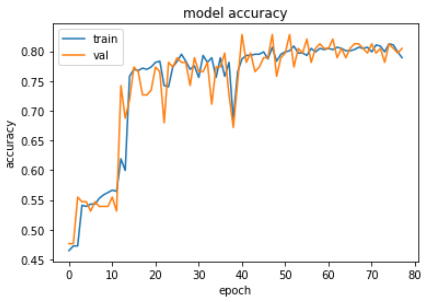
2nd round early stop at epoch 48, accuracy 0.7812

```
Epoch 48/500
16/16 [==============================] - 0s 2ms/step - loss: 0.4455 - accuracy: 0.8066 - val_loss: 0.4620 - val_accuracy: 0.789
1
4/4 [==============================] - 0s 1ms/step - loss: 0.4546 - accuracy: 0.7812
[0.45460543036460876, 0.78125]
```



3rd round early stop at epoch 78, accuracy0.8047

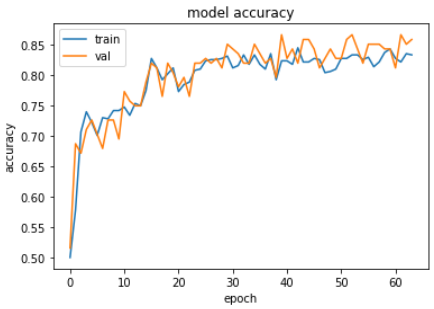
```
Epoch 78/500
16/16 [==============================] - 0s 2ms/step - loss: 0.4512 - accuracy: 0.7891 - val_loss: 0.5082 - val_accuracy: 0.804
7
4/4 [==============================] - 0s 1ms/step - loss: 0.5087 - accuracy: 0.8047
[0.508728563785553, 0.8046875]
```



Breast Cancer Noise (1/18) Data Result

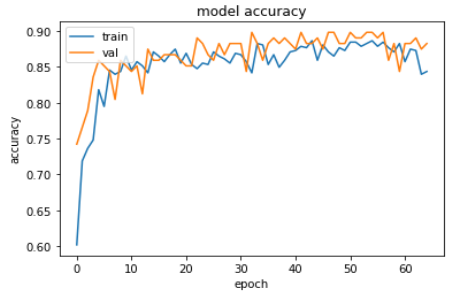
1st round early stop at epoch 64, accuracy 0.8438

```
Epoch 64/500
16/16 [==============================] - 0s 2ms/step - loss: 0.4125 - accuracy: 0.8340 - val_loss: 0.3642 - val_accuracy: 0.859
4
4/4 [==============================] - 0s 997us/step - loss: 0.3691 - accuracy: 0.8438
[0.3690894842147827, 0.84375]
```



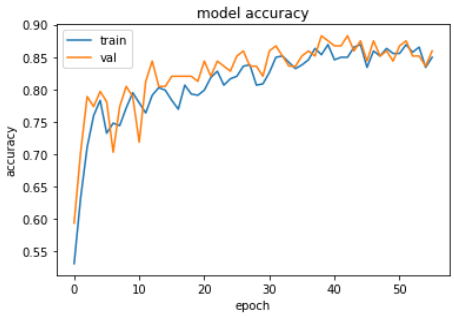
2nd round early stop at epoch 65, accuracy 0.8438

```
Epoch 65/500
16/16 [==============================] - 0s 2ms/step - loss: 0.3605 - accuracy: 0.8438 - val_loss: 0.3438 - val_accuracy: 0.882
8
4/4 [==============================] - 0s 1ms/step - loss: 0.3371 - accuracy: 0.8438
[0.33709776401519775, 0.84375]
```



3rd round early stop at epoch 56, accuracy 0.8672

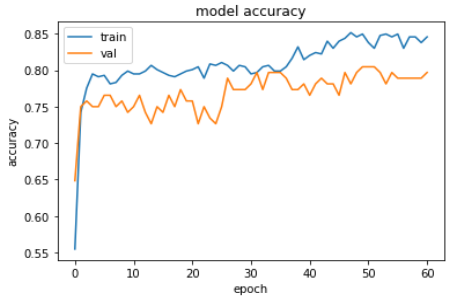
```
Epoch 56/500
16/16 [==============================] - 0s 2ms/step - loss: 0.3525 - accuracy: 0.8496 - val_loss: 0.3637 - val_accuracy: 0.859
4
4/4 [==============================] - 0s 1ms/step - loss: 0.3492 - accuracy: 0.8672
[0.3491585850715637, 0.8671875]
```



Breast Cancer Noise (1/27) Data Result

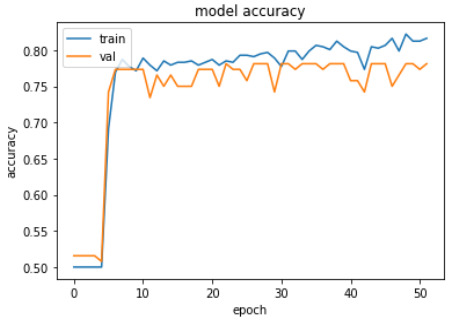
1st round early stop at epoch 61 accuracy 0.7891

```
Epoch 61/500
16/16 [==============================] - 0s 2ms/step - loss: 0.3934 - accuracy: 0.8457 - val_loss: 0.4714 - val_accuracy: 0.796
9
4/4 [==============================] - 0s 997us/step - loss: 0.4688 - accuracy: 0.7891
[0.46878331899642944, 0.7890625]
```



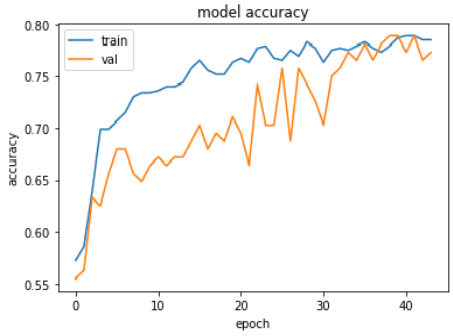
2nd round early stop at epoch 52, accuracy 0.7500

```
Epoch 52/500
16/16 [==============================] - 0s 2ms/step - loss: 0.4665 - accuracy: 0.8164 - val_loss: 0.5410 - val_accuracy: 0.781
2
4/4 [==============================] - 0s 998us/step - loss: 0.5832 - accuracy: 0.7500
[0.5831547379493713, 0.75]
```



3rd round early stop at epoch 44, accuracy 0.7891

```
Epoch 44/500
16/16 [==============================] - 0s 2ms/step - loss: 0.4889 - accuracy: 0.7852 - val_loss: 0.5220 - val_accuracy: 0.773
4
4/4 [==============================] - 0s 997us/step - loss: 0.5251 - accuracy: 0.7891
[0.5250576734542847, 0.7890625]
```

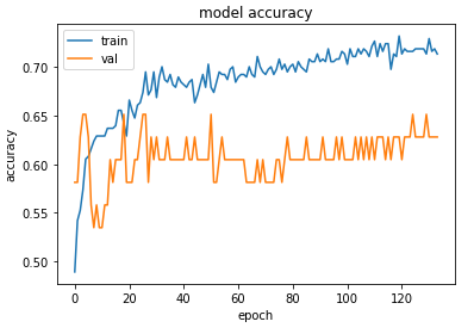


Diabetes Experiment Result

Diabetes Original Data Result

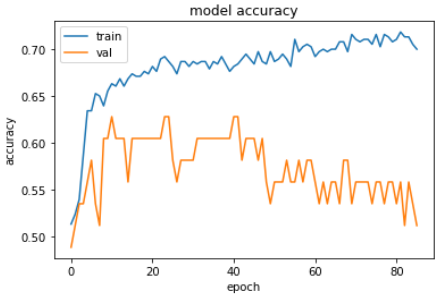
1st round early stop at epoch 134, accuracy 0.6279

```
Epoch 134/500
12/12 [==============================] - 0s 2ms/step - loss: 0.5446 - accuracy: 0.7132 - val_loss: 0.6671 - val_accuracy: 0.627
9
2/2 [==============================] - 0s 998us/step - loss: 0.6683 - accuracy: 0.6279
[0.668312668800354, 0.6279069781303406]
```



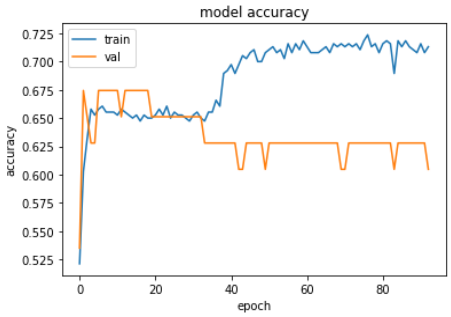
2nd round early stop at epoch 86, accuracy 0.5349

```
Epoch 86/500
12/12 [==============================] - 0s 3ms/step - loss: 0.5545 - accuracy: 0.7000 - val_loss: 0.8010 - val_accuracy: 0.511
6
2/2 [==============================] - 0s 998us/step - loss: 0.7932 - accuracy: 0.5349
[0.7932260036468506, 0.5348837375640869]
```



3rd round early stop at epoch 93, accuracy 0.6279

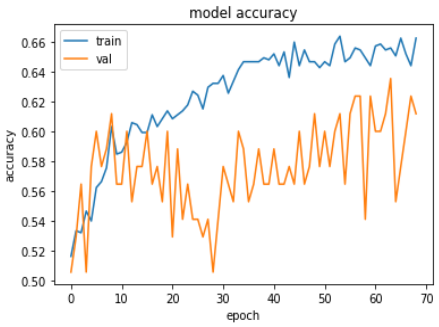
```
Epoch 93/500
12/12 [==============================] - 0s 3ms/step - loss: 0.5532 - accuracy: 0.7132 - val_loss: 0.6764 - val_accuracy: 0.604
7
2/2 [==============================] - 0s 2ms/step - loss: 0.6753 - accuracy: 0.6279
[0.6752756834030151, 0.6279069781303406]
```



Diabetes Noise (1/9) Data Result

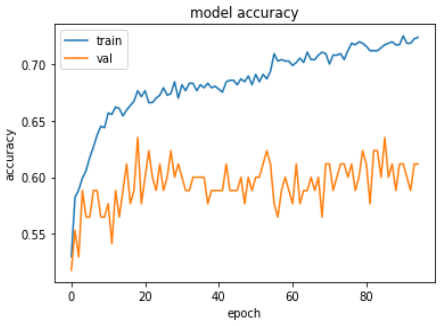
1st round early stop at epoch 69, accuracy 0.6353

```
Epoch 69/500
24/24 [==============================] - 0s 2ms/step - loss: 0.6219 - accuracy: 0.6623 - val_loss: 0.6657 - val_accuracy: 0.611
8
3/3 [==============================] - 0s 1ms/step - loss: 0.6569 - accuracy: 0.6353
[0.6569207310676575, 0.6352941393852234]
```



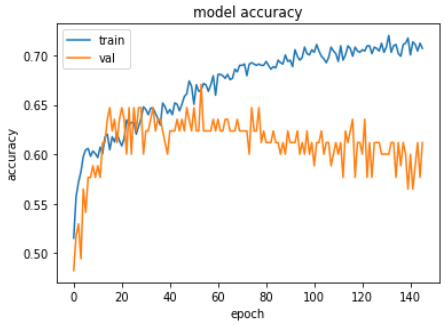
2nd round early stop at epoch 95, accuracy 0.6118

```
Epoch 95/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5578 - accuracy: 0.7240 - val_loss: 0.6824 - val_accuracy: 0.611
8
3/3 [==============================] - 0s 1ms/step - loss: 0.6938 - accuracy: 0.6118
[0.6938436627388, 0.6117647290229797]
```



3rd round early stop at epoch 146, accuracy 0.6000

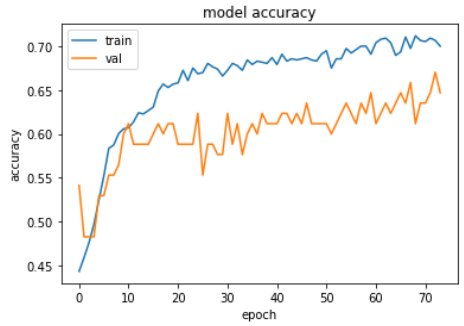
```
Epoch 146/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5586 - accuracy: 0.7070 - val_loss: 0.6947 - val_accuracy: 0.611
8
3/3 [==============================] - 0s 1ms/step - loss: 0.6866 - accuracy: 0.6000
[0.6866051554679871, 0.6000000238418579]
```



Diabetes Noise (1/18) Data Result

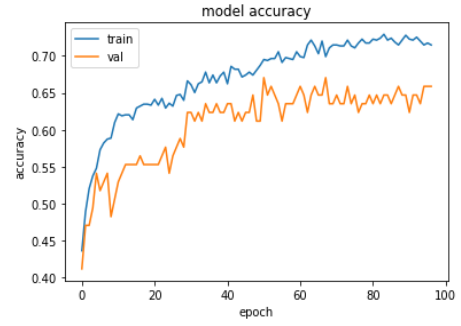
1st round early stop at epoch 74, accuracy 0.6118

```
Epoch 74/500
24/24 [==============================] - 0s 1ms/step - loss: 0.5698 - accuracy: 0.7004 - val_loss: 0.6767 - val_accuracy: 0.647
1
3/3 [==============================] - 0s 1ms/step - loss: 0.6845 - accuracy: 0.6118
[0.684471070766449, 0.6117647290229797]
```



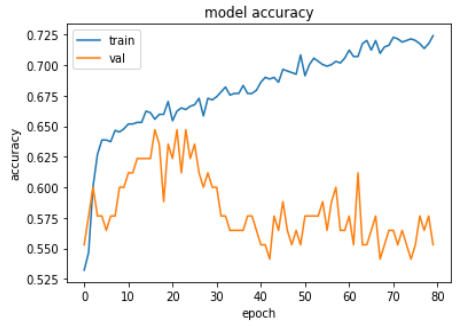
2nd round early stop at epoch 97, accuracy 0.6471

```
Epoch 97/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5583 - accuracy: 0.7148 - val_loss: 0.6776 - val_accuracy: 0.658
8
3/3 [==============================] - 0s 1ms/step - loss: 0.6855 - accuracy: 0.6471
[0.6855429410934448, 0.6470588445663452]
```



3rd round early stop at epoch 80, accuracy 0.5412

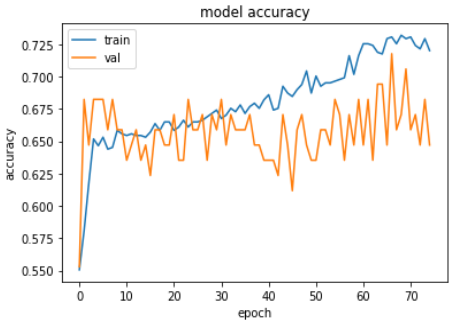
```
Epoch 80/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5588 - accuracy: 0.7240 - val_loss: 0.7172 - val_accuracy: 0.552
9
3/3 [==============================] - 0s 1ms/step - loss: 0.7153 - accuracy: 0.5412
[0.7153028845787048, 0.5411764979362488]
```



Diabetes Noise (1/27) Data Result

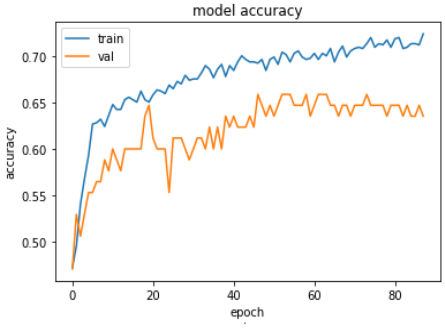
1st round early stop at epoch 75, accuracy 0.7059

```
Epoch 75/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5533 - accuracy: 0.7201 - val_loss: 0.6228 - val_accuracy: 0.647
1
3/3 [==============================] - 0s 2ms/step - loss: 0.6105 - accuracy: 0.7059
[0.6104506850242615, 0.7058823704719543]
```



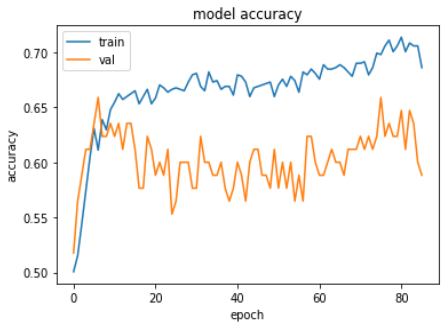
2nd round early stop at epoch 88, accuracy 0.6353

```
Epoch 88/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5532 - accuracy: 0.7240 - val_loss: 0.6430 - val_accuracy: 0.635
3
3/3 [==============================] - 0s 2ms/step - loss: 0.6481 - accuracy: 0.6353
[0.6480743288993835, 0.6352941393852234]
```



3rd round early stop at epoch 86, accuracy 0.6471

```
Epoch 86/500
24/24 [==============================] - 0s 2ms/step - loss: 0.5663 - accuracy: 0.6859 - val_loss: 0.7199 - val_accuracy: 0.588
2
3/3 [==============================] - 0s 998us/step - loss: 0.7130 - accuracy: 0.6471
[0.7130467295646667, 0.6470588445663452]
```

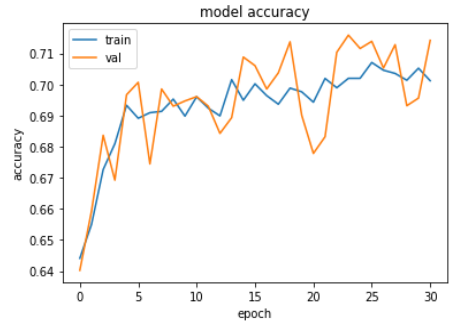


Smoking Experiment Result

Smoking Original Data Result

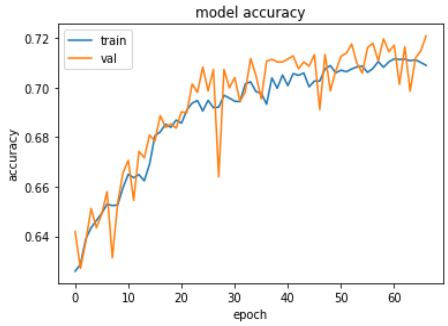
1st round early stop at epoch 31, accuracy 0.7138

```
Epoch 31/500
816/816 [==============================] - 1s 915us/step - loss: 0.5673 - accuracy: 0.7012 - val_loss: 0.5656 - val_accuracy: 
0.7142
204/204 [==============================] - 0s 599us/step - loss: 0.5661 - accuracy: 0.7138
[0.5661206841468811, 0.7138475775718689]
```



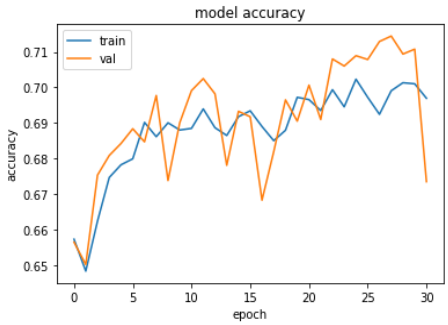
2nd round early stop at epoch 67, accuracy 0.7013

```
Epoch 67/500
816/816 [==============================] - 1s 867us/step - loss: 0.5622 - accuracy: 0.7090 - val_loss: 0.5612 - val_accuracy: 0
0.7209
204/204 [==============================] - 0s 629us/step - loss: 0.5584 - accuracy: 0.7013
[0.5583899617195129, 0.7012727856636047]
```



3rd round early stop at epoch 31, accuracy 0.7079

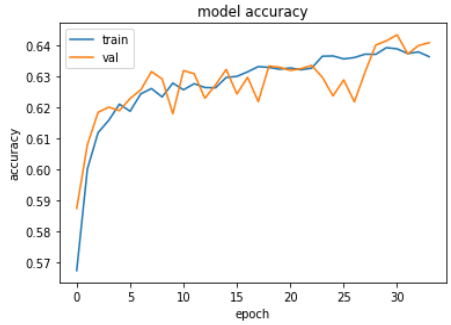
```
Epoch 31/500
816/816 [==============================] - 1s 1ms/step - loss: 0.5807 - accuracy: 0.6970 - val_loss: 0.5818 - val_accuracy: 0.6
735
204/204 [==============================] - 0s 722us/step - loss: 0.5728 - accuracy: 0.7079
[0.5727924108505249, 0.707866907119751]
```



Smoking Noise (1/9) Data Result

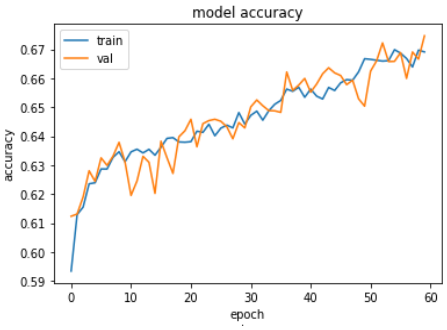
1st round early stop at epoch 34, accuracy 0.6400

```
Epoch 34/500
1631/1631 [==============================] - 1s 898us/step - loss: 0.6302 - accuracy: 0.6363 - val_loss: 0.6300 - val_accuracy:
0.6408
408/408 [==============================] - 0s 620us/step - loss: 0.6305 - accuracy: 0.6400
[0.6305201053619385, 0.6400092244148254]
```



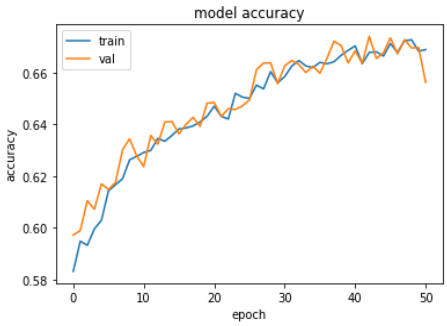
2nd round early stop at epoch 60, accuracy 0.6658

```
Epoch 60/500
1631/1631 [==============================] - 1s 888us/step - loss: 0.6105 - accuracy: 0.6691 - val_loss: 0.6116 - val_accuracy: :
0.6747
408/408 [==============================] - 0s 750us/step - loss: 0.6051 - accuracy: 0.6658
[0.6051488518714905, 0.6658487915992737]
```



3rd round early stop at epoch 51, accuracy 0.6734

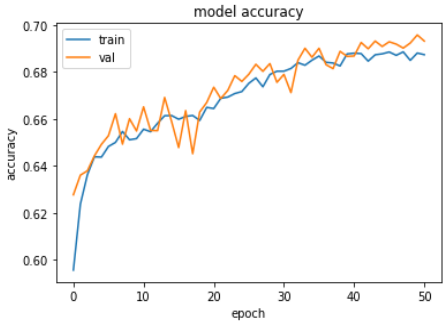
```
Epoch 51/500
1631/1631 [==============================] - 2s 986us/step - loss: 0.6090 - accuracy: 0.6689 - val_loss: 0.6162 - val_accuracy:
0.6563
408/408 [==============================] - 0s 723us/step - loss: 0.6076 - accuracy: 0.6734
[0.6075748205184937, 0.6734396815299988]
```



Smoking Noise (1/18) Data Result

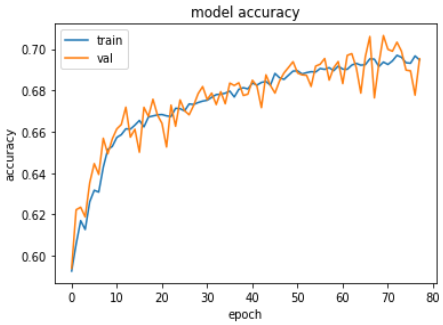
1st round early stop at epoch 51, accuracy 0.6928

```
Epoch 51/500
1631/1631 [==============================] - 1s 873us/step - loss: 0.5912 - accuracy: 0.6873 - val_loss: 0.5845 - val_accuracy:
0.6931
408/408 [==============================] - 0s 588us/step - loss: 0.5853 - accuracy: 0.6928
[0.585300862789154, 0.6927618384361267]
```



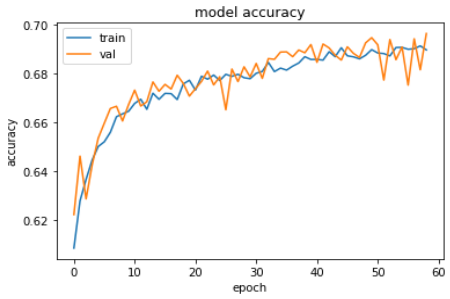
2nd round early stop at epoch 78, accuracy 0.7033

```
Epoch 78/500
1631/1631 [==============================] - 2s 981us/step - loss: 0.5809 - accuracy: 0.6948 - val_loss: 0.5768 - val_accuracy: :
0.6953
408/408 [==============================] - 0s 870us/step - loss: 0.5780 - accuracy: 0.7033
[0.5779806971549988, 0.7033430337905884]
```



3rd round early stop at epoch 59, accuracy 0.6587

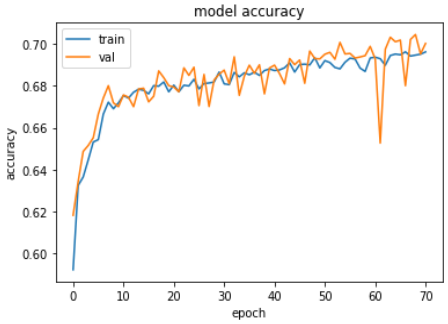
```
Epoch 59/500
1631/1631 [==============================] - 1s 904us/step - loss: 0.5880 - accuracy: 0.6897 - val_loss: 0.5807 - val_accuracy:
0.6964
408/408 [==============================] - 0s 632us/step - loss: 0.5901 - accuracy: 0.6857
[0.5901364684104919, 0.685707688331604]
```



Smoking Noise (1/27) Data Result

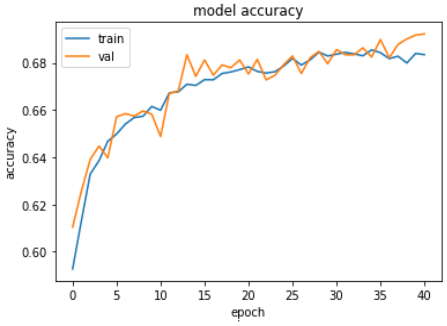
1st round early stop at epoch 71, accuracy 0.7019

```
Epoch 71/500
1631/1631 [==============================] - 1s 893us/step - loss: 0.5834 - accuracy: 0.6963 - val_loss: 0.5799 - val_accuracy: ':
0.7003
408/408 [==============================] - 0s 698us/step - loss: 0.5744 - accuracy: 0.7019
[0.574417233467102, 0.7018862366676331]
```



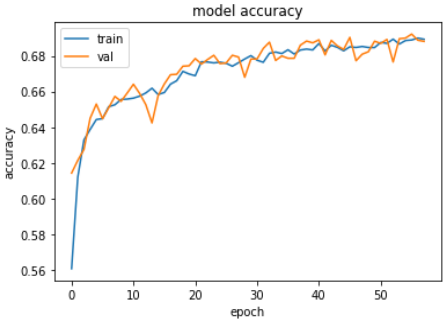
2nd round early stop at epoch 41, accuracy 0.6899

```
Epoch 41/500
1631/1631 [==============================] - 1s 882us/step - loss: 0.5954 - accuracy: 0.6835 - val_loss: 0.5845 - val_accuracy: :
0.6923
408/408 [==============================] - 0s 642us/step - loss: 0.5899 - accuracy: 0.6899
[0.5898662805557251, 0.6899248361587524]
```



3rd round early stop at epoch 58, accuracy 0.6766

```
Epoch 58/500
1631/1631 [==============================] - 1s 873us/step - loss: 0.5892 - accuracy: 0.6893 - val_loss: 0.5881 - val_accuracy:
0.6882
408/408 [==============================] - 0s 613us/step - loss: 0.5904 - accuracy: 0.6766
[0.5903554558753967, 0.6765833497047424]
```

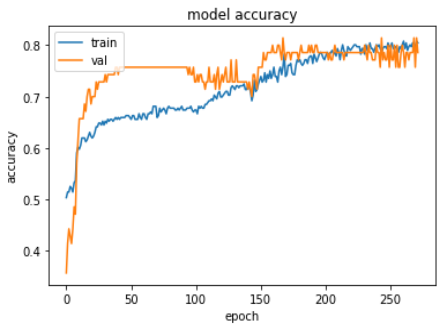


Wine Quality Experiment Result

Wine Quality Original Data Result

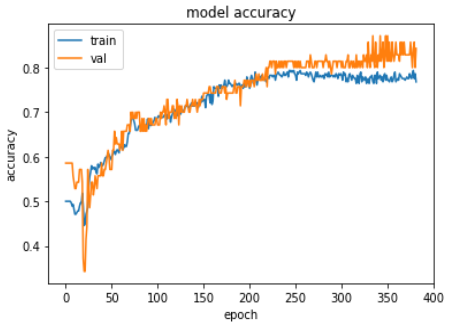
1st round early stop at epoch 272, accuracy 0.7857

```
Epoch 272/500
9/9 [==============================] - 0s 4ms/step - loss: 0.4639 - accuracy: 0.8043 - val_loss: 0.4961 - val_accuracy: 0.7857
3/3 [==============================] - 0s 1ms/step - loss: 0.4907 - accuracy: 0.7857
[0.49072641134262085, 0.7857142686843872]
```



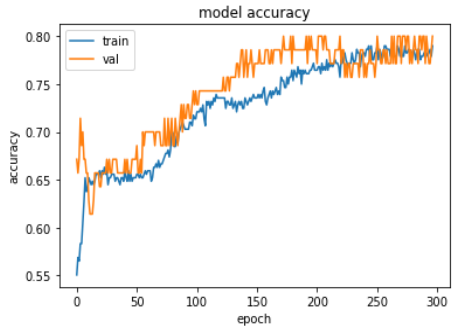
2nd round early stop at epoch 382, accuracy 0.8286

```
Epoch 382/500
9/9 [==============================] - 0s 3ms/step - loss: 0.4668 - accuracy: 0.7681 - val_loss: 0.4642 - val_accuracy: 0.8429
3/3 [==============================] - 0s 1ms/step - loss: 0.4768 - accuracy: 0.8286
[0.4767661988735199, 0.8285714387893677]
```



3rd round early stop at epoch 297, accuracy 0.8000

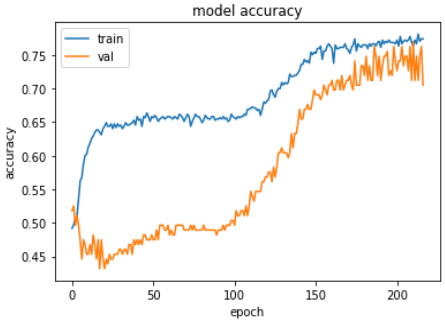
```
Epoch 297/500
9/9 [==============================] - 0s 3ms/step - loss: 0.4828 - accuracy: 0.7899 - val_loss: 0.5298 - val_accuracy: 0.8000
3/3 [==============================] - 0s 1ms/step - loss: 0.5271 - accuracy: 0.8000
[0.5270569920539856, 0.800000011920929]
```



Wine Quality Noise (1/9) Data Result

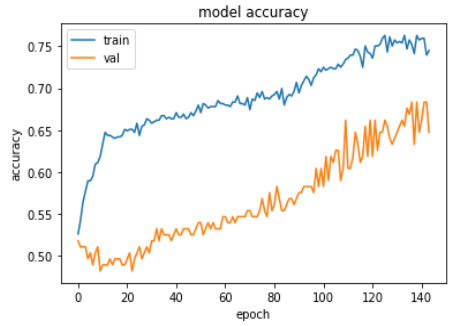
1st round early stop at epoch 217, accuracy 0.7122

```
Epoch 217/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4659 - accuracy: 0.7740 - val_loss: 0.5882 - val_accuracy: 0.705
0
5/5 [==============================] - 0s 998us/step - loss: 0.5780 - accuracy: 0.7122
[0.5780197978019714, 0.7122302055358887]
```



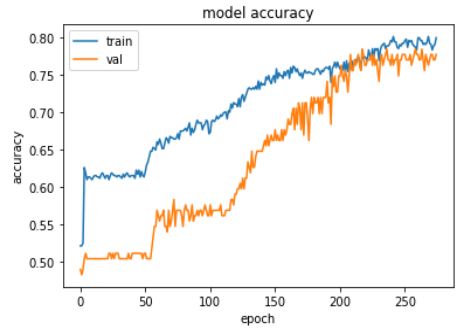
2nd round early stop at epoch 144, accuracy 0.6835

```
Epoch 144/500
18/18 [==============================] - 0s 2ms/step - loss: 0.5007 - accuracy: 0.7450 - val_loss: 0.6083 - val_accuracy: 0.647
5
5/5 [==============================] - 0s 997us/step - loss: 0.6092 - accuracy: 0.6835
[0.6092318892478943, 0.6834532618522644]
```



3rd round early stop at epoch 275, 0.7698

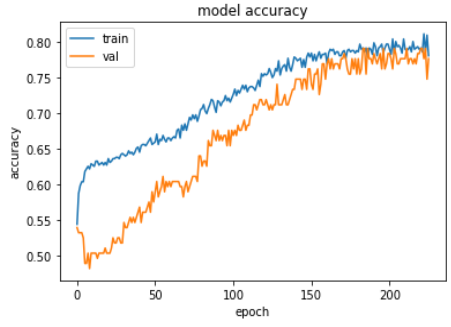
```
Epoch 275/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4548 - accuracy: 0.7993 - val_loss: 0.5498 - val_accuracy: 0.777
0
5/5 [==============================] - 0s 997us/step - loss: 0.5419 - accuracy: 0.7698
[0.5418552160263062, 0.769784152507782]
```



Wine Quality Noise (1/18) Data Result

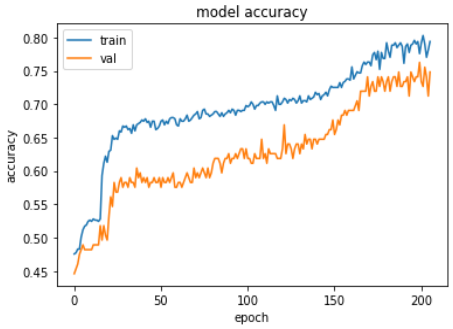
1st round early stop at epoch 226, accuracy 0.7914

```
Epoch 226/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4536 - accuracy: 0.7812 - val_loss: 0.5063 - val_accuracy: 0.777
0
5/5 [==============================] - 0s 998us/step - loss: 0.4996 - accuracy: 0.7914
[0.4996003210544586, 0.7913669347763062]
```



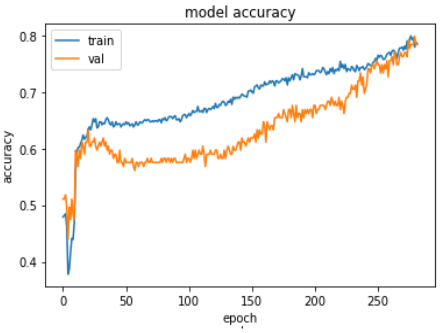
2nd round early stop at epoch 206, accuracy 0.7338

```
Epoch 206/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4681 - accuracy: 0.7939 - val_loss: 0.5383 - val_accuracy: 0.748
2
5/5 [==============================] - 0s 1ms/step - loss: 0.5473 - accuracy: 0.7338
[0.547322690486908, 0.7338129281997681]
```



3rd round early stop at epoch 282, accuracy 0.7842

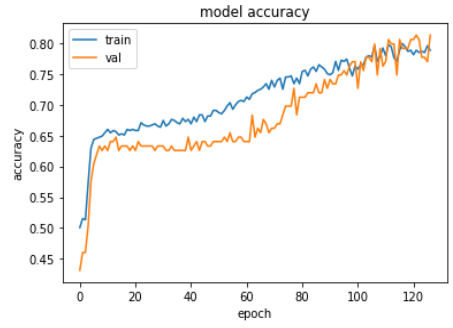
```
Epoch 282/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4642 - accuracy: 0.7848 - val_loss: 0.5273 - val_accuracy: 0.784
2
5/5 [==============================] - 0s 992us/step - loss: 0.5319 - accuracy: 0.7842
[0.5319104790687561, 0.7841726541519165]
```



Wine Quality Noise (1/27) Data Result

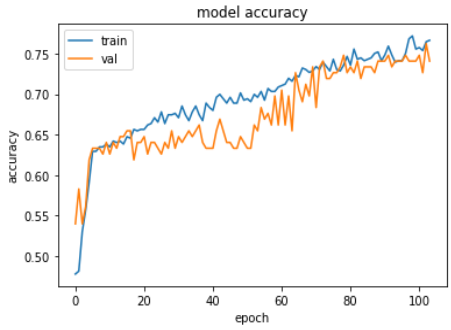
1st round early stop at epoch 127, accuracy 0.8129

```
Epoch 127/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4639 - accuracy: 0.7884 - val_loss: 0.5032 - val_accuracy: 0.812
9
5/5 [==============================] - 0s 1ms/step - loss: 0.5082 - accuracy: 0.8129
[0.5082343816757202, 0.8129496574401855]
```



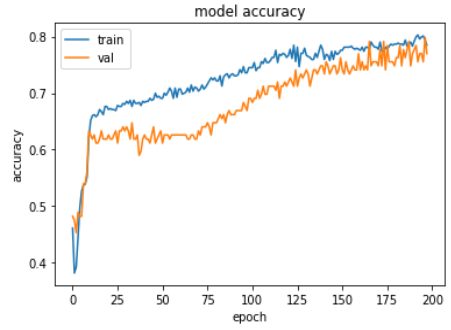
2nd round early stop at epoch 104, accuracy 0.7410

```
Epoch 104/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4800 - accuracy: 0.7667 - val_loss: 0.5537 - val_accuracy: 0.741
0
5/5 [==============================] - 0s 1ms/step - loss: 0.5555 - accuracy: 0.7410
[0.5554778575897217, 0.7410072088241577]
```



3rd round early stop at epoch 198, accuracy 0.7554

```
Epoch 198/500
18/18 [==============================] - 0s 2ms/step - loss: 0.4483 - accuracy: 0.7848 - val_loss: 0.5080 - val_accuracy: 0.769
8
5/5 [==============================] - 0s 1ms/step - loss: 0.5094 - accuracy: 0.7554
[0.5093525648117065, 0.7553957104682922]
```

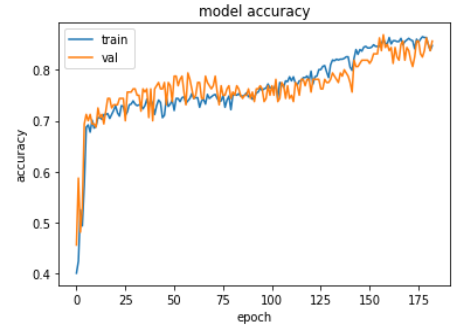


Heart Disease Experiment Result

Heart Disease Original Data Result

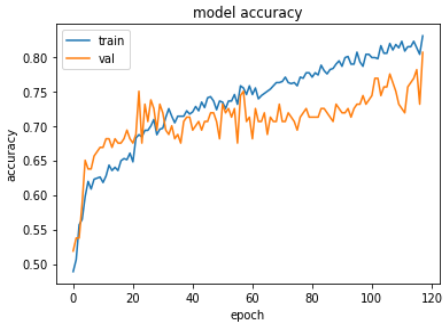
1st round early stop at epoch 183, accuracy 0.8250

```
Epoch 183/500
20/20 [==============================] - 0s 2ms/step - loss: 0.3701 - accuracy: 0.8475 - val_loss: 0.3895 - val_accuracy: 0.856
2
5/5 [==============================] - 0s 998us/step - loss: 0.3944 - accuracy: 0.8250
[0.3944062292575836, 0.824999988079071]
```



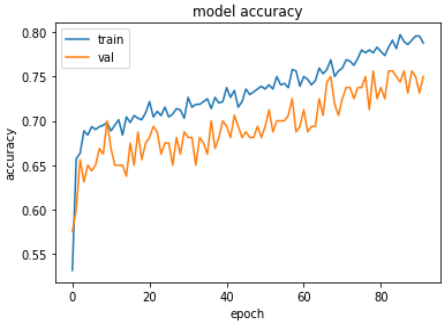
2nd round early stop at epoch 118, accuracy 0.7563

```
Epoch 118/500
20/20 [==============================] - 0s 2ms/step - loss: 0.3799 - accuracy: 0.8302 - val_loss: 0.4250 - val_accuracy: 0.806
2
5/5 [==============================] - 0s 997us/step - loss: 0.4230 - accuracy: 0.7563
[0.4230409562587738, 0.7562500238418579]
```



3rd round early stop at epoch 92, accuracy 0.7563

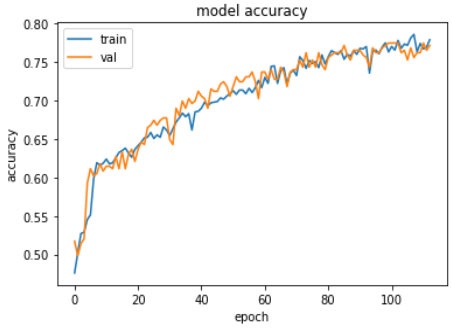
```
Epoch 92/500
20/20 [==============================] - 0s 2ms/step - loss: 0.4839 - accuracy: 0.7877 - val_loss: 0.5199 - val_accuracy: 0.750
0
5/5 [==============================] - 0s 997us/step - loss: 0.5173 - accuracy: 0.7563
[0.5173337459564209, 0.7562500238418579]
```



Heart Disease Noise (1/9) Data Result

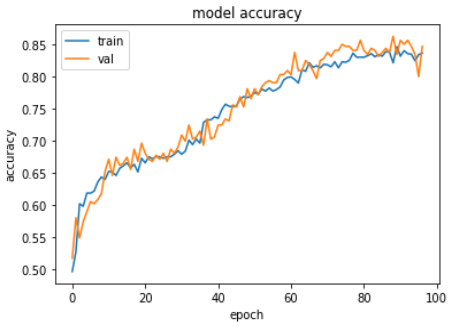
1st round early stop at epoch 113, accuracy 0.7555

```
Epoch 113/500
40/40 [==============================] - 0s 1ms/step - loss: 0.5027 - accuracy: 0.7785 - val_loss: 0.4998 - val_accuracy: 0.771
2
10/10 [==============================] - 0s 997us/step - loss: 0.5127 - accuracy: 0.7555
[0.5126573443412781, 0.7554858922958374]
```



2nd round early stop at epoch 97, accuracy 0.8495

```
Epoch 97/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3927 - accuracy: 0.8358 - val_loss: 0.3925 - val_accuracy: 0.846
4
10/10 [==============================] - 0s 886us/step - loss: 0.4067 - accuracy: 0.8495
[0.4067074656486511, 0.8495298027992249]
```



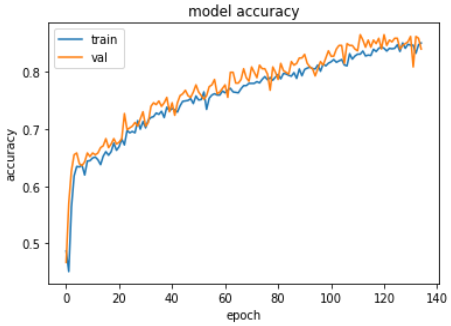
3rd round early stop at epoch 107, accuracy 0.8621

```
Epoch 107/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3551 - accuracy: 0.8437 - val_loss: 0.3501 - val_accuracy: 0.849
5
10/10 [==============================] - 0s 998us/step - loss: 0.3489 - accuracy: 0.8621
[0.3489378094673157, 0.8620689511299133]
```

Heart Disease Noise (1/18) Data Result

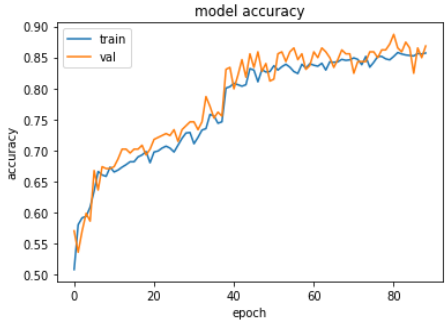
1st round early stop at epoch 135, accuracy 0.8527

```
Epoch 135/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3752 - accuracy: 0.8507 - val_loss: 0.3781 - val_accuracy: 0.840
1
10/10 [==============================] - 0s 887us/step - loss: 0.3528 - accuracy: 0.8527
[0.3528061807155609, 0.852664589881897]
```



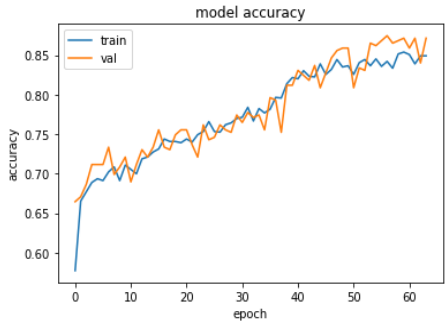
2nd round early stop at epoch 89, accuracy 0.8746

```
Epoch 89/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3454 - accuracy: 0.8570 - val_loss: 0.3551 - val_accuracy: 0.868
3
10/10 [==============================] - 0s 888us/step - loss: 0.3655 - accuracy: 0.8746
[0.36554524302482605, 0.8746081590652466]
```



3rd round early stop at epoch 64, accuracy 0.8683

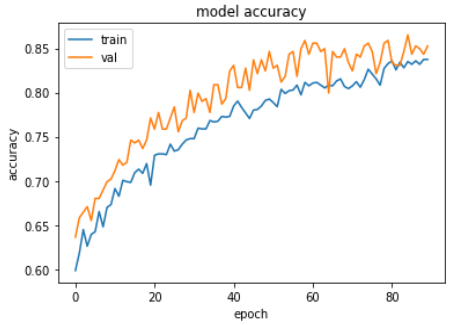
```
Epoch 64/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3622 - accuracy: 0.8492 - val_loss: 0.3845 - val_accuracy: 0.871
5
10/10 [==============================] - 0s 886us/step - loss: 0.3908 - accuracy: 0.8683
[0.39081111550331116, 0.8683385848999023]
```



Heart Disease Noise (1/27) Data Result

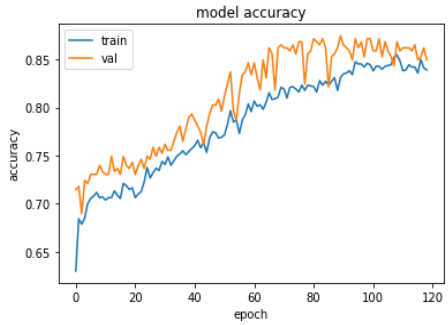
1st round early stop at epoch 90, accuracy 0.8652

```
Epoch 90/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3867 - accuracy: 0.8374 - val_loss: 0.3616 - val_accuracy: 0.852
7
10/10 [==============================] - 0s 776us/step - loss: 0.3798 - accuracy: 0.8652
[0.37976720929145813, 0.8652037382125854]
```



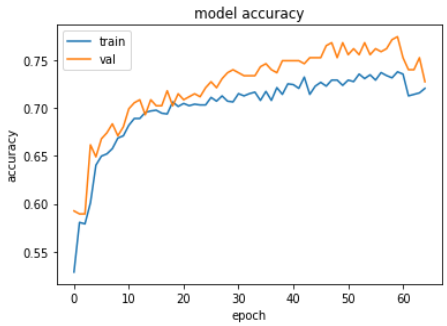
2nd round early stop at epoch 119, accuracy 0.8589

```
Epoch 119/500
40/40 [==============================] - 0s 1ms/step - loss: 0.3626 - accuracy: 0.8390 - val_loss: 0.3404 - val_accuracy: 0.849
5
10/10 [==============================] - 0s 775us/step - loss: 0.3303 - accuracy: 0.8589
[0.3302598297595978, 0.8589341640472412]
```



3rd round early stop at epoch 65, accuracy 0.7743

```
Epoch 65/500
40/40 [==============================] - 0s 1ms/step - loss: 0.5496 - accuracy: 0.7203 - val_loss: 0.5376 - val_accuracy: 0.727
3
10/10 [==============================] - 0s 776us/step - loss: 0.5246 - accuracy: 0.7743
[0.5246334671974182, 0.7742946743965149]
```



Result Table

Breast Cancer	Original	Noise1/9	Noise1/18	Noise1/27
1st round	0.8594	0.8438	0.8438	0.7891
2nd round	0.9062	0.7812	0.8438	0.7500
3rd round	0.8750	0.8047	0.8672	0.7891
Average	**0.8802**	**0.8099**	**0.8516**	**0.7760**

Diabetes	Original	Noise1/9	Noise1/18	Noise1/27
1st round	0.6279	0.6353	0.6118	0.7059
2nd round	0.5349	0.6118	0.6471	0.6353
3rd round	0.6279	0.6000	0.5412	0.6471
Average	**0.5969**	**0.6157**	**0.6000**	**0.6627**

Smoking	Original	Noise1/9	Noise1/18	Noise1/27
1st round	0.7138	0.6400	0.6928	0.7019
2nd round	0.7013	0.6658	0.7033	0.6899
3rd round	0.7079	0.6734	0.6587	0.6766
Average	**0.7076**	**0.6597**	**0.6849**	**0.6894**

Wine Quality	Original	Noise1/9	Noise1/18	Noise1/27
1st round	0.7857	0.7122	0.7914	0.8129
2nd round	0.8286	0.6835	0.7338	0.7410
3rd round	0.8000	0.7698	0.7842	0.7554
Average	**0.8047**	**0.7218**	**0.7698**	**0.7697**

Heart Disease	Original	Noise1/9	Noise1/18	Noise1/27
1st round	0.8250	0.7555	0.8527	0.8652
2nd round	0.7563	0.8495	0.8746	0.8589
3rd round	0.7563	0.8621	0.8683	0.7743
Average	**0.7792**	**0.8223**	**0.8652**	**0.8328**

Conclusion

Dataset	Original	Noise1/9	Noise1/18	Noise1/27
Breast Cancer	0.8802	0.8099	0.8516	0.7760
Diabetes	0.5969	0.6157	0.6000	0.6627
Smoking	0.7076	0.6597	0.6849	0.6894
Wine Quality	0.8047	0.7218	0.7698	0.7697
Heart Disease	0.7792	0.8223	0.8652	0.8328

From the results that we get from each dataset with the original and noise injection in each scale value. We could clearly see from the table that the noise injection technique doesn't make impact on training the model that much. Just only for the heart disease dataset that the accuracy from training noise is slightly higher than the accuracy of training the original data,

To sum up, based on this experiment, the noise injection technique doesn't make significant improvement to the generalization of neural network.

References

[1] Explicit Regularisation in Gaussian Noise Injections Research/Retrieved

Oct 18,2022<https://arxiv.org/pdf/2007.07368.pdf>

[2] Train Neural Network With noise/Retrieved

Oct 18,2022<https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/>

[3] Noise Affect generalization/Retrieved Oct 18,2022<https://ai.stackexchange.com/questions/2/how-does-noise-affect-generalization>

[4] Kaggle Datasets/retrieved Sep 10,2022<https://www.kaggle.com/>

[5] Data finder/ retrieved Oct 3,2022https://datasetsearch.research.google.com/

[6] Breast Cancer Dataset/retrieved Sep 10,2022<https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data>

[7] Diabetes Dataset/retrieved Sep 10,2022<https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

[8] Smoking Dataset/retrieved Sep 10,2022<https://www.kaggle.com/datasets/kukuroo3/body-signal-of-smoking>

[9] Wine Quality Dataset/retrieved Sep 10,2022< https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009>

[10] Heart Disease Dataset/retrieved Sep 10,2022<https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset>