LAPORAN PRAKTIKUM KECERDASAN BUATAN

KLASIFIKASI GAGAL JANTUNG MENGGUNAKAN ARTIFICIAL NEURAL NETWORK

Bane Rael Sharin[1)], Dimas Wahyu Saputro[2)], Fadilla Muhesa Latri[3)], Lisnurani[4)], Marhanny Zahra N[5)].

Program Studi Sains Data, Jurusan Sains, Institut Teknologi Sumatera

bane.120450101@student.itera.ac.id [1)], *dimas.120450081*@student.itera.ac.id [2)], *fadillah.120450003*@student.itera.ac.id [3)], *lisnurani.*120450055student.itera.ac.id [4)], marhanny.120450017@student.itera.ac.id [5]

Abstrak

Gagal jantung merupakan kondisi otot jantung yang sangat lemah sehingga tidak dapat memompa darah ke seluruh tubuh. Gagal jantung merupakan salah satu penyakit mematikan di seluruh dunia, ada lebih 550.000 kasus yang terjadi setiap tahunnya. Sebagian besar penyakit gagal jantung dapat dicegah dengan mengatasi faktor risiko perilaku, seperti konsumsi rokok, pola makan yang tidak sehat dan obesitas, kurangnya aktivitas fisik, dan konsumsi alkohol yang berbahaya. Orang dengan penyakit gagal jantung atau memiliki risiko adanya penyakit gagal jantung memerlukan deteksi dan penanganan dini yang dibantu oleh machine learning yang dapat mendeteksinya. Maka dari itu dalam penelitian ini, dilakukan studi kasus ini menggunakan metode *Artificial Neural Network* dengan menggunakan 12 variabel yang berguna untuk mengetahui klasifikasi penyebab gagal jantung. Setelah dilakukan percobaan, didapatkan akurasi 0.85 untuk klasifikasi.

Kata Kunci : *Artificial Neural Network,* Gagal jantung

1. Pendahuluan

Gagal jantung merupakan masalah kesehatan serius yang berdampak besar pada kehidupan masyarakat. Selain usia, bertambahnya ukuran porsi dan ketidakaktifan, kebanyakan orang mengabaikan kesehatannya. Juga karena kerusakan lingkungan, faktor-faktor ini dapat menyebabkan masalah gagal jantung, yang akan lebih sering terjadi di masa mendatang. Akhirnya, ketika orang tidak memperhatikan gagal jantung, itu menyebabkan kematian [1]. Dalam beberapa tahun terakhir, berbagai peneliti telah mengumpulkan dan menganalisis berbagai metode untuk memprediksi gagal jantung.

Gagal jantung pada dasarnya dapat dikatakan sebagai suatu penyakit yang disebabkan oleh melemahnya fungsi dari jantung penderitanya sehingga tidak dapat memompa darah keseluruh tubuh. Kondisi tersebut akan semakin bertambah parah ketika penderitanya tidak menyadari akan kondisi tersebut. Penderita dengan penyakit jantung akan merasa seperti pada hari-hari biasa saja namun dalam hal ini kondisi tersebut dapat dengan cepat membahayakan penderita dan merenggut nyawa dari penderita. Ada berbagai faktor yang mendasari terjadinya gagal jantung diantaranya disebabkan

oleh penumpukan lemak yang ada pada daerah arteri dan pola hidup dari penderita yang kurang bagus.

Berdasarkan fakta yang ada, bahwasanya penderita jantung memiliki faktor penyebab yang menjadi dasar terjadinya penyakit jantung diantaranya pola tidur yang berlebihan, pola makan yang tidak sehat, pola hidup yang sembarangan dan bahkan disebabkan oleh faktor genetik dari penderitanya.

Penyakit ini menjadi satu-satunya penyakit yang saat ini terus meningkat resiko dan juga insiden di setiap tahunya. Perubahan gaya hidup di era sekarang yang menyebabkan penyakit ini semakin merajai. Pola makanan seperti berlebihan Mengonsumsi kalori, garam, lemak, rokok, alkohol dan penurunan aktivitas menjadi faktor utama hadirnya penyakit gagal jantung.

Sampai saat ini penyakit gagal jantung masuk terus meningkat, namun meskipun penyakit ini terus memakan korbannya secara tiba-tiba, untuk upaya pencegahan dan antisipasi telah dilakukan. Dalam hal ini, penelitian kami bertujuan untuk mengetahui untuk mengetahui faktor penyebab dari penyakit gagal jantung menggunakan Algoritma *Artificial Neural Network* (ANN).

Penelitian ini diharapkan dapat digunakan untuk media penyalur informasi bagi pelayanan kesehatan terkait penyakit gagal jantung kepada masyarakat dan menyampaikan informasi-informasi penting terkait faktor-faktor yang menjadi pangkal utama dari penyakit gagal jantung sedini mungkin agar dapat dilakukan suatu pencegahan dan modifikasi atas penyakit tersebut agar dapat meminimalisir dari insiden penyakit gagal jantung.

2. Metode

a. Deskripsi Data Set

Data set penelitian ini berasal dari kumpulan data Davide Chicco, Giuseppe Jurman kaggle. Data terdiri dari 13 kolom yaitu *age, anaemia, creatinine phosphokinase, diabetes, ejection fraction, high blood pressure, platelets, serum creatinine, serum sodium, sex, smoking, time, death event* dan 299 baris. Seperti pada gambar dibawah ini.

[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

Gambar 1. Data yang di gunakan

b. Proses Pembersihan Data

Data cleaning adalah suatu prosedur untuk memastikan kebenaran, konsistensi, dan kegunaan suatu data yang ada dalam dataset. Caranya adalah dengan mendeteksi adanya error atau corrupt pada data, kemudian memperbaiki atau menghapus data jika memang diperlukan. Pada data yang digunakan pada artikel ini, semua baris terisi

dengan baik. Selain itu, tipe data yang digunakan juga sudah pas. Jadi, tidak dibutuhkan pembersihan data.

c. *Artificial Neural Network*

Pemodelan klasifikasi untuk mendeteksi gagal jantung yang dilakukan pada penelitian ini menggunakan algoritma *Artificial Neural Network* (ANN). ANN merupakan salah satu pemodelan kompleks yang dapat memprediksi bagaimana ekosistem merespon perubahan variabel lingkungan dengan terinspirasi oleh cara kerja sistem saraf biologis, khususnya pada sel otak manusia dalam memproses informasi. Dikarenakan model ANN terinspirasi oleh sistem saraf biologis manusia, arsitekturnya pun dibuat seperti struktur otak manusia dimana terdiri dari neuron yang saling terhubung satu sama lain dan bentuk yang kompleks dan non-linear. Jumlah data yang digunakan sebanyak 299 baris dikali 13 kolom.

d. Matriks Evaluasi

Pada penelitian ini menggunakan *confusion matrix* yaitu pengukuran masalah klasifikasi *machine learning* dengan dua (2) keluaran atau lebih, pada *confusion matrix* memiliki empat (4) tabel, setiap tabel berisi *True Positif, True Negatif, False Positif, dan False Negatif.*

Confusion Matrix

	Prediksi	
Aktual	**TRUE**	**FALSE**
TRUE	*TP*	*FP*
FALSE	*FN*	*TN*

Gambar 2. Tabel *confusion matrix*

Selain *confusion matrix* digunakan juga *accuracy matrix* yaitu menghitung rasio prediksi benar dibagi dengan prediksi salah [2].

Accuracy Predictions/ Classifications $= \frac{\text{Correct}}{\text{Correct + Incorrect}}$

Gambar 3. *matrix accuracy*

3. Hasil

Artificial Neural Network (ANN) atau Jaringan Saraf Tiruan merupakan pemodelan kompleks yang digunakan pada praktikum kali ini. Langkah awal, dilakukan *modeling* data menggunakan ANN seperti pada gambar 4. Layar masukan (*input*) terdiri dari 156 *layer*, layar tersembunyi (*hidden*) terdiri dari 128 *layer,* dan akan mengeluarkan 1 layar keluaran (*output*).

```
[ ]  1  #Importing libraries
     2  import tensorflow.keras
     3  from tensorflow.keras.models import Sequential
     4  from tensorflow.keras.layers import Dense
     5
     6  model = Sequential()
     7  model.add(Dense(156, input_dim=12, activation='relu'))
     8  model.add(Dense(128, activation='relu'))
     9  model.add(Dense(1, activation='sigmoid'))
    10  model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 156)               2028

 dense_4 (Dense)             (None, 128)               20096

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 22,253
Trainable params: 22,253
Non-trainable params: 0
_________________________________________________________________
```

Gambar 4. *Model Neural Network*

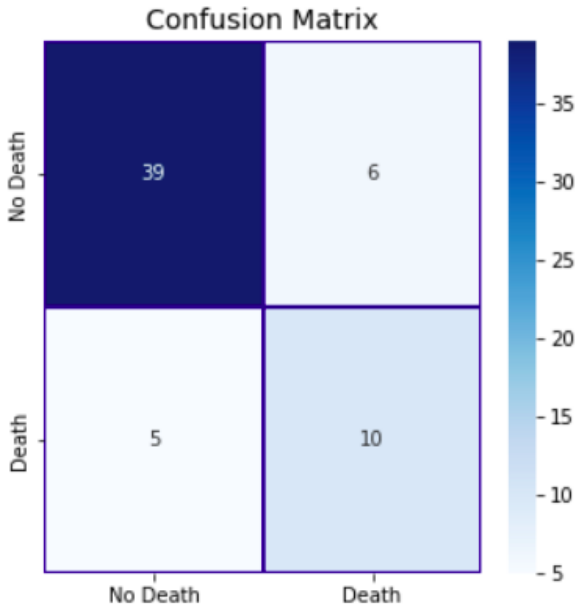
Setelah dilakukan pemodelan, langkah selanjutnya yaitu meng-*compile* model. Model optimizer yang digunakan adalah *Adam, metrics* menggunakan *Accuracy,* dan *loss* menggunakan *binary_crossentropy,* seperti pada Gambar 6. Selanjutnya, dilakukan *training*.

```
1  # Compiling the ANN
2  #opt = Adam(learning_rate=0.00009)
3  model.compile(optimizer = "Adam", loss = 'binary_crossentropy', metrics = ['accuracy'])
4
5  # Train the ANN
6  history = model.fit(xtrain, ytrain, batch_size = 32, epochs = 30,  validation_split=0.2)
```

```
Epoch 1/30
6/6 [==============================] - 1s 33ms/step - loss: 0.6716 - accuracy: 0.6178 - val_loss: 0.6691 - val_accuracy: 0.6250
Epoch 2/30
6/6 [==============================] - 0s 10ms/step - loss: 0.6197 - accuracy: 0.6702 - val_loss: 0.6577 - val_accuracy: 0.6250
Epoch 3/30
6/6 [==============================] - 0s 11ms/step - loss: 0.5987 - accuracy: 0.6702 - val_loss: 0.6391 - val_accuracy: 0.6250
Epoch 4/30
6/6 [==============================] - 0s 10ms/step - loss: 0.5788 - accuracy: 0.6702 - val_loss: 0.6162 - val_accuracy: 0.6250
```

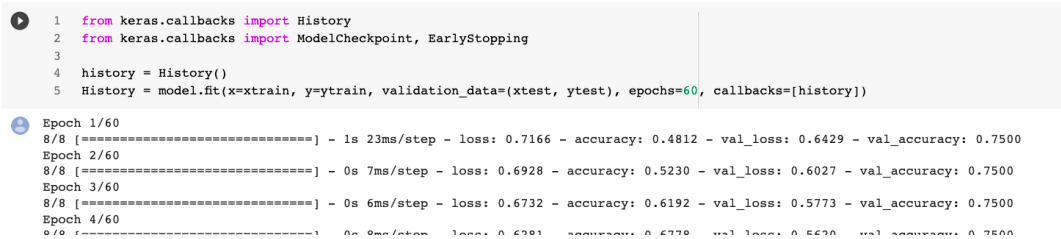
Gambar 5. Compile Model

Setelah itu dievaluasi menggunakan *accuracy_score*, dan didapatkan *accuracy test set* sebesar 0,81. Confusion Matrix terlampir pada Gambar 6.



Gambar 6. Confusion Matrix

Kemudian, dilakukan Tuning Hyperparameter seperti menambah *dropout* pada model, atau melakukan *early stopping* pada proses model fit, seperti pada Gambar 7.



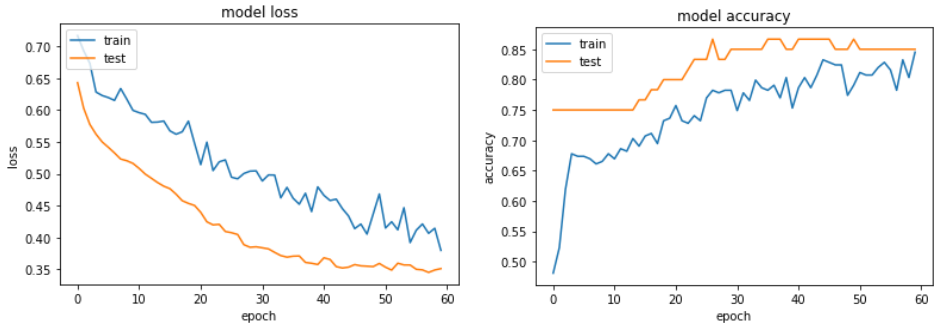
```
from keras.callbacks import History
from keras.callbacks import ModelCheckpoint, EarlyStopping

history = History()
History = model.fit(x=xtrain, y=ytrain, validation_data=(xtest, ytest), epochs=60, callbacks=[history])
```

```
Epoch 1/60
8/8 [==============================] - 1s 23ms/step - loss: 0.7166 - accuracy: 0.4812 - val_loss: 0.6429 - val_accuracy: 0.7500
Epoch 2/60
8/8 [==============================] - 0s 7ms/step - loss: 0.6928 - accuracy: 0.5230 - val_loss: 0.6027 - val_accuracy: 0.7500
Epoch 3/60
8/8 [==============================] - 0s 6ms/step - loss: 0.6732 - accuracy: 0.6192 - val_loss: 0.5773 - val_accuracy: 0.7500
Epoch 4/60
```

Gambar 7. Tuning Hyperparameter

Untuk memudahkan, dilakukan visualisasi *model loss* dan *model accuracy*, seperti pada Gambar 8. Hasil yang didapatkan sangat menarik setelah dilakukan sebanyak 60 epoch.



Gambar 8. Visualisasi

4. Kesimpulan

Berdasarkan hasil penelitian yang dilakukan dapat disimpulkan bahwa penerapan klasifikasi gagal jantung pada faktor penyebab penyakit gagal jantung dapat diterapkan menggunakan algoritma Artificial Neural Network. Atribut yang digunakan sebagai parameter peningkatan sistem pembelajaran mahasiswa antara lain: age (umur), anaemia (anemia), creatinine phosphokinase (tingkat enzim CPK dalam darah mcg/L), diabetes, ejection fraction (persentase darah yang meninggalkan jantung pada tiap kontraksi), high blood pressure (tekanan darah tinggi), platelets (trombosit), serum creatinine (tingkat kreatinin dalam darah mg/dL), serum sodium (tingkat natrium serum dalam darah mEq/L), sex (jenis kelamin), smoking (rokok), time (periode tindak lanjut), death event (pasien yang meninggal saat masa tindak lanjut). Hasil analisis yang didapatkan bahwa urutan variabel yang memiliki tingkat kepentingan yang tinggi adalah periode tindak lanjut (time), ejection fraction, merokok (smoking), diabetes, kreatinin (serum creatinine), jenis kelamin (sex), anemia (anaemia), tingkat natrium sodium (serum sodium), umur (age), tingkat enzim CPK (creatinine phosphokinase), trombosit (platelets), dan tekanan darah tinggi (high blood preassure). Pengujian juga dilakukan untuk pembuktian bahwa penggunaan algoritma Artificial Neural Network (ANN) dalam mengklasifikasikan penyakit gagal jantung memperoleh tingkat akurasi klasifikasi sebesar 0,85 atau 85%.

Referensi

[1] K. Sulaiman, L. Hakim Ismail, M. Adib Mohammad Razi, M. Shalahuddin Adnan, and R. Ghazali, “Water Quality Classification Using an Artificial Neural Network (ANN),” in *IOP Conference Series: Materials Science and Engineering*, Sep. 2019, vol. 601, no. 1. doi: 10.1088/1757-899X/601/1/012005.

[2] D. M. W. Powers and Ailab, “EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION.”

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
df = pd.read_csv('/Users/dimas/Documents/Programming/praktikum/ai/heart_failure_clinical_records_dataset.csv')
```

Reading data set

In [3]:

```
df.head()
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	se
0	75.0	0	582	0	20	1	265000.00	1.9	
1	55.0	0	7861	0	38	0	263358.03	1.1	
2	65.0	0	146	0	20	0	162000.00	1.3	
3	50.0	1	111	0	20	0	210000.00	1.9	
4	65.0	1	160	1	20	0	327000.00	2.7	

Check, no of columns, type and null values

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
 11  time                      299 non-null    int64  
 12  DEATH_EVENT               299 non-null    int64  
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

Alterntaevily checking null values

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

Ccheck unique value in Target collumn, which is Death_Event in our case

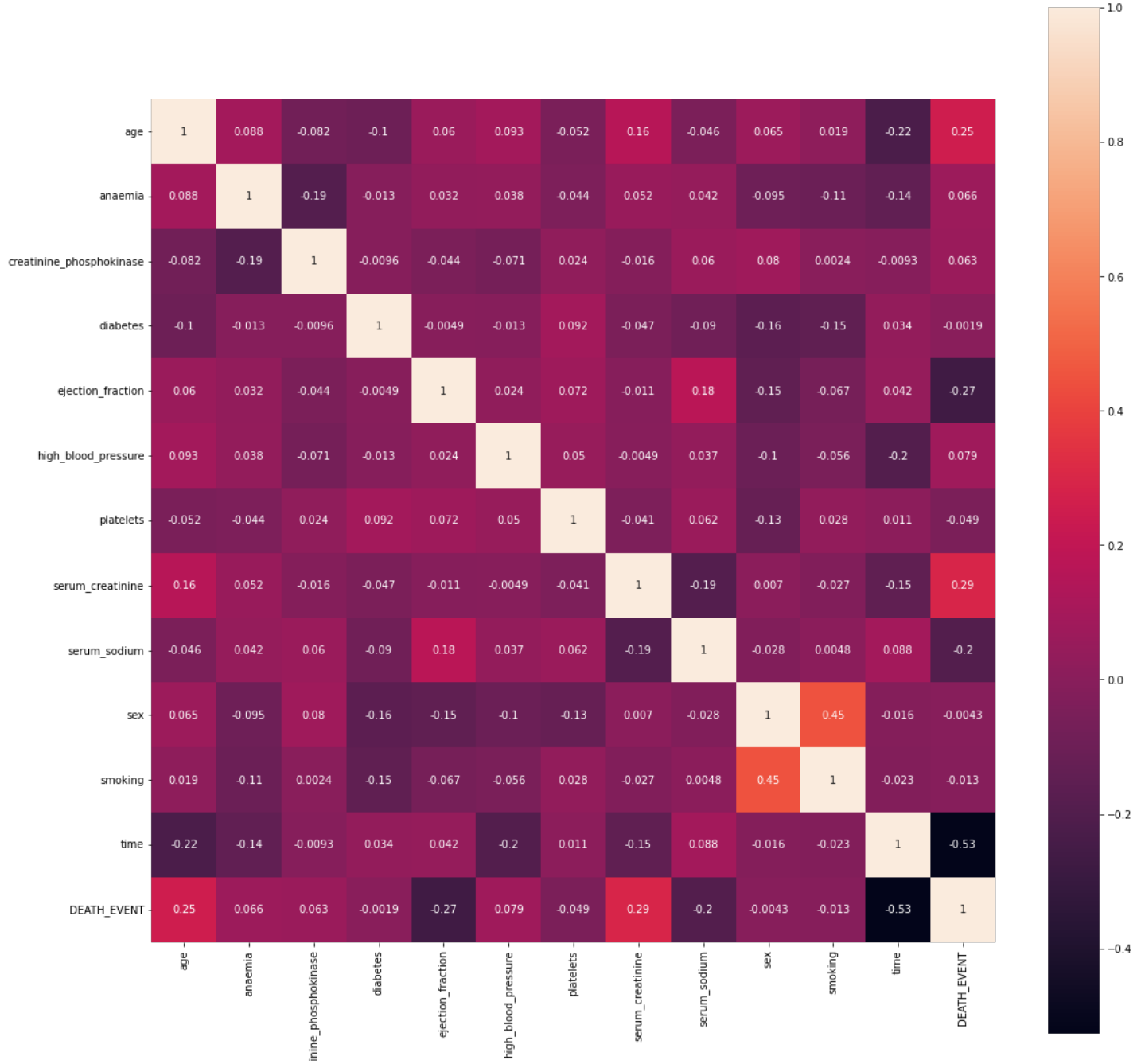
Co relation b/w all variables

In [31]:

```
plt.subplots(figsize=(18,18))
sns.heatmap(df.corr(),annot=True, square=True)
```

Out[31]:

```
<AxesSubplot:>
```



creati

Define Y and X variables

In [13]:

```
Y=df["DEATH_EVENT"]
X=df.drop(columns="DEATH_EVENT")
```

Scale the Variable

In [14]:

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X_scaled = mms.fit_transform(X)
Y_scaled = mms.fit_transform(Y.values.reshape(-1,1))
```

Spliting the data set into Train and Test

In [15]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(X_scaled,Y_scaled,test_size=0.20,random_stat
e=101)
```

In [16]:

```
xtrain.shape
```

Out[16]:

```
(239, 12)
```

In [17]:

```
ytrain.shape
```

Out[17]:

```
(239, 1)
```

Designing neural network for predcition

In [21]:

```
#Importing libraries
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(156, input_dim=12, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_3 (Dense)              (None, 156)               2028

 dense_4 (Dense)              (None, 128)               20096

 dense_5 (Dense)              (None, 1)                 129

=================================================================
Total params: 22,253
```

```
Trainable params: 22,253
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

```
# Compiling the ANN
#opt = Adam(learning_rate=0.00009)
model.compile(optimizer = "Adam", loss = 'binary_crossentropy', metrics = ['accuracy'])

# Train the ANN
history = model.fit(xtrain, ytrain, batch_size = 32, epochs = 30,  validation_split=0.2)
```

```
Epoch 1/30
6/6 [==============================] - 1s 33ms/step - loss: 0.6716 - accuracy: 0.6178 - v
al_loss: 0.6691 - val_accuracy: 0.6250
Epoch 2/30
6/6 [==============================] - 0s 10ms/step - loss: 0.6197 - accuracy: 0.6702 - v
al_loss: 0.6577 - val_accuracy: 0.6250
Epoch 3/30
6/6 [==============================] - 0s 11ms/step - loss: 0.5987 - accuracy: 0.6702 - v
al_loss: 0.6391 - val_accuracy: 0.6250
Epoch 4/30
6/6 [==============================] - 0s 10ms/step - loss: 0.5788 - accuracy: 0.6702 - v
al_loss: 0.6162 - val_accuracy: 0.6250
Epoch 5/30
6/6 [==============================] - 0s 16ms/step - loss: 0.5576 - accuracy: 0.6806 - v
al_loss: 0.5877 - val_accuracy: 0.6042
Epoch 6/30
6/6 [==============================] - 0s 14ms/step - loss: 0.5378 - accuracy: 0.7016 - v
al_loss: 0.5614 - val_accuracy: 0.6875
Epoch 7/30
6/6 [==============================] - 0s 10ms/step - loss: 0.5134 - accuracy: 0.7539 - v
al_loss: 0.5344 - val_accuracy: 0.7083
Epoch 8/30
6/6 [==============================] - 0s 11ms/step - loss: 0.4923 - accuracy: 0.7539 - v
al_loss: 0.5075 - val_accuracy: 0.7708
Epoch 9/30
6/6 [==============================] - 0s 10ms/step - loss: 0.4752 - accuracy: 0.8010 - v
al_loss: 0.4756 - val_accuracy: 0.8125
Epoch 10/30
6/6 [==============================] - 0s 12ms/step - loss: 0.4483 - accuracy: 0.8063 - v
al_loss: 0.4689 - val_accuracy: 0.7917
Epoch 11/30
6/6 [==============================] - 0s 13ms/step - loss: 0.4320 - accuracy: 0.8115 - v
al_loss: 0.4472 - val_accuracy: 0.7917
Epoch 12/30
6/6 [==============================] - 0s 10ms/step - loss: 0.4164 - accuracy: 0.8220 - v
al_loss: 0.4251 - val_accuracy: 0.7917
Epoch 13/30
6/6 [==============================] - 0s 9ms/step - loss: 0.4086 - accuracy: 0.8272 - va
l_loss: 0.4142 - val_accuracy: 0.7708
Epoch 14/30
6/6 [==============================] - 0s 9ms/step - loss: 0.4012 - accuracy: 0.8220 - va
l_loss: 0.4215 - val_accuracy: 0.7708
Epoch 15/30
6/6 [==============================] - 0s 16ms/step - loss: 0.3854 - accuracy: 0.8377 - v
al_loss: 0.4013 - val_accuracy: 0.8333
Epoch 16/30
6/6 [==============================] - 0s 10ms/step - loss: 0.3841 - accuracy: 0.8377 - v
al_loss: 0.4006 - val_accuracy: 0.7917
Epoch 17/30
6/6 [==============================] - 0s 11ms/step - loss: 0.3736 - accuracy: 0.8377 - v
al_loss: 0.4076 - val_accuracy: 0.8125
Epoch 18/30
6/6 [==============================] - 0s 10ms/step - loss: 0.3734 - accuracy: 0.8482 - v
al_loss: 0.4035 - val_accuracy: 0.8125
Epoch 19/30
6/6 [==============================] - 0s 13ms/step - loss: 0.3587 - accuracy: 0.8586 - v
al_loss: 0.4132 - val_accuracy: 0.8125
Epoch 20/30
6/6 [==============================] - 0s 15ms/step - loss: 0.3603 - accuracy: 0.8377 - v
```

```
al_loss: 0.4093 - val_accuracy: 0.8125
Epoch 21/30
6/6 [==============================] - 0s 15ms/step - loss: 0.3504 - accuracy: 0.8534 - v
al_loss: 0.4000 - val_accuracy: 0.7917
Epoch 22/30
6/6 [==============================] - 0s 13ms/step - loss: 0.3476 - accuracy: 0.8586 - v
al_loss: 0.4084 - val_accuracy: 0.8125
Epoch 23/30
6/6 [==============================] - 0s 14ms/step - loss: 0.3420 - accuracy: 0.8639 - v
al_loss: 0.4097 - val_accuracy: 0.7917
Epoch 24/30
6/6 [==============================] - 0s 16ms/step - loss: 0.3383 - accuracy: 0.8534 - v
al_loss: 0.4071 - val_accuracy: 0.7917
Epoch 25/30
6/6 [==============================] - 0s 14ms/step - loss: 0.3365 - accuracy: 0.8586 - v
al_loss: 0.4209 - val_accuracy: 0.7917
Epoch 26/30
6/6 [==============================] - 0s 21ms/step - loss: 0.3299 - accuracy: 0.8639 - v
al_loss: 0.4218 - val_accuracy: 0.7917
Epoch 27/30
6/6 [==============================] - 0s 17ms/step - loss: 0.3298 - accuracy: 0.8586 - v
al_loss: 0.4243 - val_accuracy: 0.7708
Epoch 28/30
6/6 [==============================] - 0s 11ms/step - loss: 0.3190 - accuracy: 0.8482 - v
al_loss: 0.4422 - val_accuracy: 0.7708
Epoch 29/30
6/6 [==============================] - 0s 16ms/step - loss: 0.3247 - accuracy: 0.8482 - v
al_loss: 0.4301 - val_accuracy: 0.7917
Epoch 30/30
6/6 [==============================] - 0s 13ms/step - loss: 0.3165 - accuracy: 0.8586 - v
al_loss: 0.4313 - val_accuracy: 0.7917
```

In [26]:

```
# Predicting the Y
y_pred=model.predict(xtest)
```

```
2/2 [==============================] - 0s 2ms/step
```

In [27]:

```
y_pred.round()
```

Out[27]:

```
array([[0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.]
```

```
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]], dtype=float32)
```

In [28]:

```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classificat
ion_report, accuracy_score, f1_score,roc_auc_score
```

In [29]:

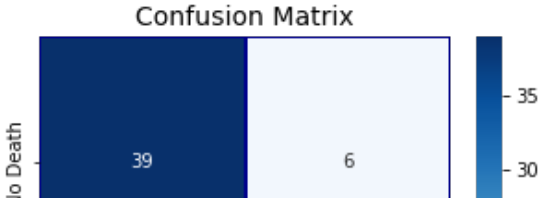
```
cm = confusion_matrix(ytest.round(),y_pred.round())
print("Train set Accuracy: ", accuracy_score(ytrain.round(), model.predict(xtrain).round
()))
print("Test set Accuracy: ", accuracy_score(ytest.round(), y_pred.round()))
```

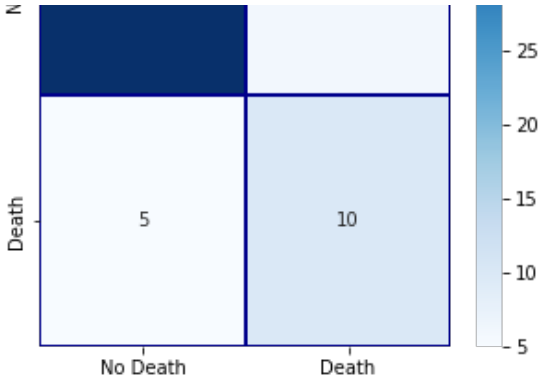
```
8/8 [==============================] - 0s 2ms/step
Train set Accuracy:  0.8410041841004184
Test set Accuracy:  0.8166666666666667
```

COnfusion Matrix for Model Evaluation

In [32]:

```
cf_matrix = confusion_matrix(ytest.round(), y_pred.round())
#cf_matrix
fig, (ax1) = plt.subplots(ncols=1, figsize=(5,5))
sns.heatmap(cf_matrix,
            xticklabels=['No Death', 'Death'],
            yticklabels=['No Death', 'Death'],
            annot=True,ax=ax1,
            linewidths=.2,linecolor="Darkblue", cmap="Blues")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```





In [35]:

```
from tensorflow.keras.layers import Dense, Dropout

model=Sequential()

model.add(Dense(90, input_dim = 12, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(40, activation='relu'))
model.add(Dropout(0.5))

# sigmoid output between 0 and 1
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 90)                1170

 dropout (Dropout)           (None, 90)                0

 dense_8 (Dense)             (None, 40)                3640

 dropout_1 (Dropout)         (None, 40)                0

 dense_9 (Dense)             (None, 1)                 41

=================================================================
Total params: 4,851
Trainable params: 4,851
Non-trainable params: 0
_________________________________________________________________
```

In [38]:

```
from keras.callbacks import History
from keras.callbacks import ModelCheckpoint, EarlyStopping

history = History()
History = model.fit(x=xtrain, y=ytrain, validation_data=(xtest, ytest), epochs=60, callb
acks=[history])
```

```
Epoch 1/60
8/8 [==============================] - 1s 23ms/step - loss: 0.7166 - accuracy: 0.4812 - v
al_loss: 0.6429 - val_accuracy: 0.7500
Epoch 2/60
8/8 [==============================] - 0s 7ms/step - loss: 0.6928 - accuracy: 0.5230 - va
l_loss: 0.6027 - val_accuracy: 0.7500
Epoch 3/60
8/8 [==============================] - 0s 6ms/step - loss: 0.6732 - accuracy: 0.6192 - va
l_loss: 0.5773 - val_accuracy: 0.7500
Epoch 4/60
8/8 [==============================] - 0s 8ms/step - loss: 0.6281 - accuracy: 0.6778 - va
l_loss: 0.5620 - val_accuracy: 0.7500
```

```
l_loss: 0.5020 - val_accuracy: 0.7500
Epoch 5/60
8/8 [==============================] - 0s 8ms/step - loss: 0.6227 - accuracy: 0.6736 - va
l_loss: 0.5499 - val_accuracy: 0.7500
Epoch 6/60
8/8 [==============================] - 0s 11ms/step - loss: 0.6194 - accuracy: 0.6736 - v
al_loss: 0.5414 - val_accuracy: 0.7500
Epoch 7/60
8/8 [==============================] - 0s 12ms/step - loss: 0.6150 - accuracy: 0.6695 - v
al_loss: 0.5326 - val_accuracy: 0.7500
Epoch 8/60
8/8 [==============================] - 0s 24ms/step - loss: 0.6337 - accuracy: 0.6611 - v
al_loss: 0.5230 - val_accuracy: 0.7500
Epoch 9/60
8/8 [==============================] - 0s 11ms/step - loss: 0.6169 - accuracy: 0.6653 - v
al_loss: 0.5203 - val_accuracy: 0.7500
Epoch 10/60
8/8 [==============================] - 0s 11ms/step - loss: 0.5994 - accuracy: 0.6778 - v
al_loss: 0.5162 - val_accuracy: 0.7500
Epoch 11/60
8/8 [==============================] - 0s 11ms/step - loss: 0.5958 - accuracy: 0.6695 - v
al_loss: 0.5083 - val_accuracy: 0.7500
Epoch 12/60
8/8 [==============================] - 0s 11ms/step - loss: 0.5931 - accuracy: 0.6862 - v
al_loss: 0.4994 - val_accuracy: 0.7500
Epoch 13/60
8/8 [==============================] - 0s 12ms/step - loss: 0.5806 - accuracy: 0.6820 - v
al_loss: 0.4928 - val_accuracy: 0.7500
Epoch 14/60
8/8 [==============================] - 0s 13ms/step - loss: 0.5810 - accuracy: 0.7029 - v
al_loss: 0.4861 - val_accuracy: 0.7500
Epoch 15/60
8/8 [==============================] - 0s 10ms/step - loss: 0.5827 - accuracy: 0.6904 - v
al_loss: 0.4805 - val_accuracy: 0.7667
Epoch 16/60
8/8 [==============================] - 0s 10ms/step - loss: 0.5673 - accuracy: 0.7071 - v
al_loss: 0.4765 - val_accuracy: 0.7667
Epoch 17/60
8/8 [==============================] - 0s 13ms/step - loss: 0.5619 - accuracy: 0.7113 - v
al_loss: 0.4679 - val_accuracy: 0.7833
Epoch 18/60
8/8 [==============================] - 0s 10ms/step - loss: 0.5658 - accuracy: 0.6946 - v
al_loss: 0.4576 - val_accuracy: 0.7833
Epoch 19/60
8/8 [==============================] - 0s 10ms/step - loss: 0.5825 - accuracy: 0.7322 - v
al_loss: 0.4534 - val_accuracy: 0.8000
Epoch 20/60
8/8 [==============================] - 0s 8ms/step - loss: 0.5478 - accuracy: 0.7364 - va
l_loss: 0.4502 - val_accuracy: 0.8000
Epoch 21/60
8/8 [==============================] - 0s 10ms/step - loss: 0.5142 - accuracy: 0.7573 - v
al_loss: 0.4394 - val_accuracy: 0.8000
Epoch 22/60
8/8 [==============================] - 0s 12ms/step - loss: 0.5495 - accuracy: 0.7322 - v
al_loss: 0.4246 - val_accuracy: 0.8000
Epoch 23/60
8/8 [==============================] - 0s 9ms/step - loss: 0.5049 - accuracy: 0.7280 - va
l_loss: 0.4197 - val_accuracy: 0.8167
Epoch 24/60
8/8 [==============================] - 0s 9ms/step - loss: 0.5186 - accuracy: 0.7406 - va
l_loss: 0.4206 - val_accuracy: 0.8333
Epoch 25/60
8/8 [==============================] - 0s 9ms/step - loss: 0.5217 - accuracy: 0.7322 - va
l_loss: 0.4093 - val_accuracy: 0.8333
Epoch 26/60
8/8 [==============================] - 0s 9ms/step - loss: 0.4945 - accuracy: 0.7699 - va
l_loss: 0.4074 - val_accuracy: 0.8333
Epoch 27/60
8/8 [==============================] - 0s 9ms/step - loss: 0.4920 - accuracy: 0.7824 - va
l_loss: 0.4046 - val_accuracy: 0.8667
Epoch 28/60
8/8 [==============================] - 0s 22ms/step - loss: 0.5003 - accuracy: 0.7782 - v
al_loss: 0.3887 - val_accuracy: 0.8333
```

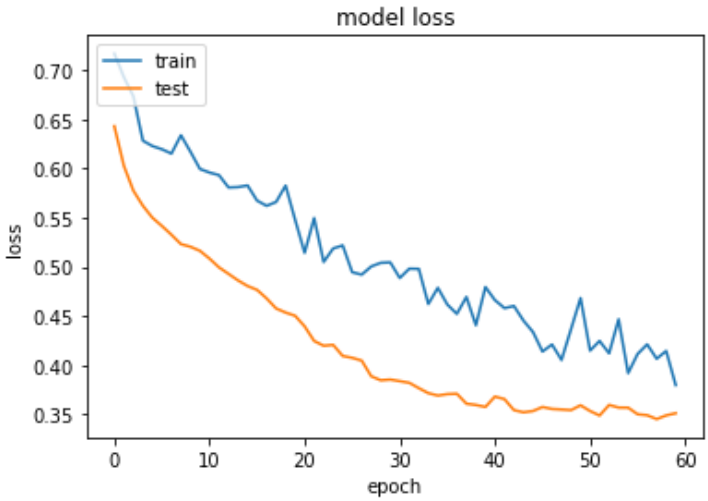
```
al_loss: 0.3667 - val_accuracy: 0.8333
Epoch 29/60
8/8 [==============================] - 0s 9ms/step - loss: 0.5040 - accuracy: 0.7824 - va
l_loss: 0.3846 - val_accuracy: 0.8333
Epoch 30/60
8/8 [==============================] - 0s 10ms/step - loss: 0.5045 - accuracy: 0.7824 - v
al_loss: 0.3853 - val_accuracy: 0.8500
Epoch 31/60
8/8 [==============================] - 0s 13ms/step - loss: 0.4886 - accuracy: 0.7490 - v
al_loss: 0.3837 - val_accuracy: 0.8500
Epoch 32/60
8/8 [==============================] - 0s 10ms/step - loss: 0.4981 - accuracy: 0.7782 - v
al_loss: 0.3820 - val_accuracy: 0.8500
Epoch 33/60
8/8 [==============================] - 0s 10ms/step - loss: 0.4979 - accuracy: 0.7657 - v
al_loss: 0.3766 - val_accuracy: 0.8500
Epoch 34/60
8/8 [==============================] - 0s 12ms/step - loss: 0.4621 - accuracy: 0.7992 - v
al_loss: 0.3715 - val_accuracy: 0.8500
Epoch 35/60
8/8 [==============================] - 0s 11ms/step - loss: 0.4786 - accuracy: 0.7866 - v
al_loss: 0.3691 - val_accuracy: 0.8500
Epoch 36/60
8/8 [==============================] - 0s 11ms/step - loss: 0.4616 - accuracy: 0.7824 - v
al_loss: 0.3706 - val_accuracy: 0.8667
Epoch 37/60
8/8 [==============================] - 0s 9ms/step - loss: 0.4521 - accuracy: 0.7908 - va
l_loss: 0.3709 - val_accuracy: 0.8667
Epoch 38/60
8/8 [==============================] - 0s 8ms/step - loss: 0.4693 - accuracy: 0.7699 - va
l_loss: 0.3608 - val_accuracy: 0.8667
Epoch 39/60
8/8 [==============================] - 0s 8ms/step - loss: 0.4406 - accuracy: 0.8033 - va
l_loss: 0.3595 - val_accuracy: 0.8500
Epoch 40/60
8/8 [==============================] - 0s 8ms/step - loss: 0.4794 - accuracy: 0.7531 - va
l_loss: 0.3575 - val_accuracy: 0.8500
Epoch 41/60
8/8 [==============================] - 0s 10ms/step - loss: 0.4661 - accuracy: 0.7866 - v
al_loss: 0.3680 - val_accuracy: 0.8667
Epoch 42/60
8/8 [==============================] - 0s 12ms/step - loss: 0.4579 - accuracy: 0.8033 - v
al_loss: 0.3655 - val_accuracy: 0.8667
Epoch 43/60
8/8 [==============================] - 0s 8ms/step - loss: 0.4601 - accuracy: 0.7866 - va
l_loss: 0.3543 - val_accuracy: 0.8667
Epoch 44/60
8/8 [==============================] - 0s 8ms/step - loss: 0.4452 - accuracy: 0.8075 - va
l_loss: 0.3521 - val_accuracy: 0.8667
Epoch 45/60
8/8 [==============================] - 0s 22ms/step - loss: 0.4337 - accuracy: 0.8326 - v
al_loss: 0.3534 - val_accuracy: 0.8667
Epoch 46/60
8/8 [==============================] - 0s 21ms/step - loss: 0.4138 - accuracy: 0.8285 - v
al_loss: 0.3574 - val_accuracy: 0.8667
Epoch 47/60
8/8 [==============================] - 0s 15ms/step - loss: 0.4210 - accuracy: 0.8243 - v
al_loss: 0.3555 - val_accuracy: 0.8500
Epoch 48/60
8/8 [==============================] - 0s 10ms/step - loss: 0.4052 - accuracy: 0.8243 - v
al_loss: 0.3549 - val_accuracy: 0.8500
Epoch 49/60
8/8 [==============================] - 0s 12ms/step - loss: 0.4367 - accuracy: 0.7741 - v
al_loss: 0.3543 - val_accuracy: 0.8500
Epoch 50/60
8/8 [==============================] - 0s 11ms/step - loss: 0.4682 - accuracy: 0.7908 - v
al_loss: 0.3592 - val_accuracy: 0.8667
Epoch 51/60
8/8 [==============================] - 0s 8ms/step - loss: 0.4148 - accuracy: 0.8117 - va
l_loss: 0.3534 - val_accuracy: 0.8500
Epoch 52/60
8/8 [==============================] - 0s 10ms/step - loss: 0.4246 - accuracy: 0.8075 - v
al_loss: 0.3488 - val_accuracy: 0.8500
```

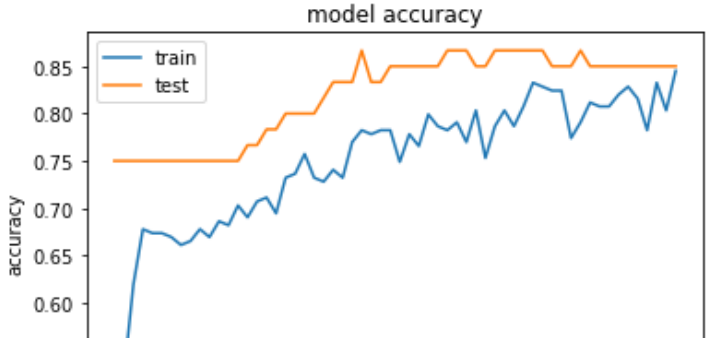
```
al_loss: 0.3466 - val_accuracy: 0.8500
Epoch 53/60
8/8 [==============================] - 0s 12ms/step - loss: 0.4120 - accuracy: 0.8075 - v
al_loss: 0.3595 - val_accuracy: 0.8500
Epoch 54/60
8/8 [==============================] - 0s 11ms/step - loss: 0.4466 - accuracy: 0.8201 - v
al_loss: 0.3569 - val_accuracy: 0.8500
Epoch 55/60
8/8 [==============================] - 0s 9ms/step - loss: 0.3919 - accuracy: 0.8285 - va
l_loss: 0.3568 - val_accuracy: 0.8500
Epoch 56/60
8/8 [==============================] - 0s 9ms/step - loss: 0.4114 - accuracy: 0.8159 - va
l_loss: 0.3501 - val_accuracy: 0.8500
Epoch 57/60
8/8 [==============================] - 0s 10ms/step - loss: 0.4211 - accuracy: 0.7824 - v
al_loss: 0.3491 - val_accuracy: 0.8500
Epoch 58/60
8/8 [==============================] - 0s 9ms/step - loss: 0.4065 - accuracy: 0.8326 - va
l_loss: 0.3452 - val_accuracy: 0.8500
Epoch 59/60
8/8 [==============================] - 0s 9ms/step - loss: 0.4144 - accuracy: 0.8033 - va
l_loss: 0.3489 - val_accuracy: 0.8500
Epoch 60/60
8/8 [==============================] - 0s 10ms/step - loss: 0.3797 - accuracy: 0.8452 - v
al_loss: 0.3511 - val_accuracy: 0.8500
```

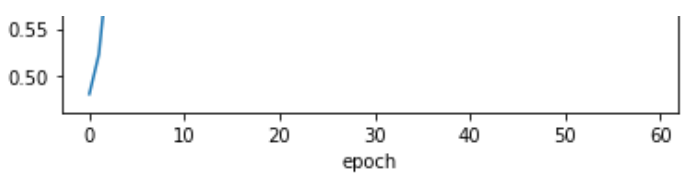
In [39]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```







In [ ]: