In [2]:
```
import tensorflow as tf
from tensorflow import keras
```

In [3]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential ([
    Dense (units = 5, input_shape = (3,), activation = 'relu'),
    Dense (units = 2, activation = 'softmax')
])
```

# OR

In [4]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
layers = [
    Dense (units = 5, input_shape = (3,), activation = 'relu'),
    Dense (units = 2, activation = 'softmax')
]
model = Sequential (layers)
```

# OR

In [5]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential ()
model.add(Dense (units = 4, input_shape = (3,), activation = 'relu')),
model.add(Dense (units = 2, activation = 'softmax'))
```

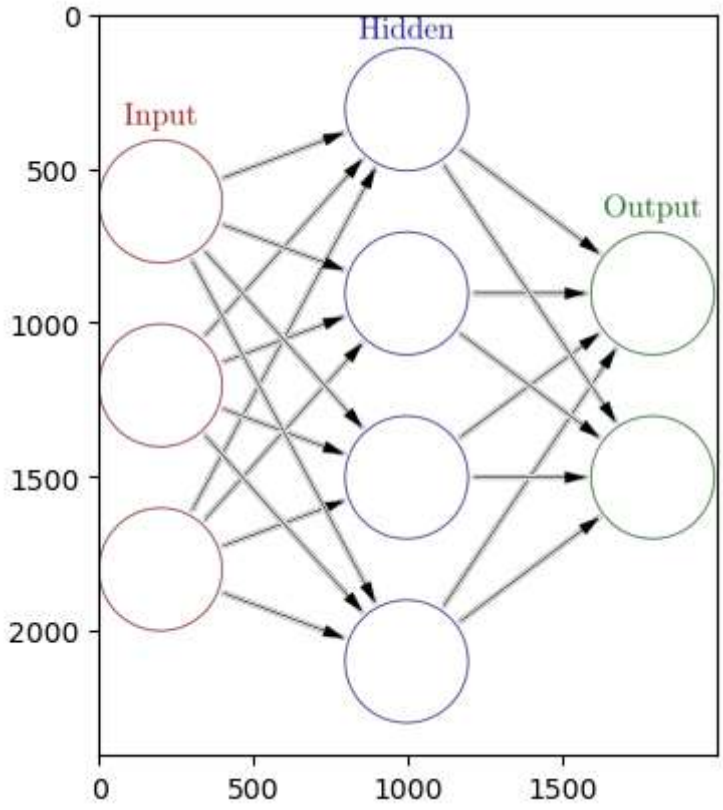
In [6]:
```
import numpy as np
a= np.array ([0,0,1,1,1,1,1,1,1])
print(a.shape)
```

```
(9,)
```

In [7]:
```
import numpy as np
from scipy import *
import matplotlib.pyplot as plt
%matplotlib inline
import imageio
imag = np.expand_dims (imageio.imread("neural_network.png"),0)
plt.imshow (imag[0])
```

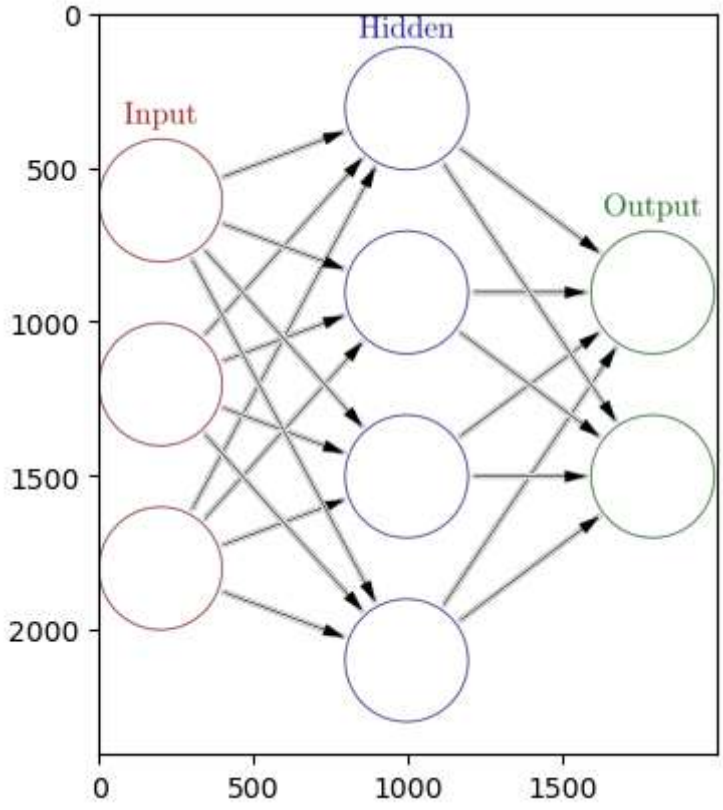
```
C:\Users\shalini193091\.conda\envs\tf\lib\site-packages\ipykernel_launcher.py:6: Deprecationwarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and make this warning dissapear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x20139502608>
```




In [8]:
```
import matplotlib.image as mpimg
img = mpimg.imread ("neural_network.png")
plt.imshow(img)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x2013c873f08>
```



In [9]:
```
from PIL import Image
image = Image.open("neural_network.png")
print (image.format)
print (image.size)
print (image.mode)
image.show()
```

```
PNG
(2000, 2405)
RGBA
```

In [28]:
```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam
from keras import metrics

model = Sequential ([
    Dense (units = 16, input_shape = (1,), activation = 'relu'),
    Dense (units = 32, activation = 'relu'),
    #Dense (units = 32, activation = 'relu'),
    #Dense (units = 16, activation = 'relu'),
    Dense (units = 2, activation = 'sigmoid')
])

model.compile(
    optimizer = Adam (learning_rate = 0.0001),
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)
```

In [29]:
```
import numpy as np
from random import randint
from sklearn.preprocessing import MinMaxScaler
train_labels = []
train_samples = []

for i in range (1000):
    random_younger = randint (13,64)
    train_samples.append(random_younger)
    train_labels.append(0)

    random_older = randint (65,100)
    train_samples.append(random_older)
    train_labels.append(1)

for i in range (50):
    random_younger = randint (13,64)
    train_samples.append(random_younger)
    train_labels.append(1)

    random_older = randint (65,100)
    train_samples.append(random_older)
    train_labels.append(0)

#print raw data
for i in train_samples:
    print (i)
```

```
train_labels = np.array (train_labels)
train_samples = np.array (train_samples)

scaler = MinMaxScaler(feature_range = (0,1))
scaled_train_samples = scaler.fit_transform ((train_samples).reshape(-1,1))
```

58
89
48
73
36
71
61
89
50
74
30
83
47
100
17
88
17
66
18
75
62
87
15
99
48
73
27
71
31
92
13
71
60
65
48
72
33
88
41
99
37
93
55
89
27
73
27
75
19
94
16
86
36
97
24
67
64
65
21
76

15
89
17
66
34
94
39
100
23
89
51
90
35
67
58
66
18
68
43
67
36
87
63
90
42
90
36
75
15
90
45
73
35
66
40
100
29
92
28
68
39
87
61
78
17
76
41
96
35
78
16
67
33
97
63
79
37
67
21
68

39
90
52
69
53
90
40
92
44
85
18
97
32
68
14
72
35
70
14
69
39
81
13
86
58
98
64
69
37
95
30
87
34
83
38
98
33
70
14
79
20
76
36
73
21
73
19
66
32
98
14
89
23
67
15
99
13
100
32
87

63
94
46
87
34
73
64
73
34
67
57
88
52
74
63
98
57
78
32
67
59
87
21
90
22
88
57
95
13
92
26
87
37
92
24
83
59
91
28
65
47
69
62
81
60
91
29
74
38
90
35
86
24
77
28
94
44
97
62
97

49
66
51
84
19
94
47
87
17
81
51
75
20
88
21
82
17
68
29
76
25
88
14
91
23
85
49
84
29
78
62
68
61
79
34
95
50
86
17
90
32
68
22
77
19
89
36
66
64
72
36
94
36
78
18
82
24
69
56
74

64
92
17
94
51
75
54
86
34
94
37
91
56
66
15
85
21
72
35
96
14
99
40
71
37
77
25
65
52
93
30
91
63
93
54
80
23
95
59
87
17
70
20
69
28
88
39
97
61
72
64
67
27
82
46
80
31
90
37
79

23
95
21
80
34
70
47
78
37
79
20
89
43
84
31
67
21
97
23
82
15
95
56
98
14
74
45
92
31
67
44
67
13
99
64
95
47
66
40
96
36
97
21
98
33
99
20
72
56
66
45
98
40
69
29
86
43
87
23
87

54
93
45
72
64
70
61
83
37
91
27
83
32
76
29
100
15
90
50
93
40
90
25
96
36
73
48
74
25
74
34
80
49
96
39
79
41
94
36
95
35
94
39
81
36
69
13
80
14
83
55
68
49
99
46
72
55
100
45
70

18
80
14
66
19
76
61
69
53
90
14
73
60
82
42
80
39
71
49
100
62
98
43
70
31
65
26
67
15
75
56
85
54
74
45
98
34
97
45
96
52
66
22
80
14
91
35
70
64
77
30
91
58
88
49
71
47
76
58
100

32
84
13
67
47
67
62
74
18
82
47
88
39
100
18
72
54
88
21
94
17
79
15
91
16
73
26
96
22
98
43
77
53
69
19
92
19
72
62
77
52
92
26
72
59
94
64
86
37
85
25
97
19
86
36
87
46
97
14
81

14
93
38
76
31
65
64
100
22
99
31
90
26
91
22
66
42
71
19
66
17
92
37
91
54
83
63
79
40
88
27
91
44
97
34
86
57
99
30
81
60
76
56
89
26
87
48
90
26
100
46
70
38
85
47
84
30
86
45
95

23
83
44
94
13
91
45
72
23
81
32
92
14
96
32
89
63
94
18
83
64
90
19
75
13
79
37
66
53
72
37
94
41
93
42
79
52
93
40
73
36
73
27
87
56
75
49
78
41
84
16
86
64
71
17
78
13
79
17
73

55
92
37
65
44
68
28
92
19
92
25
79
36
82
31
78
31
85
37
69
35
70
40
83
52
68
59
70
28
73
42
91
20
79
49
76
53
82
59
90
54
85
36
73
63
93
30
97
62
84
29
98
21
77
34
67
17
92
43
70

17
98
36
98
15
73
21
86
26
94
19
93
58
99
41
100
24
73
55
81
64
67
26
75
36
68
35
93
53
73
35
93
48
69
18
76
62
66
22
100
37
83
28
100
22
77
29
92
52
96
31
84
13
66
41
82
23
98
52
81

31
97
43
96
41
72
41
88
50
72
18
99
57
85
55
70
45
94
41
82
26
98
17
99
16
89
40
78
29
88
40
97
61
99
55
72
40
66
46
80
36
95
38
74
37
100
54
86
40
89
31
83
42
71
16
72
33
68
46
87

62
71
31
69
63
99
42
91
34
92
14
89
38
75
17
94
20
97
63
82
34
90
53
73
56
66
59
72
51
80
30
100
63
91
18
79
54
76
15
89
44
83
38
72
63
98
19
76
42
77
49
85
63
69
28
68
43
88
15
91

16
80
27
96
41
68
45
86
51
97
25
85
21
69
21
87
21
84
27
89
29
82
28
94
15
68
44
90
49
92
51
92
43
85
41
100
24
72
20
69
27
74
39
77
31
80
31
83
38
74
34
94
46
96
39
72
18
88
15
85

47
85
28
79
34
99
41
96
56
86
59
96
49
95
57
66
21
82
63
76
38
96
15
85
63
93
45
78
52
71
47
75
14
72
58
87
34
67
34
96
15
87
47
80
61
65
35
76
63
93
50
71
34
96
52
92
13
95
61
75

61
95
46
94
47
79
38
86
45
89
30
70
53
73
30
97
63
67
22
68
29
66
55
96
49
80
34
98
52
79
59
95
48
98
28
98
45
79
14
82
49
95
30
76
34
92
51
91
47
91
53
79
52
67
35
90
48
99
22
87

32
97
62
92
17
99
17
78
14
66
39
70
25
70
20
68
63
66
41
74
44
91
57
65
19
76
58
93
57
89
34
68
26
68
29
96
57
94
61
74
18
87
45
98
55
83
17
84
47
65
54
98
33
96
19
89
33
98
52
82

59
79
42
92
21
97
31
68
53
90
39
84
56
88
48
81
57
92
24
84
49
68
27
80
40
77
64
72
59
90
29
69
24
73
64
80
35
82
42
74
29
88
22
77
53
68
13
96
60
88
20
84
15
94
54
97
18
66
24
67

41
71
36
93
22
69
44
75
34
69
29
68
53
88
16
73
43
96
15
86
50
75
56
75
32
70
47
95
64
69
28
100
18
71
38
82
46
66
19
68
54
74
43
78
27
73
51
87
47
83
26
66
52
100
19
71
41
97
27
95

61
100
60
84
37
71
18
69
64
78
20
96
64
94
47
91
57
82
37
71
50
90
45
74
36
99
27
81
36
84
38
75
30
85
48
66
45
93
55
82
62
77
25
80
49
98
45
84
28
97
61
73
35
80
47
65
50
73
35
73

20
65
51
82
34
79
64
76
58
82
19
88
42
77
24
95
59
78
34
66
19
98
40
88
63
91
26
74
24
69
31
91
25
70
53
73
16
89
21
71
15
84
57
97
62
90
17
80
20
88
32
69
32
99
27
87
15
76
49
71

32
80
42
74
24
82
32
82
23
66
54
90
55
74
20
72
48
95
15
99
59
69
59
97
22
72
52
90
25
78
39
82
13
68
25
67
43
80
17
68
57
79
39
93
37
68
64
83
37
81
52
66
30
92
32
79
20
81
55
71

50
69
30
66
25
76
45
95
52
99
20
93
22
70
33
79
42
94
36
71
61
68
28
88
14
98
58
95
19
73
22
81
42
87
63
88
51
83
28
82
62
72
48
70
16
91
29
96
17
97
62
81
27
91
16
76
14
82
24
69

20
67
19
91
55
99
18
96
14
76
54
80
23
75
44
93
32
65
48
100
24
87
59
70
56
80
30
98
40
83
55
65
45
92
24
65
34
92
57
80
20
95
14
99
51
97
28
82
61
89
33
83
26
83
41
80
39
91
36
78

42
79
20
83
59
97
18
84
61
75
60
78
26
71
64
93
21
80
59
95
39
95
64
73
20
82
64
93
58
76
37
80
37
79
25
70
20
74
48
79
55
71
35
69
56
66
22
66
30
95
39
93
35
98
27
89
56
81
39
72

32
79
37
76
50
92
24
77
54
87
39
65
34
85
15
93
54
98
16
97
18
91
26
76
37
87
19
88
39
67
60
84
47
91
35
76
48
75
23
78
55
78
63
99
30
65
63
91
32
65
18
100
53
67
45
65
16
69
13
73

20
70
47
75
55
94
54
72
42
84
41
66
50
97
53
83
63
71
19
76
45
80
42
98
44
86
55
92
17
68
50
96
52
77
42
93
24
69
36
70
61
68
21
78
29
68
59
100
27
84
53
91
24
97
29
96
15
97
21
95

29
97
64
79
58
78
58
71
22
89
50
92
24
85
16
94
15
82
26
81
59
71
57
86
38
87
24
75
28
83
64
94
39
95
21
91
62
83
26
72
40
97
36
95
54
84
48
91
37
99
28
73
18
70
51
76
32
79
19
71

19
70
24
65
34
84
29
95
34
92
35
85
36
65
55
78
62
78
19
100
60
87
23
81
49
80
39
80
56
90
59
89
50
72
19
85
22
69
33
93
39
83
36
87
13
82
48
90
46
91
62
69
17
94
63
90
50
69
33
68

58
100
56
98
48
94
47
96
40
89
28
95
34
82
30
95
35
98
18
67
41
81
57
90
24
68
51
95
16
99
13
95
18
72
54
97
63
66
44
67
35
66
49
85
62
81
63
87
43
91
43
100
29
100
14
76
17
88
32
99

15
76
46
67
43
72
16
85
19
98
60
67
20
85
49
75
18
71
50
90
26
87
17
85
32
75
32
85
49
74
44
90
60
81
16
96
16
91
45
76
40
81
24
77
16
97
59
85
35
65
26
83
59
93
39
77
29
73
46
93

15
94
38
96
27
66
40
81
22
92
48
67
38
74
54
93
47
98
33
76
37
91
62
65
32
94
47
94
48
84
57
92
28
87
60
73
15
76
47
78
45
65
16
80
27
84
27
67
57
73
42
93
49
95
21
97
47
93
38
95

In [30]:

```
model.fit (
    x = scaled_train_samples,
    y = train_labels,
    epochs = 30,
    batch_size = 10,
    shuffle = False,
    verbose = 2
)
```

```
Epoch 1/30
210/210 - 1s - loss: 0.6707 - accuracy: 0.6900 - 840ms/epoch - 4ms/step
Epoch 2/30
210/210 - 0s - loss: 0.6401 - accuracy: 0.8233 - 270ms/epoch - 1ms/step
Epoch 3/30
210/210 - 0s - loss: 0.6045 - accuracy: 0.8490 - 249ms/epoch - 1ms/step
Epoch 4/30
210/210 - 0s - loss: 0.5644 - accuracy: 0.8629 - 308ms/epoch - 1ms/step
Epoch 5/30
210/210 - 0s - loss: 0.5238 - accuracy: 0.8733 - 239ms/epoch - 1ms/step
Epoch 6/30
210/210 - 0s - loss: 0.4838 - accuracy: 0.8800 - 293ms/epoch - 1ms/step
Epoch 7/30
210/210 - 0s - loss: 0.4473 - accuracy: 0.8895 - 385ms/epoch - 2ms/step
Epoch 8/30
210/210 - 0s - loss: 0.4157 - accuracy: 0.8938 - 369ms/epoch - 2ms/step
Epoch 9/30
210/210 - 0s - loss: 0.3894 - accuracy: 0.8995 - 303ms/epoch - 1ms/step
Epoch 10/30
210/210 - 0s - loss: 0.3680 - accuracy: 0.9033 - 283ms/epoch - 1ms/step
Epoch 11/30
210/210 - 0s - loss: 0.3508 - accuracy: 0.9076 - 379ms/epoch - 2ms/step
Epoch 12/30
210/210 - 0s - loss: 0.3371 - accuracy: 0.9095 - 385ms/epoch - 2ms/step
Epoch 13/30
210/210 - 0s - loss: 0.3260 - accuracy: 0.9119 - 306ms/epoch - 1ms/step
Epoch 14/30
210/210 - 0s - loss: 0.3171 - accuracy: 0.9133 - 277ms/epoch - 1ms/step
Epoch 15/30
210/210 - 0s - loss: 0.3098 - accuracy: 0.9138 - 237ms/epoch - 1ms/step
Epoch 16/30
210/210 - 0s - loss: 0.3038 - accuracy: 0.9152 - 274ms/epoch - 1ms/step
Epoch 17/30
210/210 - 0s - loss: 0.2988 - accuracy: 0.9176 - 358ms/epoch - 2ms/step
Epoch 18/30
210/210 - 0s - loss: 0.2947 - accuracy: 0.9190 - 317ms/epoch - 2ms/step
Epoch 19/30
210/210 - 0s - loss: 0.2911 - accuracy: 0.9205 - 317ms/epoch - 2ms/step
Epoch 20/30
210/210 - 0s - loss: 0.2881 - accuracy: 0.9210 - 254ms/epoch - 1ms/step
Epoch 21/30
210/210 - 0s - loss: 0.2855 - accuracy: 0.9210 - 262ms/epoch - 1ms/step
Epoch 22/30
210/210 - 0s - loss: 0.2833 - accuracy: 0.9214 - 228ms/epoch - 1ms/step
Epoch 23/30
210/210 - 0s - loss: 0.2813 - accuracy: 0.9219 - 221ms/epoch - 1ms/step
Epoch 24/30
210/210 - 0s - loss: 0.2795 - accuracy: 0.9219 - 147ms/epoch - 702us/step
Epoch 25/30
210/210 - 0s - loss: 0.2779 - accuracy: 0.9224 - 142ms/epoch - 676us/step
Epoch 26/30
210/210 - 0s - loss: 0.2766 - accuracy: 0.9238 - 344ms/epoch - 2ms/step
Epoch 27/30
210/210 - 0s - loss: 0.2753 - accuracy: 0.9248 - 394ms/epoch - 2ms/step
Epoch 28/30
210/210 - 0s - loss: 0.2741 - accuracy: 0.9262 - 332ms/epoch - 2ms/step
Epoch 29/30
210/210 - 0s - loss: 0.2731 - accuracy: 0.9276 - 385ms/epoch - 2ms/step
Epoch 30/30
210/210 - 0s - loss: 0.2721 - accuracy: 0.9276 - 385ms/epoch - 2ms/step
```

Out[30]: <keras.callbacks.History at 0x20795f399c8>

In [31]:
```
test_samples = []

for i in range (50):
    random_younger = randint (13,64)
    test_samples.append(random_younger)

    random_older = randint (65,100)
    test_samples.append(random_older)

for i in test_samples:
    print (i)

test_samples = np.array (test_samples)

scaler = MinMaxScaler(feature_range = (0,1))
scaled_test_samples = scaler.fit_transform ((test_samples).reshape(-1,1))

for i in scaled_test_samples:
    print (i)
```

64
91
49
70
54
68
22
75
55
67
14
94
60
93
44
96
30
94
31
66
62
84
56
99
45
80
21
98
18
90
35
85
61
70
31
90
25
71
16
99
24
89
46
70
62
89
50
70
51
88
36
76
37
82
43
93
25
97
30
71

20
75
52
75
53
75
39
76
46
93
35
71
24
89
51
89
56
98
46
67
25
79
53
83
28
90
51
97
27
68
48
93
54
71
44
75
46
72
37
100
[0.58139535]
[0.89534884]
[0.40697674]
[0.65116279]
[0.46511628]
[0.62790698]
[0.09302326]
[0.70930233]
[0.47674419]
[0.61627907]
[0.]
[0.93023256]
[0.53488372]
[0.91860465]
[0.34883721]
[0.95348837]
[0.18604651]
[0.93023256]
[0.19767442]
[0.60465116]

[0.55813953]
[0.81395349]
[0.48837209]
[0.98837209]
[0.36046512]
[0.76744186]
[0.08139535]
[0.97674419]
[0.04651163]
[0.88372093]
[0.24418605]
[0.8255814]
[0.54651163]
[0.65116279]
[0.19767442]
[0.88372093]
[0.12790698]
[0.6627907]
[0.02325581]
[0.98837209]
[0.11627907]
[0.87209302]
[0.37209302]
[0.65116279]
[0.55813953]
[0.87209302]
[0.41860465]
[0.65116279]
[0.43023256]
[0.86046512]
[0.25581395]
[0.72093023]
[0.26744186]
[0.79069767]
[0.3372093]
[0.91860465]
[0.12790698]
[0.96511628]
[0.18604651]
[0.6627907]
[0.06976744]
[0.70930233]
[0.44186047]
[0.70930233]
[0.45348837]
[0.70930233]
[0.29069767]
[0.72093023]
[0.37209302]
[0.91860465]
[0.24418605]
[0.6627907]
[0.11627907]
[0.87209302]
[0.43023256]
[0.87209302]
[0.48837209]
[0.97674419]
[0.37209302]
[0.61627907]

```
[0.12790698]
[0.75581395]
[0.45348837]
[0.80232558]
[0.1627907]
[0.88372093]
[0.43023256]
[0.96511628]
[0.15116279]
[0.62790698]
[0.39534884]
[0.91860465]
[0.46511628]
[0.6627907]
[0.34883721]
[0.70930233]
[0.37209302]
[0.6744186]
[0.26744186]
[1.]
```

In [32]:

```
predictions = model.predict(
    x = scaled_test_samples,
    batch_size = 10,
    verbose = 0)
for p in predictions:
    print (p)
```

```
[0.5648297  0.59331715]
[0.27852812 0.89253867]
[0.75505406 0.32088315]
[0.478713   0.69607157]
[0.6979238  0.40759602]
[0.50753176 0.66336775]
[0.85641414 0.17045613]
[0.40769047 0.769318  ]
[0.68562967 0.4258627 ]
[0.52193433 0.646383  ]
[0.85578513 0.16884887]
[0.26141167 0.90622616]
[0.62044656 0.5192536 ]
[0.26701218 0.9019699 ]
[0.8044077  0.24498883]
[0.25044078 0.91424435]
[0.8570408  0.17207551]
[0.26141167 0.90622616]
[0.857119   0.17227879]
[0.53630054 0.6290205 ]
[0.59293586 0.55659986]
[0.32114983 0.85295784]
[0.6730697 0.444334 ]
[0.23456877 0.9251095 ]
[0.79517555 0.25915614]
[0.34788144 0.8251245 ]
[0.85633564 0.17025456]
[0.23978087 0.9216362 ]
[0.8560999 0.169651 ]
[0.28440112 0.8875205 ]
[0.85003924 0.17924349]
[0.31485185 0.8592735 ]
[0.6067759 0.53798  ]
[0.478713   0.69607157]
[0.857119   0.17227879]
[0.28440112 0.8875205 ]
[0.8566494  0.17106198]
[0.46434468 0.71173406]
[0.8559426  0.16924955]
[0.23456877 0.9251095 ]
[0.856571   0.17085983]
[0.29034814 0.88229895]
[0.7856237  0.27384555]
[0.478713   0.69607157]
[0.59293586 0.55659986]
[0.29034814 0.88229895]
[0.7442331  0.33747756]
[0.478713   0.69607157]
[0.733103   0.35448223]
[0.29636797 0.87686855]
[0.84776527 0.18214285]
[0.3938433 0.7823859]
[0.8447999  0.18565916]
[0.33394137 0.8396242 ]
[0.8124939  0.23275463]
[0.26701218 0.9019699 ]
[0.8566494  0.17106198]
[0.24507168 0.91801614]
[0.8570408  0.17207551]
[0.46434468 0.71173406]
```

```
[0.8562571  0.17005318]
[0.40769047 0.769318  ]
[0.7216697  0.37186277]
[0.40769047 0.769318  ]
[0.70994043 0.38958117]
[0.40769047 0.769318  ]
[0.83731914 0.1958783 ]
[0.3938433 0.7823859]
[0.7856237  0.27384555]
[0.26701218 0.9019699 ]
[0.85003924 0.17924349]
[0.46434468 0.71173406]
[0.856571   0.17085983]
[0.29034814 0.88229895]
[0.733103   0.35448223]
[0.29034814 0.88229895]
[0.6730697 0.444334 ]
[0.23978087 0.9216362 ]
[0.7856237  0.27384555]
[0.52193433 0.646383  ]
[0.8566494  0.17106198]
[0.35527778 0.8174197 ]
[0.70994043 0.38958117]
[0.32751358 0.84640944]
[0.85688436 0.17166954]
[0.28440112 0.8875205 ]
[0.733103   0.35448223]
[0.24507168 0.91801614]
[0.8568061  0.17146681]
[0.50753176 0.66336775]
[0.76556146 0.3047294 ]
[0.26701218 0.9019699 ]
[0.6979238  0.40759602]
[0.46434468 0.71173406]
[0.8044077  0.24498883]
[0.40769047 0.769318  ]
[0.7856237  0.27384555]
[0.4500353 0.7269061]
[0.8447999  0.18565916]
[0.22943577 0.9284408 ]
```

# CNN (CONVOLUTIONAL NEURAL NETWORK)

In [33]:
```
# CNN have hidden layers called as convolutional layers
```

In [34]:
```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense, Flatten
from keras.layers.convolutional import *

model_val = Sequential ([
    Dense (units = 16, input_shape = (20,20,3), activation = 'relu'),
    Conv2D (32,kernel_size = (3,3),activation = 'relu', padding = 'same'),
    Conv2D (64,kernel_size = (5,5),activation = 'relu', padding = 'same'),
```

```
    Conv2D (128,kernel_size = (7,7),activation = 'relu', padding = 'same'),
    Flatten(),
    Dense (units = 2, activation = 'sigmoid')
])

model_val.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_50 (Dense)            (None, 20, 20, 16)        64

 conv2d_30 (Conv2D)          (None, 20, 20, 32)        4640

 conv2d_31 (Conv2D)          (None, 20, 20, 64)        51264

 conv2d_32 (Conv2D)          (None, 20, 20, 128)       401536

 flatten_10 (Flatten)        (None, 51200)             0

 dense_51 (Dense)            (None, 2)                 102402

=================================================================
Total params: 559,906
Trainable params: 559,906
Non-trainable params: 0
_________________________________________________________________
```

In [35]:

```
model_val = Sequential ([
    Dense (units = 16, input_shape = (20,20,3), activation = 'relu'),
    Conv2D (32,kernel_size = (3,3),activation = 'relu'),
    Conv2D (64,kernel_size = (5,5),activation = 'relu', padding = 'valid'),
    Conv2D (128,kernel_size = (7,7),activation = 'relu', padding = 'valid'),
    Flatten(),
    Dense (units = 2, activation = 'sigmoid')
])

model_val.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_52 (Dense)            (None, 20, 20, 16)        64

 conv2d_33 (Conv2D)          (None, 18, 18, 32)        4640

 conv2d_34 (Conv2D)          (None, 14, 14, 64)        51264

 conv2d_35 (Conv2D)          (None, 8, 8, 128)         401536

 flatten_11 (Flatten)        (None, 8192)              0

 dense_53 (Dense)            (None, 2)                 16386

=================================================================
Total params: 473,890
Trainable params: 473,890
Non-trainable params: 0
_________________________________________________________________
```

In [36]:

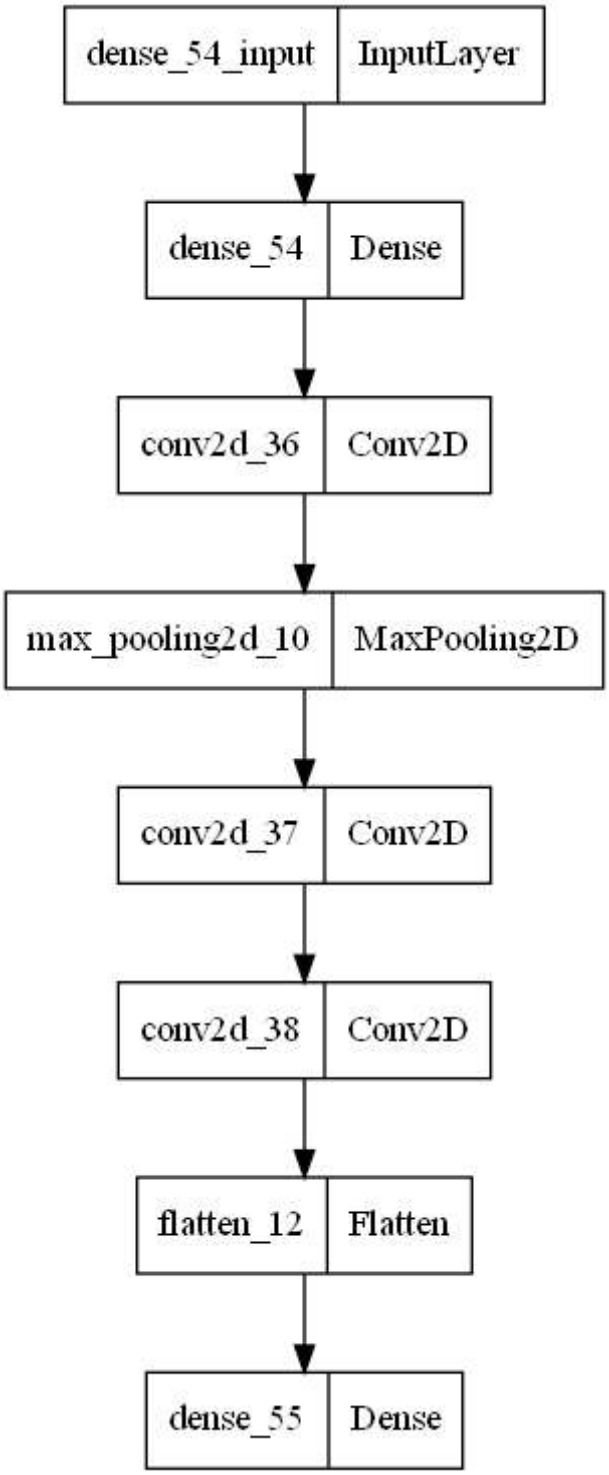
```python
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense, Flatten
from keras.layers.convolutional import *
from keras.layers.pooling import *


model_valid = Sequential ([
    Dense (units = 16, input_shape = (20,20,3), activation = 'relu'),
    Conv2D (32,kernel_size = (3,3),activation = 'relu', padding = 'same'),
    MaxPooling2D (pool_size = (2,2), strides = 2, padding = 'same'),
    Conv2D (64,kernel_size = (5,5),activation = 'relu', padding = 'same'),
    Conv2D (128,kernel_size = (7,7),activation = 'relu', padding = 'same'),
    Flatten(),
    Dense (units = 2, activation = 'sigmoid')
])
print(model_valid.layers)
print()
print(model_valid.inputs)
print()
print(model_valid.outputs)
print()
print(model_valid.get_weights)
print()
print(model_valid.summary())
```

```
[<keras.layers.core.dense.Dense object at 0x0000020791927208>, <keras.layers.convolut
ional.conv2d.Conv2D object at 0x0000020791927988>, <keras.layers.pooling.max_pooling2
d.MaxPooling2D object at 0x00000207919026C8>, <keras.layers.convolutional.conv2d.Conv
2D object at 0x00000207905A5DC8>, <keras.layers.convolutional.conv2d.Conv2D object at
0x0000020795F5FCC8>, <keras.layers.reshaping.flatten.Flatten object at 0x0000020791AE
0EC8>, <keras.layers.core.dense.Dense object at 0x0000020791B56908>]

[<KerasTensor: shape=(None, 20, 20, 3) dtype=float32 (created by layer 'dense_54_inpu
t')>]

[<KerasTensor: shape=(None, 2) dtype=float32 (created by layer 'dense_55')>]

<bound method Model.get_weights of <keras.engine.sequential.Sequential object at 0x00
00020791B56088>>

Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_54 (Dense)            (None, 20, 20, 16)        64

 conv2d_36 (Conv2D)          (None, 20, 20, 32)        4640

 max_pooling2d_10 (MaxPoolin  (None, 10, 10, 32)       0
 g2D)

 conv2d_37 (Conv2D)          (None, 10, 10, 64)        51264

 conv2d_38 (Conv2D)          (None, 10, 10, 128)       401536

 flatten_12 (Flatten)        (None, 12800)             0

 dense_55 (Dense)            (None, 2)                 25602

=================================================================
Total params: 483,106
Trainable params: 483,106
Non-trainable params: 0
_________________________________________________________________
None
```

In [37]:

```
from keras.utils import plot_model
plot_model(model_valid,to_file = 'image.png')
```

Out[37]:




In [38]:

```
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy

model = Sequential ([
    Dense (units = 16, input_shape = (8,), activation = 'relu'),
    Dense (units = 32, activation = 'relu'),
    #Dense (units = 32, activation = 'relu'),
    #Dense (units = 16, activation = 'relu'),
    Dense (units = 2, activation = 'sigmoid')
```

```
])
print(model.layers)
print(model.inputs)
print(model.outputs)
```

```
[<keras.layers.core.dense.Dense object at 0x0000020791756788>, <keras.layers.core.den
se.Dense object at 0x000002079172A548>, <keras.layers.core.dense.Dense object at 0x00
00020791A71B88>]
[<KerasTensor: shape=(None, 8) dtype=float32 (created by layer 'dense_56_input')>]
[<KerasTensor: shape=(None, 2) dtype=float32 (created by layer 'dense_58')>]
```

In [ ]:

In [ ]: