In [24]:

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
```

In [27]:

```
df=pd.read_csv('pima-indians-diabetes.csv')
```

In [28]:

```
df.info
```

Out[28]:

```
<bound method DataFrame.info of      6  148  72  35    0  33.6  0.627  50  1
0     1   85  66  29    0  26.6  0.351  31  0
1     8  183  64   0    0  23.3  0.672  32  1
2     1   89  66  23   94  28.1  0.167  21  0
3     0  137  40  35  168  43.1  2.288  33  1
4     5  116  74   0    0  25.6  0.201  30  0
..   ..  ...  ..  ..  ...   ...    ...  .. ..
762  10  101  76  48  180  32.9  0.171  63  0
763   2  122  70  27    0  36.8  0.340  27  0
764   5  121  72  23  112  26.2  0.245  30  0
765   1  126  60   0    0  30.1  0.349  47  1
766   1   93  70  31    0  30.4  0.315  23  0

[767 rows x 9 columns]>
```

In [63]:

```
import numpy as np

# Load the CSV file
data = np.loadtxt('pima-indians-diabetes.csv', delimiter=',', skiprows=1)
# Adjust `skiprows` if there's a header

print(data)
```

```
[[1.00e+00 8.50e+01 6.60e+01 ... 3.51e-01 3.10e+01 0.00e+00]
 [8.00e+00 1.83e+02 6.40e+01 ... 6.72e-01 3.20e+01 1.00e+00]
 [1.00e+00 8.90e+01 6.60e+01 ... 1.67e-01 2.10e+01 0.00e+00]
 ...
 [5.00e+00 1.21e+02 7.20e+01 ... 2.45e-01 3.00e+01 0.00e+00]
 [1.00e+00 1.26e+02 6.00e+01 ... 3.49e-01 4.70e+01 1.00e+00]
 [1.00e+00 9.30e+01 7.00e+01 ... 3.15e-01 2.30e+01 0.00e+00]]
```

In [64]:

```
df.columns
```

```
Out[64]: Index(['6', '148', '72', '35', '0', '33.6', '0.627', '50', '1'], dtype='object')
```

```
In [65]: X=df.iloc[:,0:-1].values
```

```
In [66]: y=df.iloc[:,8].values
```

```
In [67]: model=Sequential()
         model.add(Dense(12,input_dim=8, activation='relu'))
         model.add(Dense(8, activation='relu'))
         model.add(Dense(1, activation='sigmoid'))

         #####################################################################################
         #The code initializes a sequential neural network model and adds a dense layer with 12 units,
         #expecting 8 input features, and using the ReLU activation function.
         # ReLU is Rectified Linear Unit
         #Leaky ReLU: Allows a small, non-zero gradient when the input is negative.
         #Parametric ReLU (PReLU): Similar to Leaky ReLU, but the slope for negative inputs is learned during training.
         #ELU (Exponential Linear Unit): Outputs negative values for negative inputs, helping to reduce bias shift.
         #SELU (Scaled Exponential Linear Unit): Self-normalizing variant of ELU that can improve performance in deep networks
         #Sigmoid: A function that maps inputs to a range between 0 and 1.
         #Tanh: Maps inputs to a range between -1 and 1, centering the data.
         ##################################################################################################
         model.compile(loss='binary_crossentropy' ,optimizer='adam', metrics=['accuracy'])
         model.fit(X,y, epochs=150, batch_size=10)
         _, accuracy=model.evaluate(X, y)
         print('Accuracy: %.2f ' %(accuracy*100))
         from ann_visualizer. visualize import ann_viz;
         ann_viz(model , title="My first neural network")
```

```
Epoch 1/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4690 - loss: 6.4709
Epoch 2/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5979 - loss: 3.2157
Epoch 3/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6524 - loss: 1.7781
Epoch 4/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6324 - loss: 1.3866
Epoch 5/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6541 - loss: 1.0479
Epoch 6/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6538 - loss: 0.7856
Epoch 7/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6002 - loss: 0.7366
Epoch 8/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6595 - loss: 0.7183
Epoch 9/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7014 - loss: 0.6576
Epoch 10/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6438 - loss: 0.7137
Epoch 11/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6647 - loss: 0.7030
Epoch 12/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6707 - loss: 0.6565
Epoch 13/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6859 - loss: 0.6554
Epoch 14/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6706 - loss: 0.6490
Epoch 15/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6473 - loss: 0.6359
Epoch 16/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7074 - loss: 0.5967
Epoch 17/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6493 - loss: 0.6753
Epoch 18/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6675 - loss: 0.6864
Epoch 19/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6361 - loss: 0.6640
Epoch 20/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6277 - loss: 0.6576
Epoch 21/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7127 - loss: 0.5706
```

```
Epoch 148/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7687 - loss: 0.4815
Epoch 149/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7543 - loss: 0.5254
Epoch 150/150
77/77 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7669 - loss: 0.4818
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7480 - loss: 0.4905
Accuracy: 77.05
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[67], line 21
     19 _, accuracy=model.evaluate(X, y)
     20 print('Accuracy: %.2f ' %(accuracy*100))
---> 21 from ann_visualizer. visualize import ann_viz;
     22 ann_viz(model , title="My first neural network")

ModuleNotFoundError: No module named 'ann_visualizer'
```

In [ ]:

```
model=Sequential()
model.add(Dense(12,input_dim=8, activation='relu'))
model.add(Dense(8, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy' ,optimizer='adam', metrics=['accuracy'])
model.fit(X,y, epochs=200, batch_size=10)
#change in epoch value
_, accuracy=model.evaluate(X, y)
print('Accuracy: %.2f ' %(accuracy*100))
from ann_visualizer. visualize import ann_viz;
ann_viz(model , title="My first neural network")
```

In [ ]:

```
import sys
print(sys.executable)
```

In [ ]:

```
import os
os.environ["PATH"] += os.pathsep + r"C:\Program Files\Graphviz\bin"
# Adjust this path as needed
```

In [ ]:

```
import tensorflow as tf
tf.get_logger().setLevel('INFO')
```

In [ ]:

```
from tensorflow.keras.utils import plot_model

plot_model(model, to_file='al.png', show_shape=True, show_layer_names=True)
```

In [ ]:

```
X.shape
```

In [ ]:

```
y.shape
```

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
```

In [ ]:

```
model=Sequential()
```

In [ ]:

```
#input layer
model.add(Dense(12, activation='relu'))  #hidden layer
model.add(Dense(8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(8, activation='sigmoid'))
```

In [ ]:

```
model.compile(loss="binary_crossentropy", optimizer='adam', metrics=['accuracy'])
#This code configures the Keras model for training with binary cross-entropy as the loss function,
#Adam as the optimizer, and accuracy as the evaluation metric.
```

In [68]:

```
model.fit(X,y, epochs=200, batch_size=15)
```

```
Epoch 1/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7915 - loss: 0.4439
Epoch 2/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7973 - loss: 0.4560
Epoch 3/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7443 - loss: 0.4787
Epoch 4/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7839 - loss: 0.4565
Epoch 5/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7586 - loss: 0.4760
Epoch 6/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7982 - loss: 0.4925
Epoch 7/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7745 - loss: 0.4540
Epoch 8/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7708 - loss: 0.4744
Epoch 9/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7750 - loss: 0.4691
Epoch 10/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7669 - loss: 0.4761
Epoch 11/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7643 - loss: 0.4886
Epoch 12/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7443 - loss: 0.4869
Epoch 13/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7759 - loss: 0.4628
Epoch 14/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7713 - loss: 0.4741
Epoch 15/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7702 - loss: 0.4750
Epoch 16/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7614 - loss: 0.4942
Epoch 17/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7551 - loss: 0.4789
Epoch 18/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7573 - loss: 0.4823
Epoch 19/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7750 - loss: 0.4559
Epoch 20/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7648 - loss: 0.4891
Epoch 21/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7575 - loss: 0.4855
```

```
Epoch 169/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7614 - loss: 0.6036
Epoch 170/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7682 - loss: 0.4529
Epoch 171/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7749 - loss: 0.4915
Epoch 172/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7771 - loss: 0.4470
Epoch 173/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8307 - loss: 0.4125
Epoch 174/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7565 - loss: 0.4682
Epoch 175/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7980 - loss: 0.4192
Epoch 176/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7822 - loss: 0.4343
Epoch 177/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8073 - loss: 0.4213
Epoch 178/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7939 - loss: 0.4300
Epoch 179/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8199 - loss: 0.4321
Epoch 180/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8232 - loss: 0.3999
Epoch 181/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8046 - loss: 0.3936
Epoch 182/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8063 - loss: 0.4146
Epoch 183/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8067 - loss: 0.4241
Epoch 184/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7802 - loss: 0.4395
Epoch 185/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8070 - loss: 0.4130
Epoch 186/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7994 - loss: 0.4356
Epoch 187/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7732 - loss: 0.4379
Epoch 188/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7779 - loss: 0.4677
Epoch 189/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7815 - loss: 0.4378
```

```
Epoch 190/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7570 - loss: 0.4488
Epoch 191/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8166 - loss: 0.4203
Epoch 192/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7859 - loss: 0.4434
Epoch 193/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7778 - loss: 0.4631
Epoch 194/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7582 - loss: 0.4551
Epoch 195/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8057 - loss: 0.4214
Epoch 196/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7927 - loss: 0.4399
Epoch 197/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7492 - loss: 0.4905
Epoch 198/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7981 - loss: 0.4325
Epoch 199/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7853 - loss: 0.4449
Epoch 200/200
52/52 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7740 - loss: 0.4516
```

```
Out[68]: <keras.src.callbacks.history.History at 0x256fd682220>
```

```
In [69]: model.evaluate(X,y)
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7742 - loss: 0.4391
```

```
Out[69]: [0.41424041986465454, 0.7992177605628967]
```

```
In [70]: model.summary()
         #trains the Keras model on the input data X and target labels y
         #for 200 epochs using batches of 15 samples at a time.
```

Model: "sequential_7"

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense_20 (Dense)                     │ (None, 12)                  │             108 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_21 (Dense)                     │ (None, 8)                   │             104 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_22 (Dense)                     │ (None, 1)                   │               9 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 665 (2.60 KB)
 Trainable params: 221 (884.00 B)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 444 (1.74 KB)
```

```
In [ ]:
```