Design Model

In [1]:
```
# dont forget activate conda virtual environment (cs461)
# import dependencies
# !pip install numpy pandas tensorflow matplotlib scikit-learn

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from matplotlib import pyplot as plt
%matplotlib inline
```

Upload data

In [2]:
```
# worked on google colab only
# from google.colab import files

data_file = "./data/population_dataset.csv"

df = pd.read_csv(data_file)  # read the dataset
```

In [3]:
```
df.shape
```

Out[3]:
```
(400, 11)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'Marriage',
       'work_type', 'Living_type', 'avg_glucose', 'bmi', 'smoking_status',
       'illness'],
      dtype='object')
```

In [5]:
```
df.isnull().any()
```

Out[5]:
```
gender            False
age               False
hypertension      False
heart_disease     False
Marriage          False
work_type         False
Living_type       False
avg_glucose       False
bmi               False
smoking_status    False
illness           False
dtype: bool
```

In [6]:
```
df.nunique()
```

Out[6]:
```
gender             2
age               79
hypertension       2
heart_disease      2
Marriage           2
work_type          4
Living_type        2
avg_glucose      393
bmi              199
smoking_status     4
illness            2
dtype: int64
```

In [7]:
```
df.describe(include='all')
```

Out[7]:

	gender	age	hypertension	heart_disease	Marriage	work_type	Living_type	avg_glucose	bmi	smoking_status
count	400	400.00000	400.000000	400.000000	400	400	400	400.000000	400.000000	400
unique	2	NaN	NaN	NaN	2	4	2	NaN	NaN	4
top	Female	NaN	NaN	NaN	Yes	Private	Urban	NaN	NaN	never smoked
freq	214	NaN	NaN	NaN	306	231	201	NaN	NaN	164
mean	NaN	55.26780	0.180000	0.132500	NaN	NaN	NaN	119.391950	29.481750	NaN
std	NaN	22.51279	0.384669	0.339458	NaN	NaN	NaN	54.377459	6.488354	NaN
min	NaN	0.80000	0.000000	0.000000	NaN	NaN	NaN	56.070000	15.600000	NaN
25%	NaN	44.00000	0.000000	0.000000	NaN	NaN	NaN	80.460000	25.575000	NaN
50%	NaN	59.00000	0.000000	0.000000	NaN	NaN	NaN	97.665000	28.600000	NaN
75%	NaN	74.25000	0.000000	0.000000	NaN	NaN	NaN	144.345000	33.025000	NaN
max	NaN	82.00000	1.000000	1.000000	NaN	NaN	NaN	271.740000	48.900000	NaN

In [8]:
```
df["gender"] = df["gender"].map({"Male": 0, "Female": 1})
df["Marriage"] = df["Marriage"].map({"No": 0, "Yes": 1})
df["work_type"] = df["work_type"].map({"Private": 0, "Self-employed": 1, "Govt_job": 2, "children": 3})
df["Living_type"] = df["Living_type"].map({"Rural": 0, "Urban": 1})
df["smoking_status"] = df["smoking_status"].map({"formerly smoked": 0, "never smoked": 1, "smokes": 2, "Unknown": 3})
```

In [9]:
```
df
```

Out[9]:

	gender	age	hypertension	heart_disease	Marriage	work_type	Living_type	avg_glucose	bmi	smoking_status	illness
0	0	67.0	0	1	1	0	1	228.69	36.6	0	1
1	0	80.0	0	1	1	0	0	105.92	32.5	1	1
2	1	49.0	0	0	1	0	1	171.23	34.4	2	1
3	1	79.0	1	0	1	1	0	174.12	24.0	1	1
4	0	81.0	0	0	1	0	1	186.21	29.0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
395	1	54.0	0	1	1	0	1	140.28	37.1	0	0
396	0	67.0	0	0	1	2	0	244.28	29.4	0	0
397	0	53.0	0	0	1	0	1	124.16	31.7	1	0
398	0	47.0	0	0	1	0	0	93.55	31.4	1	0
399	0	44.0	0	0	1	0	1	99.34	33.1	1	0

400 rows × 11 columns

Data pre-processing

In [10]:
```
y = df["illness"]
x = df.drop(columns=["illness"])
```

In [11]:
```
x.head()
```

Out[11]:

	gender	age	hypertension	heart_disease	Marriage	work_type	Living_type	avg_glucose	bmi	smoking_status
0	0	67.0	0	1	1	0	1	228.69	36.6	0
1	0	80.0	0	1	1	0	0	105.92	32.5	1
2	1	49.0	0	0	1	0	1	171.23	34.4	2
3	1	79.0	1	0	1	1	0	174.12	24.0	1
4	0	81.0	0	0	1	0	1	186.21	29.0	0

In [12]:
```
y.head()
```

Out[12]:
```
0    1
1    1
2    1
3    1
4    1
Name: illness, dtype: int64
```

In [13]:
```
y.value_counts()
```

Out[13]:
```
illness
1    200
0    200
Name: count, dtype: int64
```

In [14]:
```
from sklearn.preprocessing import MinMaxScaler

scale = MinMaxScaler()  # create a scaler instance
x_scaled = pd.DataFrame(scale.fit_transform(x), columns=x.columns)  # scale the data by fitting and transforming
x_scaled
```

Out[14]:

	gender	age	hypertension	heart_disease	Marriage	work_type	Living_type	avg_glucose	bmi	smoking_status
0	0.0	0.815271	0.0	1.0	1.0	0.000000	1.0	0.800389	0.630631	0.000000
1	0.0	0.975369	0.0	1.0	1.0	0.000000	0.0	0.231140	0.507508	0.333333
2	1.0	0.593596	0.0	0.0	1.0	0.000000	1.0	0.533964	0.564565	0.666667
3	1.0	0.963054	1.0	0.0	1.0	0.333333	0.0	0.547364	0.252252	0.333333
4	0.0	0.987685	0.0	0.0	1.0	0.000000	1.0	0.603422	0.402402	0.000000
...	...	...	...	...	...	...	...	...	...	...
395	1.0	0.655172	0.0	1.0	1.0	0.000000	1.0	0.390458	0.645646	0.000000
396	0.0	0.815271	0.0	0.0	1.0	0.666667	0.0	0.872676	0.414414	0.000000
397	0.0	0.642857	0.0	0.0	1.0	0.000000	1.0	0.315714	0.483483	0.333333
398	0.0	0.568966	0.0	0.0	1.0	0.000000	0.0	0.173784	0.474474	0.333333
399	0.0	0.532020	0.0	0.0	1.0	0.000000	1.0	0.200631	0.525526	0.333333

400 rows × 10 columns

Split dataset into Train set and Test set

In [15]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2, random_state=42)
```

Design Model and config Hyper Parameters

In [16]:
```
model = keras.Sequential([
    keras.Input(shape=(10,)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	704
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 1)	33

Total params: 2,817 (11.00 KB)

Trainable params: 2,817 (11.00 KB)

Non-trainable params: 0 (0.00 B)

In [17]:
```
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

Fitting Model (Training)

In [18]:
```
epoch = 100
model_result = model.fit(x_train, y_train, epochs=epoch)
```

```
Epoch 1/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5770 - loss: 0.6739
Epoch 2/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6579 - loss: 0.5923
Epoch 3/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7017 - loss: 0.5642
Epoch 4/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7578 - loss: 0.5231
Epoch 5/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7043 - loss: 0.5453
Epoch 6/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7522 - loss: 0.5130
Epoch 7/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7771 - loss: 0.4700
Epoch 8/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7630 - loss: 0.4586
Epoch 9/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8088 - loss: 0.4359
Epoch 10/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7779 - loss: 0.4430
Epoch 11/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8114 - loss: 0.4234
Epoch 12/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8150 - loss: 0.3922
Epoch 13/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7625 - loss: 0.4427
Epoch 14/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8231 - loss: 0.4136
Epoch 15/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7940 - loss: 0.4246
Epoch 16/100
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7762 - loss: 0.4577
Epoch 17/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8106 - loss: 0.3989
Epoch 18/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8388 - loss: 0.3527
Epoch 19/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8089 - loss: 0.4026
Epoch 20/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8580 - loss: 0.3606
Epoch 21/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7988 - loss: 0.3819
Epoch 22/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8221 - loss: 0.3590
Epoch 23/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7967 - loss: 0.3748
Epoch 24/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8184 - loss: 0.3770
Epoch 25/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8458 - loss: 0.3419
Epoch 26/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8310 - loss: 0.3399
Epoch 27/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8167 - loss: 0.3541
Epoch 28/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8228 - loss: 0.3683
Epoch 29/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8358 - loss: 0.3355
Epoch 30/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8552 - loss: 0.3135
Epoch 31/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8666 - loss: 0.2841
Epoch 32/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8437 - loss: 0.3280
Epoch 33/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8504 - loss: 0.2992
Epoch 34/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8235 - loss: 0.3452
Epoch 35/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8272 - loss: 0.3352
Epoch 36/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7976 - loss: 0.3370
Epoch 37/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8466 - loss: 0.3101
Epoch 38/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8551 - loss: 0.2994
Epoch 39/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8567 - loss: 0.2683
Epoch 40/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8620 - loss: 0.2648
Epoch 41/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8683 - loss: 0.2939
Epoch 42/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8824 - loss: 0.2652
Epoch 43/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8634 - loss: 0.2672
Epoch 44/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8919 - loss: 0.2602
Epoch 45/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8690 - loss: 0.2520
Epoch 46/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8756 - loss: 0.2822
Epoch 47/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8556 - loss: 0.2738
Epoch 48/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8530 - loss: 0.2835
Epoch 49/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9051 - loss: 0.2204
Epoch 50/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8633 - loss: 0.3039
Epoch 51/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8514 - loss: 0.3056
Epoch 52/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8713 - loss: 0.2520
Epoch 53/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8507 - loss: 0.2586
Epoch 54/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8557 - loss: 0.2455
Epoch 55/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8574 - loss: 0.2751
Epoch 56/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8919 - loss: 0.2666
Epoch 57/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8744 - loss: 0.2777
Epoch 58/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8651 - loss: 0.2947
Epoch 59/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9035 - loss: 0.2315
Epoch 60/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8849 - loss: 0.2299
Epoch 61/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8615 - loss: 0.2722
Epoch 62/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8986 - loss: 0.2095
Epoch 63/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9070 - loss: 0.1995
Epoch 64/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9165 - loss: 0.1994
Epoch 65/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9041 - loss: 0.2102
Epoch 66/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9246 - loss: 0.1749
Epoch 67/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8919 - loss: 0.2280
Epoch 68/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8893 - loss: 0.2281
Epoch 69/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9137 - loss: 0.1993
Epoch 70/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9097 - loss: 0.1832
Epoch 71/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9322 - loss: 0.1867
Epoch 72/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9210 - loss: 0.1789
Epoch 73/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9165 - loss: 0.1871
Epoch 74/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9367 - loss: 0.1644
Epoch 75/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9220 - loss: 0.1854
Epoch 76/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9032 - loss: 0.1943
Epoch 77/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9042 - loss: 0.1896
Epoch 78/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8966 - loss: 0.1725
Epoch 79/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8857 - loss: 0.2165
Epoch 80/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9120 - loss: 0.2021
Epoch 81/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9286 - loss: 0.1915
Epoch 82/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9196 - loss: 0.1789
Epoch 83/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9216 - loss: 0.1652
Epoch 84/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9008 - loss: 0.1842
Epoch 85/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9332 - loss: 0.1859
Epoch 86/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9070 - loss: 0.1655
Epoch 87/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9339 - loss: 0.1392
Epoch 88/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9420 - loss: 0.1366
Epoch 89/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9428 - loss: 0.1502
Epoch 90/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9253 - loss: 0.1481
Epoch 91/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9098 - loss: 0.1607
Epoch 92/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9382 - loss: 0.1528
Epoch 93/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9456 - loss: 0.1404
```

```
Epoch 94/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9453 - loss: 0.1316
Epoch 95/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9421 - loss: 0.1305
Epoch 96/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9185 - loss: 0.1550
Epoch 97/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9562 - loss: 0.1227
Epoch 98/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9454 - loss: 0.1267
Epoch 99/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9338 - loss: 0.1362
Epoch 100/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9493 - loss: 0.1215
```

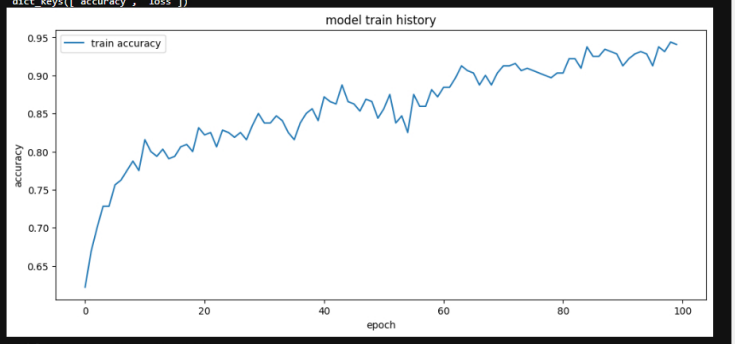
In [19]:
```
model.evaluate(x_test, y_test)
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7141 - loss: 1.0179
```

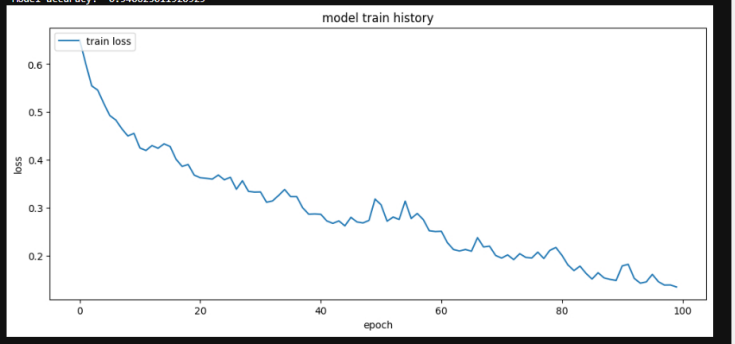
Out[19]: `[0.968537449836731, 0.7250000238418579]`

In [20]:
```
def plot_train_result(model_result, key, legend, xlim=None, ylim=None, ):
    plt.figure(figsize=(12, 5))

    plt.plot(model_result.history[key])

    # set title labels and legend
    plt.title('model train history')
    plt.ylabel(key)
    plt.xlabel('epoch')
    plt.legend([legend], loc='upper left')

    # set x-axis limit to zoom in specific area
    if (xlim and len(xlim) == 2):
        plt.xlim(xlim)          # specific epoch range
    if (ylim and len(ylim) == 2):
        plt.ylim(ylim)        # specific accuracy range

    # display the plot
    plt.show()

print(model_result.history.keys())

plot_train_result(model_result, 'accuracy', 'train accuracy')
print("Model accuracy: ", model_result.history['accuracy'][-1])

plot_train_result(model_result, 'loss', 'train loss')
print("Model loss: ", model_result.history['loss'][-1])
```

```
dict_keys(['accuracy', 'loss'])
```




```
Model accuracy:  0.940625011920929
```




```
Model loss:  0.13430866599082947
```

In [21]:
```
model.save('./model/model.keras')
```