2. Classification_Multi_Layer_Perceptron

February 9, 2026

Author : Vaibhav Thakur

[44]:
```
# importing the packages
import pandas as pd
import numpy as np
cancer = pd.read_csv('/Users/sundaramvaibhav/Downloads/SV CODE/Python/DSA/
 ↪ovariantotal.csv')
cancer.head()
```

[44]:

	AFP	AG	Age	ALB	ALP	ALT	AST	BASO#	BASO%	BUN	…	PCT	\
0	3.58	19.36	47	45.4	56	11	24	0.01	0.30	5.35	…	0.09	
1	34.24	23.98	61	39.9	95	9	13	0.02	0.30	3.21	…	0.30	
2	1.50	18.40	39	45.4	77	9	18	0.03	0.60	3.80	…	0.13	
3	2.75	16.60	45	39.2	26	16	17	0.05	0.74	5.27	…	0.25	
4	2.36	19.97	45	35.0	47	21	27	0.01	0.10	4.89	…	0.28	

	PDW	PHOS	PLT	RBC	RDW	TBIL	TP	UA	TYPE
0	13.4	1.46	74	2.64	13.7	5.5	73.9	396.4	0
1	11.2	1.09	304	4.89	12.7	6.8	72.0	119.2	0
2	15.2	0.97	112	4.62	12.0	14.8	77.9	209.2	0
3	17.4	1.25	339	4.01	14.6	10.9	66.1	215.6	0
4	11.9	0.94	272	4.40	13.4	5.3	66.5	206.0	0

[5 rows x 50 columns]

[45]:
```
cancer.shape
```

[45]:
```
(349, 50)
```

[46]:
```
# separating dependent and independent variables - in this case, separating␣
 ↪features and labels
X = cancer.drop('TYPE', axis = 1)  # axis = 0 means row, axis = 1 means column
y = cancer['TYPE']
```

fit_transform(X)

fit(X): Calculates the minimum and maximum values for each feature (i.e., for each column in X).

transform(X): Applies the following formula to each value in X: $X_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$

[47]:
```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()
X_scale = min_max_scaler.fit_transform(X)
print(X_scale)
```

```
[[2.45578349e-03 4.85071876e-01 4.70588235e-01 … 8.37988827e-02
  7.60667904e-01 5.60447761e-01]
 [2.78074070e-02 6.55363067e-01 6.76470588e-01 … 1.20111732e-01
  7.25417440e-01 4.32835821e-02]
 [7.35908185e-04 4.49686694e-01 3.52941176e-01 … 3.43575419e-01
  8.34879406e-01 2.11194030e-01]
 …
 [1.83563615e-03 4.80280133e-01 6.47058824e-01 … 3.15642458e-01
  6.58627087e-01 2.11753731e-01]
 [1.01704165e-03 7.74788058e-01 2.20588235e-01 … 2.45810056e-01
  7.99628942e-01 3.53917910e-01]
 [8.26863129e-04 2.50645042e-01 3.52941176e-01 … 1.62011173e-01
  6.62337662e-01 1.65858209e-01]]
```

[48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size = 0.
 ↪20, random_state = 0)
```

[49]:
```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

# number of hidden layers is a hyperparameter, for our model we are choosing 2␣
 ↪hidden layers\
# number of neurins in the hidden layers is also a hyperparameter, we are␣
 ↪choosing 32 here
# for hidden layer we are choosing ReLU activation function and since this is a␣
 ↪binary classification problem, the activation function in the output layer␣
 ↪would
# be the sigmoid activation function
# Dense - every neuron is connected to every other neuron in the next layer

model = Sequential()
model.add(Dense(32, activation = 'relu', input_dim = 49))  # first hidden layer
model.add(Dense(32, activation = 'relu')) # second hidden layer
model.add(Dense(1, activation = 'sigmoid'))
```

```
/opt/anaconda3/envs/py311/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:106: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Given:

Dataset size = 349 samples

test_size=0.2 → 20% for testing, so Training set = 80% of 349 = 279 samples (approx.)

batch_size = 32

$$\text{Steps per epoch} = \left\lceil \frac{\text{Number of training samples}}{\text{batch size}} \right\rceil$$

Here, the steps per epoch are ceil(279/32) = ceilin(8.718) = 9

[50]:
```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train, batch_size = 32, epochs = 15)
```

```
Epoch 1/15
9/9             0s 1ms/step -
accuracy: 0.5771 - loss: 0.6677
Epoch 2/15
9/9             0s 1ms/step -
accuracy: 0.6523 - loss: 0.6426
Epoch 3/15
9/9             0s 1ms/step -
accuracy: 0.7240 - loss: 0.6223
Epoch 4/15
9/9             0s 1ms/step -
accuracy: 0.7348 - loss: 0.5958
Epoch 5/15
9/9             0s 1ms/step -
accuracy: 0.7491 - loss: 0.5717
Epoch 6/15
9/9             0s 1ms/step -
accuracy: 0.7563 - loss: 0.5462
Epoch 7/15
9/9             0s 1ms/step -
accuracy: 0.7634 - loss: 0.5225
Epoch 8/15
9/9             0s 1ms/step -
accuracy: 0.7849 - loss: 0.5027
Epoch 9/15
9/9             0s 1ms/step -
accuracy: 0.7957 - loss: 0.4833
Epoch 10/15
9/9             0s 1ms/step -
accuracy: 0.7849 - loss: 0.4705
Epoch 11/15
9/9             0s 1ms/step -
accuracy: 0.7921 - loss: 0.4632
Epoch 12/15
9/9             0s 1ms/step -
```

```
accuracy: 0.8065 - loss: 0.4516
Epoch 13/15
9/9             0s 1ms/step -
accuracy: 0.8136 - loss: 0.4394
Epoch 14/15
9/9             0s 1ms/step -
accuracy: 0.8351 - loss: 0.4310
Epoch 15/15
9/9             0s 1ms/step -
accuracy: 0.8280 - loss: 0.4276
```

[50]:
```
<keras.src.callbacks.history.History at 0x30c97d310>
```

[51]:
```
y_pred = model.predict(X_test)
y_pred
```

```
3/3             0s 8ms/step
```

[51]:
```
array([[0.71212935],
       [0.3241959 ],
       [0.73526543],
       [0.07589307],
       [0.89595556],
       [0.913263  ],
       [0.91836065],
       [0.1853667 ],
       [0.05707509],
       [0.7589335 ],
       [0.7627426 ],
       [0.21068908],
       [0.8901929 ],
       [0.3660035 ],
       [0.1087938 ],
       [0.61308557],
       [0.64536583],
       [0.8281699 ],
       [0.92275786],
       [0.40128064],
       [0.7224502 ],
       [0.7997707 ],
       [0.2184442 ],
       [0.8681305 ],
       [0.05847168],
       [0.71207905],
       [0.0281382 ],
       [0.19439803],
       [0.09591747],
       [0.7928628 ],
```

```
       [0.46582735],
       [0.4650779 ],
       [0.03876346],
       [0.23753178],
       [0.24673244],
       [0.01681486],
       [0.50227296],
       [0.7273765 ],
       [0.1985733 ],
       [0.2876878 ],
       [0.90933496],
       [0.7714142 ],
       [0.7440073 ],
       [0.8803099 ],
       [0.8801978 ],
       [0.04768071],
       [0.3434141 ],
       [0.7774195 ],
       [0.05449778],
       [0.23791675],
       [0.69496554],
       [0.7962158 ],
       [0.30984092],
       [0.37565976],
       [0.5390751 ],
       [0.77927464],
       [0.45242253],
       [0.847864  ],
       [0.33939445],
       [0.7447282 ],
       [0.90585446],
       [0.39410678],
       [0.1908182 ],
       [0.89400923],
       [0.073841  ],
       [0.4731929 ],
       [0.25203785],
       [0.70596164],
       [0.5554633 ],
       [0.25715017]], dtype=float32)
```

[52]:
```
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.astype(int)
import numpy as np
np.column_stack((y_pred, y_test))
```

```
[52]: array([[1, 0],
             [0, 0],
             [1, 1],
             [0, 0],
             [1, 1],
             [1, 1],
             [1, 1],
             [0, 0],
             [0, 0],
             [1, 1],
             [1, 1],
             [0, 0],
             [1, 1],
             [0, 0],
             [0, 0],
             [1, 0],
             [1, 0],
             [1, 1],
             [1, 1],
             [0, 1],
             [1, 1],
             [1, 1],
             [0, 0],
             [1, 0],
             [0, 0],
             [1, 0],
             [0, 0],
             [0, 0],
             [0, 0],
             [1, 1],
             [0, 0],
             [0, 0],
             [0, 0],
             [0, 0],
             [0, 0],
             [0, 0],
             [1, 1],
             [1, 1],
             [0, 0],
             [0, 0],
             [1, 1],
             [1, 1],
             [1, 1],
             [1, 1],
             [1, 1],
             [0, 0],
             [0, 0],
```

```
       [1, 1],
       [0, 0],
       [0, 0],
       [1, 0],
       [1, 1],
       [0, 1],
       [0, 0],
       [1, 0],
       [1, 1],
       [0, 0],
       [1, 1],
       [0, 0],
       [1, 0],
       [1, 1],
       [0, 1],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [1, 1],
       [0, 0]])
```

[53]:
```
# now we will check the accuracy
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[32  9]
 [ 3 26]]
0.8285714285714286
              precision    recall  f1-score   support

           0       0.91      0.78      0.84        41
           1       0.74      0.90      0.81        29

    accuracy                           0.83        70
   macro avg       0.83      0.84      0.83        70
weighted avg       0.84      0.83      0.83        70
```

Confusion Matrix :

	Actual Positive (1)	Actual Negative (0)
Predicted Positive (1)	TP	FP
Predicted Negative (0)	FN	TN

True Positive (TP): Model predicted 1 and actual is 1

False Positive (FP): Model predicted 1, but actual is 0 (Type I error)

False Negative (FN): Model predicted 0, but actual is 1 (Type II error)

True Negative (TN): Model predicted 0 and actual is 0

Regularization Techniques

1. Dropout
2. Early stopping
3. L1 and L2 regularization
4. Batch normalization

Now we will use different regularization techniques one by one in our model and evaluate the performance.

[58]:
```
# 1. Dropout

from tensorflow.keras.layers import Dropout
model = Sequential()

#model.add(Dropout(0.2, input_dim=49))
#model.add(Dense(32, activation='relu'))

model.add(Dense(32, activation='relu', input_dim = 49))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))

#Output layer
model.add(Dense(1, activation='sigmoid'))



model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train,batch_size=32, epochs=15)

# Evaluate performance
y_pred=model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.astype(int)
from sklearn.metrics import confusion_matrix,accuracy_score
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
Epoch 1/15

/opt/anaconda3/envs/py311/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:106: UserWarning: Do not pass an
```

```
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
9/9             0s 1ms/step -
accuracy: 0.5018 - loss: 0.7011
Epoch 2/15
9/9             0s 2ms/step -
accuracy: 0.5627 - loss: 0.6908
Epoch 3/15
9/9             0s 2ms/step -
accuracy: 0.5627 - loss: 0.6678
Epoch 4/15
9/9             0s 2ms/step -
accuracy: 0.6022 - loss: 0.6554
Epoch 5/15
9/9             0s 1ms/step -
accuracy: 0.6057 - loss: 0.6478
Epoch 6/15
9/9             0s 2ms/step -
accuracy: 0.6344 - loss: 0.6343
Epoch 7/15
9/9             0s 2ms/step -
accuracy: 0.7168 - loss: 0.6124
Epoch 8/15
9/9             0s 2ms/step -
accuracy: 0.7276 - loss: 0.5955
Epoch 9/15
9/9             0s 3ms/step -
accuracy: 0.7061 - loss: 0.5909
Epoch 10/15
9/9             0s 2ms/step -
accuracy: 0.7312 - loss: 0.5846
Epoch 11/15
9/9             0s 2ms/step -
accuracy: 0.7384 - loss: 0.5661
Epoch 12/15
9/9             0s 2ms/step -
accuracy: 0.7419 - loss: 0.5598
Epoch 13/15
9/9             0s 2ms/step -
accuracy: 0.7419 - loss: 0.5505
Epoch 14/15
9/9             0s 2ms/step -
accuracy: 0.7706 - loss: 0.5176
Epoch 15/15
9/9             0s 1ms/step -
accuracy: 0.7742 - loss: 0.5343
```

```
3/3             0s 9ms/step
[[32  9]
 [ 4 25]]
0.8142857142857143
```

[56]:
```python
# 2. Early stopping
from keras.callbacks import EarlyStopping
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=49))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
# watches the validation-loss metric during training.

# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    batch_size=32,
    epochs=100,
    validation_split=0.2, # 20 % of X_train is set aside internally for␣
 ↪validation (so effectively 64 % train + 16 % validation + 20 % test).
    callbacks=[early_stop],
    verbose=1
)

# Evaluate performance

y_pred=model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.astype(int)
from sklearn.metrics import confusion_matrix,accuracy_score
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
Epoch 1/100
```

```
/opt/anaconda3/envs/py311/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:106: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
7/7            0s 13ms/step -
accuracy: 0.4798 - loss: 0.7591 - val_accuracy: 0.4286 - val_loss: 0.7453
Epoch 2/100
7/7            0s 4ms/step -
accuracy: 0.4798 - loss: 0.7079 - val_accuracy: 0.4286 - val_loss: 0.6936
Epoch 3/100
7/7            0s 4ms/step -
accuracy: 0.6233 - loss: 0.6641 - val_accuracy: 0.6607 - val_loss: 0.6695
Epoch 4/100
7/7            0s 4ms/step -
accuracy: 0.5830 - loss: 0.6721 - val_accuracy: 0.6250 - val_loss: 0.6553
Epoch 5/100
7/7            0s 4ms/step -
accuracy: 0.6368 - loss: 0.6569 - val_accuracy: 0.6786 - val_loss: 0.6418
Epoch 6/100
7/7            0s 4ms/step -
accuracy: 0.6054 - loss: 0.6474 - val_accuracy: 0.6607 - val_loss: 0.6317
Epoch 7/100
7/7            0s 5ms/step -
accuracy: 0.6502 - loss: 0.6443 - val_accuracy: 0.7321 - val_loss: 0.6231
Epoch 8/100
7/7            0s 6ms/step -
accuracy: 0.6502 - loss: 0.6381 - val_accuracy: 0.7500 - val_loss: 0.6120
Epoch 9/100
7/7            0s 6ms/step -
accuracy: 0.6771 - loss: 0.6185 - val_accuracy: 0.7143 - val_loss: 0.5979
Epoch 10/100
7/7            0s 5ms/step -
accuracy: 0.7444 - loss: 0.5998 - val_accuracy: 0.7321 - val_loss: 0.5874
Epoch 11/100
7/7            0s 4ms/step -
accuracy: 0.7309 - loss: 0.5834 - val_accuracy: 0.7500 - val_loss: 0.5765
Epoch 12/100
7/7            0s 5ms/step -
accuracy: 0.7040 - loss: 0.5949 - val_accuracy: 0.7321 - val_loss: 0.5672
Epoch 13/100
7/7            0s 4ms/step -
accuracy: 0.7444 - loss: 0.5707 - val_accuracy: 0.7321 - val_loss: 0.5546
Epoch 14/100
7/7            0s 4ms/step -
accuracy: 0.7623 - loss: 0.5438 - val_accuracy: 0.7321 - val_loss: 0.5410
Epoch 15/100
7/7            0s 4ms/step -
accuracy: 0.7399 - loss: 0.5436 - val_accuracy: 0.7321 - val_loss: 0.5296
Epoch 16/100
7/7            0s 4ms/step -
accuracy: 0.7220 - loss: 0.5414 - val_accuracy: 0.7679 - val_loss: 0.5238
Epoch 17/100
```

```
7/7            0s 4ms/step -
accuracy: 0.7534 - loss: 0.5304 - val_accuracy: 0.7500 - val_loss: 0.5235
Epoch 18/100
7/7            0s 5ms/step -
accuracy: 0.7803 - loss: 0.5228 - val_accuracy: 0.7857 - val_loss: 0.5202
Epoch 19/100
7/7            0s 5ms/step -
accuracy: 0.7713 - loss: 0.4917 - val_accuracy: 0.7679 - val_loss: 0.5058
Epoch 20/100
7/7            0s 4ms/step -
accuracy: 0.7803 - loss: 0.5001 - val_accuracy: 0.7679 - val_loss: 0.5040
Epoch 21/100
7/7            0s 5ms/step -
accuracy: 0.7892 - loss: 0.4864 - val_accuracy: 0.7679 - val_loss: 0.4990
Epoch 22/100
7/7            0s 4ms/step -
accuracy: 0.7578 - loss: 0.5145 - val_accuracy: 0.7679 - val_loss: 0.4918
Epoch 23/100
7/7            0s 5ms/step -
accuracy: 0.7758 - loss: 0.4685 - val_accuracy: 0.7679 - val_loss: 0.4903
Epoch 24/100
7/7            0s 5ms/step -
accuracy: 0.7982 - loss: 0.4625 - val_accuracy: 0.7857 - val_loss: 0.4924
Epoch 25/100
7/7            0s 4ms/step -
accuracy: 0.8206 - loss: 0.4636 - val_accuracy: 0.8036 - val_loss: 0.4883
Epoch 26/100
7/7            0s 4ms/step -
accuracy: 0.8251 - loss: 0.4583 - val_accuracy: 0.7679 - val_loss: 0.4822
Epoch 27/100
7/7            0s 4ms/step -
accuracy: 0.8296 - loss: 0.4553 - val_accuracy: 0.8036 - val_loss: 0.4852
Epoch 28/100
7/7            0s 4ms/step -
accuracy: 0.7848 - loss: 0.4493 - val_accuracy: 0.7857 - val_loss: 0.4870
Epoch 29/100
7/7            0s 4ms/step -
accuracy: 0.8251 - loss: 0.4493 - val_accuracy: 0.7857 - val_loss: 0.4958
Epoch 30/100
7/7            0s 4ms/step -
accuracy: 0.8117 - loss: 0.4509 - val_accuracy: 0.7857 - val_loss: 0.4817
Epoch 31/100
7/7            0s 5ms/step -
accuracy: 0.8206 - loss: 0.4149 - val_accuracy: 0.7857 - val_loss: 0.4768
Epoch 32/100
7/7            0s 6ms/step -
accuracy: 0.8251 - loss: 0.4215 - val_accuracy: 0.7857 - val_loss: 0.4831
Epoch 33/100
```

```
7/7            0s 5ms/step -
accuracy: 0.8341 - loss: 0.4261 - val_accuracy: 0.7857 - val_loss: 0.4741
Epoch 34/100
7/7            0s 5ms/step -
accuracy: 0.8161 - loss: 0.4299 - val_accuracy: 0.7857 - val_loss: 0.4737
Epoch 35/100
7/7            0s 5ms/step -
accuracy: 0.8206 - loss: 0.4256 - val_accuracy: 0.7857 - val_loss: 0.4685
Epoch 36/100
7/7            0s 5ms/step -
accuracy: 0.8341 - loss: 0.4072 - val_accuracy: 0.7857 - val_loss: 0.4691
Epoch 37/100
7/7            0s 4ms/step -
accuracy: 0.8161 - loss: 0.3972 - val_accuracy: 0.7857 - val_loss: 0.4686
Epoch 38/100
7/7            0s 5ms/step -
accuracy: 0.8341 - loss: 0.3871 - val_accuracy: 0.7857 - val_loss: 0.4696
Epoch 39/100
7/7            0s 5ms/step -
accuracy: 0.8386 - loss: 0.4018 - val_accuracy: 0.7857 - val_loss: 0.4624
Epoch 40/100
7/7            0s 5ms/step -
accuracy: 0.8117 - loss: 0.4107 - val_accuracy: 0.7857 - val_loss: 0.4599
Epoch 41/100
7/7            0s 4ms/step -
accuracy: 0.8117 - loss: 0.4320 - val_accuracy: 0.7857 - val_loss: 0.4670
Epoch 42/100
7/7            0s 5ms/step -
accuracy: 0.8206 - loss: 0.3846 - val_accuracy: 0.8214 - val_loss: 0.4587
Epoch 43/100
7/7            0s 5ms/step -
accuracy: 0.8296 - loss: 0.4001 - val_accuracy: 0.7679 - val_loss: 0.4703
Epoch 44/100
7/7            0s 4ms/step -
accuracy: 0.8655 - loss: 0.3585 - val_accuracy: 0.7679 - val_loss: 0.4669
Epoch 45/100
7/7            0s 4ms/step -
accuracy: 0.8341 - loss: 0.3872 - val_accuracy: 0.8214 - val_loss: 0.4559
Epoch 46/100
7/7            0s 4ms/step -
accuracy: 0.8475 - loss: 0.3627 - val_accuracy: 0.8036 - val_loss: 0.4578
Epoch 47/100
7/7            0s 5ms/step -
accuracy: 0.8117 - loss: 0.4030 - val_accuracy: 0.7679 - val_loss: 0.4676
Epoch 48/100
7/7            0s 5ms/step -
accuracy: 0.8430 - loss: 0.3441 - val_accuracy: 0.7857 - val_loss: 0.4585
Epoch 49/100
```

```
7/7             0s 5ms/step -
accuracy: 0.8430 - loss: 0.3523 - val_accuracy: 0.7857 - val_loss: 0.4497
Epoch 50/100
7/7             0s 5ms/step -
accuracy: 0.8386 - loss: 0.3544 - val_accuracy: 0.8036 - val_loss: 0.4545
Epoch 51/100
7/7             0s 5ms/step -
accuracy: 0.8520 - loss: 0.3764 - val_accuracy: 0.8036 - val_loss: 0.4574
Epoch 52/100
7/7             0s 4ms/step -
accuracy: 0.8475 - loss: 0.3542 - val_accuracy: 0.8036 - val_loss: 0.4533
Epoch 53/100
7/7             0s 4ms/step -
accuracy: 0.8700 - loss: 0.3479 - val_accuracy: 0.7857 - val_loss: 0.4602
Epoch 54/100
7/7             0s 4ms/step -
accuracy: 0.8475 - loss: 0.3647 - val_accuracy: 0.8036 - val_loss: 0.4587
3/3             0s 9ms/step
[[31 10]
 [ 2 27]]
0.8285714285714286
```

[57]:
```python
# 3(a.) L1 regularization
from tensorflow.keras import layers, models, regularizers

model = Sequential()

model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.L1(0.
 ↪001), input_dim=49))
model.add(Dense(32, kernel_regularizer=regularizers.L1(0.001),␣
 ↪activation='relu'))

# Output layer
model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train,batch_size=32, epochs=15)


# Evaluate performance

y_pred=model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.astype(int)
```

```
from sklearn.metrics import␣
  ↪confusion_matrix,classification_report,accuracy_score
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
Epoch 1/15
```

```
/opt/anaconda3/envs/py311/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:106: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
9/9             0s 1ms/step -
accuracy: 0.4731 - loss: 1.0661
Epoch 2/15
9/9             0s 1ms/step -
accuracy: 0.5376 - loss: 1.0311
Epoch 3/15
9/9             0s 1ms/step -
accuracy: 0.5986 - loss: 0.9998
Epoch 4/15
9/9             0s 2ms/step -
accuracy: 0.7061 - loss: 0.9681
Epoch 5/15
9/9             0s 2ms/step -
accuracy: 0.7240 - loss: 0.9377
Epoch 6/15
9/9             0s 2ms/step -
accuracy: 0.7312 - loss: 0.9074
Epoch 7/15
9/9             0s 1ms/step -
accuracy: 0.7312 - loss: 0.8764
Epoch 8/15
9/9             0s 1ms/step -
accuracy: 0.7814 - loss: 0.8505
Epoch 9/15
9/9             0s 1ms/step -
accuracy: 0.7634 - loss: 0.8207
Epoch 10/15
9/9             0s 1ms/step -
accuracy: 0.7634 - loss: 0.7958
Epoch 11/15
9/9             0s 1ms/step -
accuracy: 0.7670 - loss: 0.7728
Epoch 12/15
9/9             0s 1ms/step -
accuracy: 0.7885 - loss: 0.7507
Epoch 13/15
```

```
9/9             0s 1ms/step -
accuracy: 0.7957 - loss: 0.7327
Epoch 14/15
9/9             0s 1ms/step -
accuracy: 0.7957 - loss: 0.7152
Epoch 15/15
9/9             0s 1ms/step -
accuracy: 0.7885 - loss: 0.7014
3/3             0s 8ms/step
[[33  8]
 [ 5 24]]
0.8142857142857143
```

[59]:
```
# 3(b.) L2 regularization

from tensorflow.keras import layers, models, regularizers

model = Sequential()
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.L2(0.
  ↪001), input_dim=49))
model.add(Dense(32, kernel_regularizer=regularizers.L2(0.001),␣
  ↪activation='relu'))

# Output layer
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy',␣
  ↪metrics=['accuracy'])
model.fit(X_train, y_train,batch_size=32, epochs=15)

# Evaluate performance
y_pred=model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.astype(int)
from sklearn.metrics import␣
  ↪confusion_matrix,classification_report,accuracy_score
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
Epoch 1/15

/opt/anaconda3/envs/py311/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:106: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

9/9             1s 2ms/step -
accuracy: 0.5627 - loss: 0.7518
```

```
Epoch 2/15
9/9             0s 2ms/step -
accuracy: 0.6523 - loss: 0.7263
Epoch 3/15
9/9             0s 2ms/step -
accuracy: 0.7168 - loss: 0.7061
Epoch 4/15
9/9             0s 2ms/step -
accuracy: 0.7419 - loss: 0.6869
Epoch 5/15
9/9             0s 2ms/step -
accuracy: 0.7312 - loss: 0.6624
Epoch 6/15
9/9             0s 2ms/step -
accuracy: 0.7455 - loss: 0.6404
Epoch 7/15
9/9             0s 2ms/step -
accuracy: 0.7491 - loss: 0.6172
Epoch 8/15
9/9             0s 2ms/step -
accuracy: 0.7455 - loss: 0.5962
Epoch 9/15
9/9             0s 1ms/step -
accuracy: 0.7670 - loss: 0.5764
Epoch 10/15
9/9             0s 2ms/step -
accuracy: 0.7778 - loss: 0.5622
Epoch 11/15
9/9             0s 2ms/step -
accuracy: 0.7706 - loss: 0.5472
Epoch 12/15
9/9             0s 2ms/step -
accuracy: 0.7778 - loss: 0.5313
Epoch 13/15
9/9             0s 2ms/step -
accuracy: 0.7921 - loss: 0.5223
Epoch 14/15
9/9             0s 1ms/step -
accuracy: 0.7885 - loss: 0.5151
Epoch 15/15
9/9             0s 2ms/step -
accuracy: 0.7957 - loss: 0.5023
3/3             0s 9ms/step
[[33  8]
 [ 3 26]]
0.8428571428571429
```

[60]:
```
# 4. Batch normalization

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout

# Define the Sequential model
model = Sequential()

# Input layer + 1st hidden layer
model.add(Dense(32, activation='relu', input_dim=49))
model.add(BatchNormalization())          # Normalize activations to stabilize␣
 ↪learning
model.add(Dropout(0.3))                  # Optional: Dropout for regularization

# 2nd hidden layer
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())          # Normalize the second layer's␣
 ↪activations
model.add(Dropout(0.3))

# Output layer
model.add(Dense(1, activation='sigmoid'))  # Binary classification output

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train,batch_size=32, epochs=15)


# Evaluate performance
y_pred=model.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)
y_pred = y_pred.astype(int)
from sklearn.metrics import␣
 ↪confusion_matrix,classification_report,accuracy_score
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
Epoch 1/15
```

```
/opt/anaconda3/envs/py311/lib/python3.11/site-
packages/keras/src/layers/core/dense.py:106: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
9/9             1s 2ms/step -
```

```
accuracy: 0.5341 - loss: 0.9395
Epoch 2/15
9/9             0s 2ms/step -
accuracy: 0.5842 - loss: 0.8425
Epoch 3/15
9/9             0s 2ms/step -
accuracy: 0.6595 - loss: 0.7191
Epoch 4/15
9/9             0s 2ms/step -
accuracy: 0.6989 - loss: 0.6722
Epoch 5/15
9/9             0s 2ms/step -
accuracy: 0.6774 - loss: 0.6503
Epoch 6/15
9/9             0s 2ms/step -
accuracy: 0.7419 - loss: 0.5851
Epoch 7/15
9/9             0s 2ms/step -
accuracy: 0.6703 - loss: 0.6351
Epoch 8/15
9/9             0s 2ms/step -
accuracy: 0.7276 - loss: 0.5806
Epoch 9/15
9/9             0s 2ms/step -
accuracy: 0.6882 - loss: 0.6359
Epoch 10/15
9/9             0s 2ms/step -
accuracy: 0.7348 - loss: 0.6166
Epoch 11/15
9/9             0s 2ms/step -
accuracy: 0.7563 - loss: 0.5342
Epoch 12/15
9/9             0s 2ms/step -
accuracy: 0.7742 - loss: 0.5309
Epoch 13/15
9/9             0s 2ms/step -
accuracy: 0.7455 - loss: 0.5500
Epoch 14/15
9/9             0s 2ms/step -
accuracy: 0.7563 - loss: 0.5450
Epoch 15/15
9/9             0s 2ms/step -
accuracy: 0.7384 - loss: 0.5604
3/3             0s 14ms/step
[[25 16]
 [ 1 28]]
0.7571428571428571
```