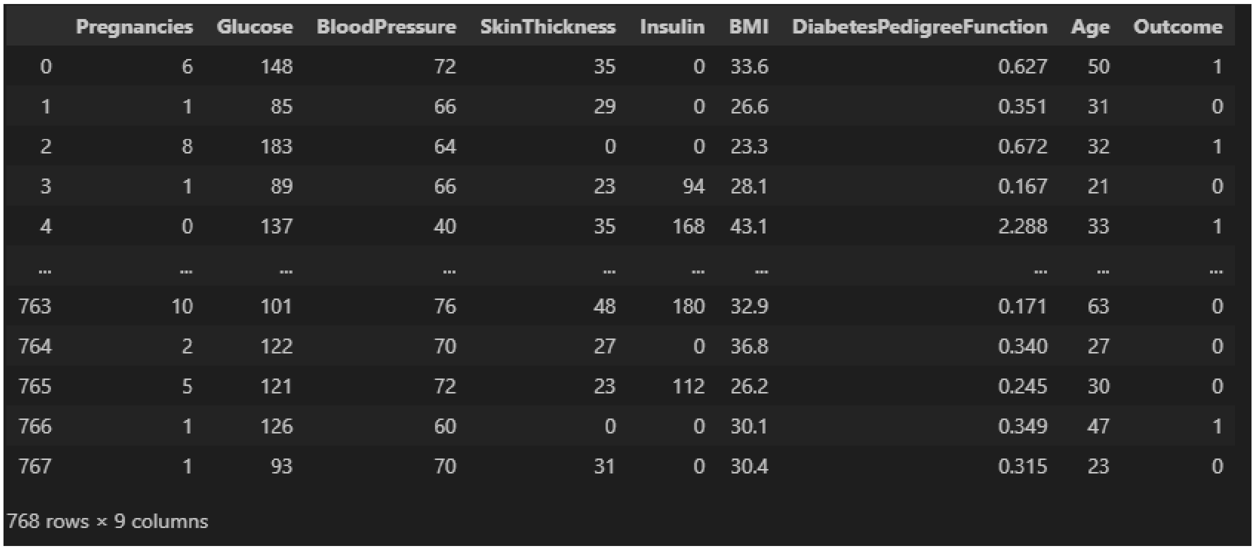
Hyperparameter Tuning a Neural Network

Library→ Keras tuner

```
pip install keras-tuner

import keras_tuner as kt
```

Dataset → *Pima Indians Diabetes Database*

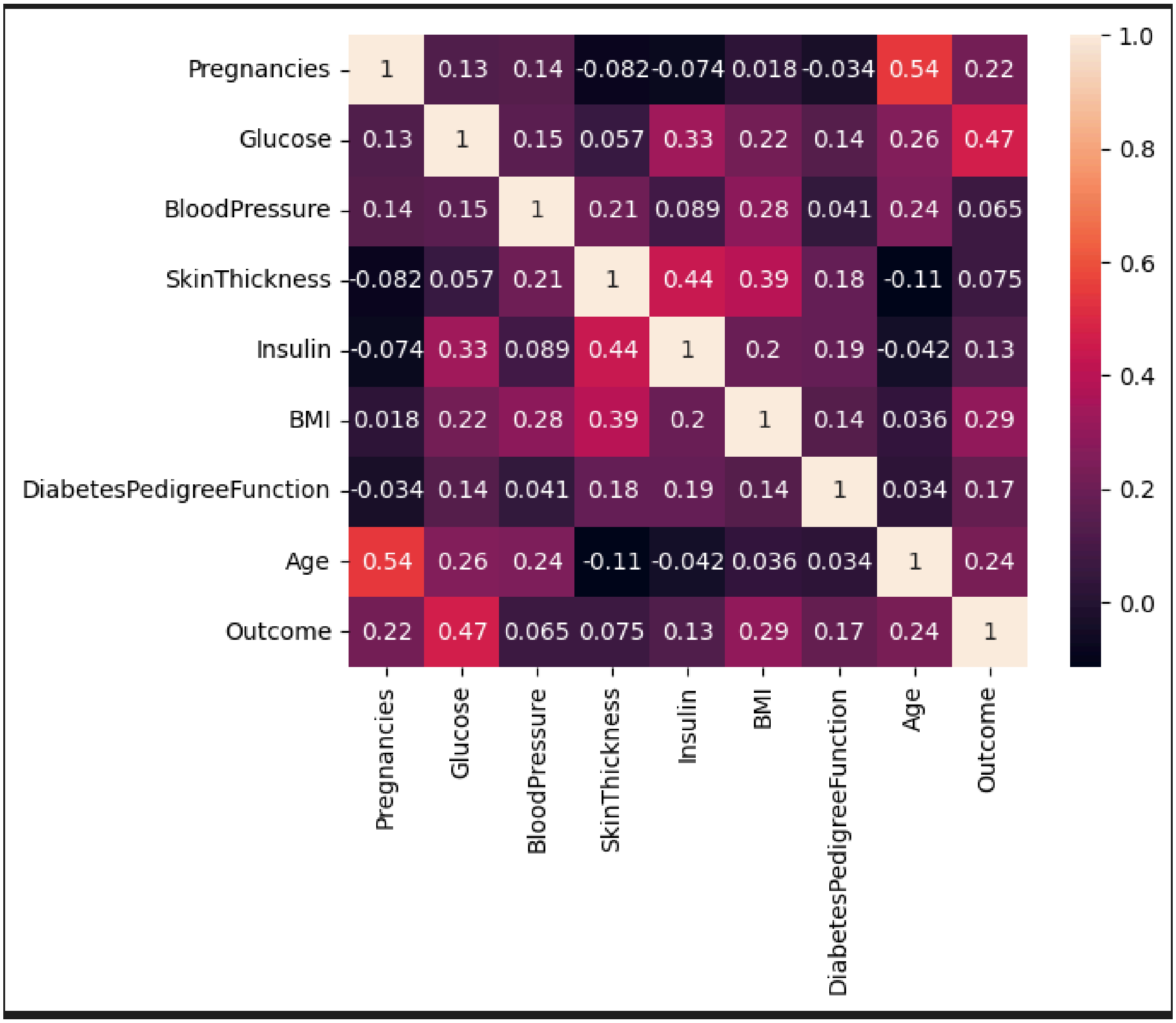
```
df = pd.read_csv(r'https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/refs/heads/master/diabetes.csv')
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

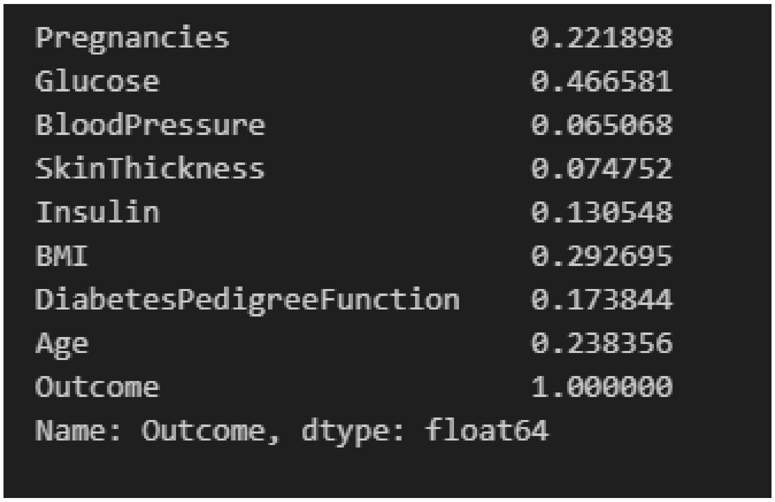
768 rows × 9 columns

```
sns.heatmap(df.corr(),annot=True)
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

```
df.corr().Outcome
```



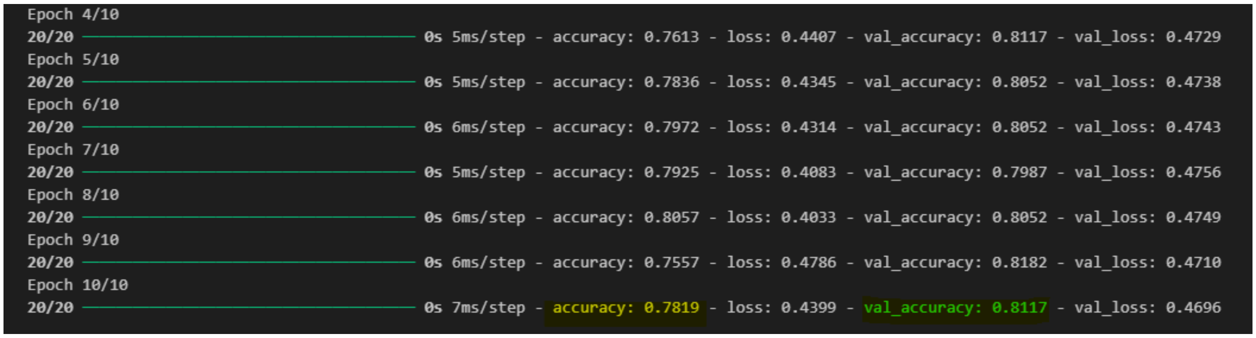
```
Pregnancies                 0.221898
Glucose                     0.466581
BloodPressure               0.065068
SkinThickness               0.074752
Insulin                     0.130548
BMI                         0.292695
DiabetesPedigreeFunction    0.173844
Age                         0.238356
Outcome                     1.000000
Name: Outcome, dtype: float64
```

```
X= df.drop(columns=['Outcome'])
y= df['Outcome']

# SCALE

from sklearn.preprocessing import StandardScaler as sc

scaler = sc()
X= scaler.fit_transform(X)

# train_test_split

from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state
=1)
```

Build a model

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

```
model = Sequential()
model.add(Dense(32, activation='relu',input_dim=8))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='BinaryCrossentropy', metrics=['accuracy'])

model.fit(X_train,y_train, batch_size=32, epochs=10,validation_data=(X_test, y_test))
```



```
Epoch 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7613 - loss: 0.4407 - val_accuracy: 0.8117 - val_loss: 0.4729
Epoch 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7836 - loss: 0.4345 - val_accuracy: 0.8052 - val_loss: 0.4738
Epoch 6/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7972 - loss: 0.4314 - val_accuracy: 0.8052 - val_loss: 0.4743
Epoch 7/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7925 - loss: 0.4083 - val_accuracy: 0.7987 - val_loss: 0.4756
Epoch 8/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8057 - loss: 0.4033 - val_accuracy: 0.8052 - val_loss: 0.4749
Epoch 9/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7557 - loss: 0.4786 - val_accuracy: 0.8182 - val_loss: 0.4710
Epoch 10/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7819 - loss: 0.4399 - val_accuracy: 0.8117 - val_loss: 0.4696
```

Tune the Hyperparameters

```
pip install keras-tuner

import keras_tuner as kt
```

Steps:

1. Make a function
2. Pass the function into a tuner object

Find best optimizer:

```
def build_model(hp):
  model = Sequential()
  model.add(Dense(32, activation='relu',input_dim=8))
  model.add(Dense(1, activation='sigmoid'))

  optimizer = hp.Choice('optimizer', ['adam','sgd','rmsprop','adadelta'])

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

Make Tuner Object:

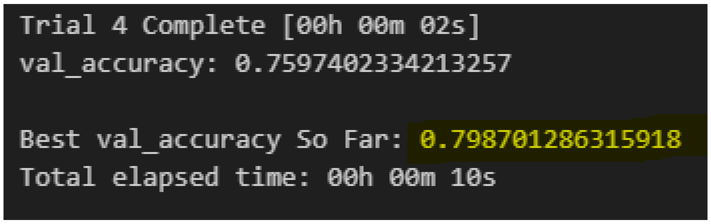
```
tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=5)
```

Default Objective: `'val_loss'`

- Option 2 : `'val_accuracy'` : The validation accuracy.

`max_trials` : Default = 10

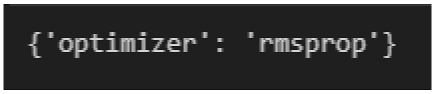
```
tuner.search(X_train,y_train, epochs=5, validation_data=(X_test,y_test))
```



```
Trial 4 Complete [00h 00m 02s]
val_accuracy: 0.7597402334213257

Best val_accuracy So Far: 0.798701286315918
Total elapsed time: 00h 00m 10s
```

Get best HP:

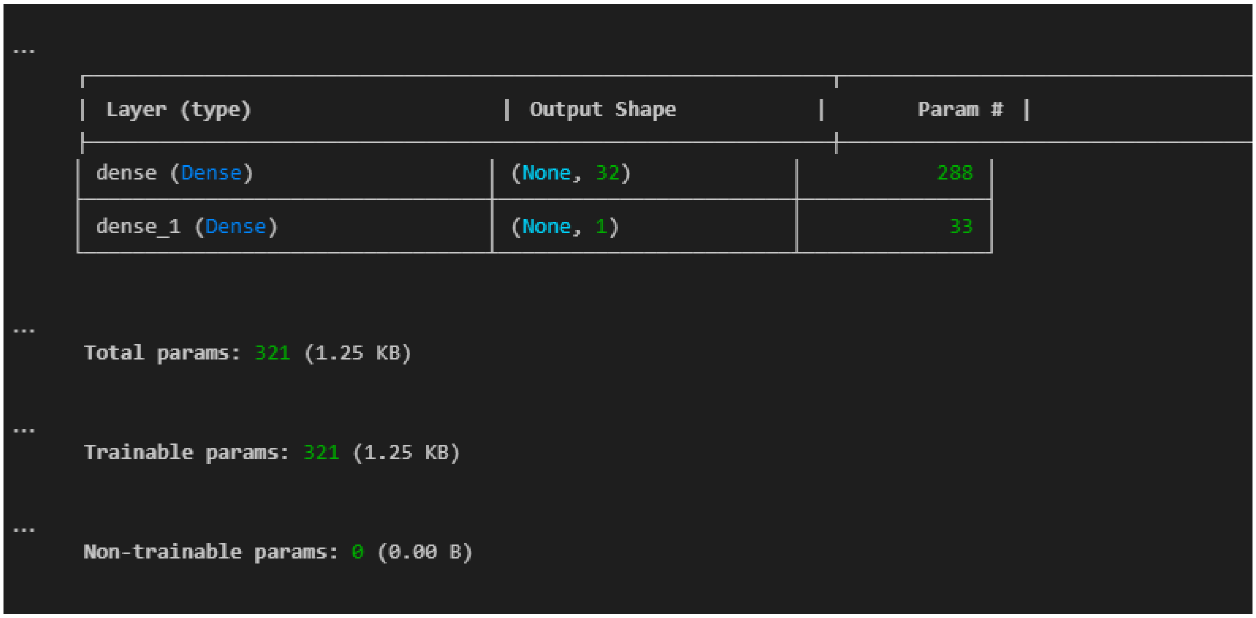
```
tuner.get_best_hyperparameters()[0].values
```



```
{'optimizer': 'rmsprop'}
```

Get best model

```
model = tuner.get_best_models(num_models=1)[0]
model.summary()
```

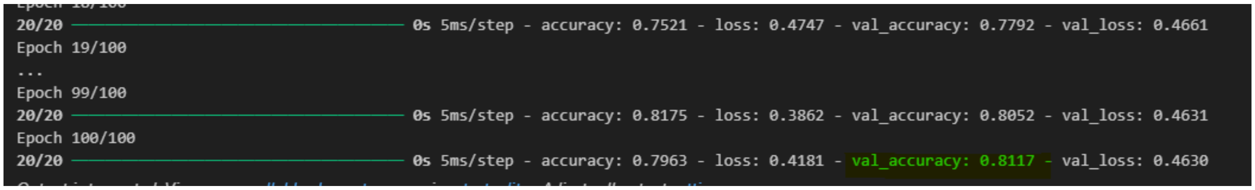


```
...
      ┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
      ┃ Layer (type)                     ┃ Output Shape            ┃       Param #  ┃
      ┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
      │ dense (Dense)                    │ (None, 32)              │           288  │
      ├──────────────────────────────────┼─────────────────────────┼────────────────┤
      │ dense_1 (Dense)                  │ (None, 1)               │            33  │
      └──────────────────────────────────┴─────────────────────────┴────────────────┘
...
      Total params: 321 (1.25 KB)
...
      Trainable params: 321 (1.25 KB)
...
      Non-trainable params: 0 (0.00 B)
```

Fit :

```
model.fit(X_train,y_train,batch_size=32,epochs=100,initial_epoch=6,validation_data=(X_test,y_test))
```

`initial_epoch=6` because we have already run 5 epochs



```
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7521 - loss: 0.4747 - val_accuracy: 0.7792 - val_loss: 0.4661
Epoch 19/100
...
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8175 - loss: 0.3862 - val_accuracy: 0.8052 - val_loss: 0.4631
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7963 - loss: 0.4181 - val_accuracy: 0.8117 - val_loss: 0.4630
```

Find out number of neurons:

```
def build_model(hp):
    model= Sequential()
    units = hp.Int('units', 8,128,8)
    model.add(Dense(units=units,activation='relu', input_dim=8))
    model.add(Dense(1,activation='sigmoid'))

    model.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['accuracy'], )

    return model
```

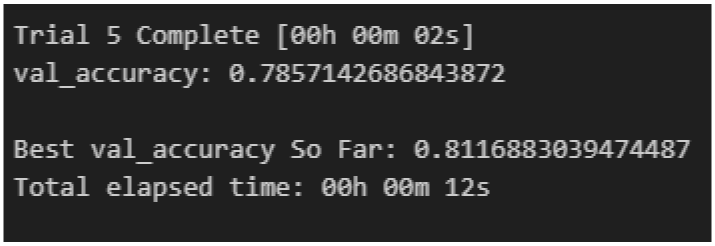
`8,128,8` → min_value = 8,max_value = 128, step=8

Create Tuner Object:

```
tuner = kt.RandomSearch(build_model, objective='val_accuracy', max_trials=5, directory='mydir', project_name='kuch_bhi')
```

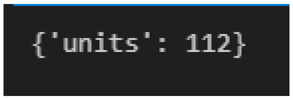
Search best parameters:

```
tuner.search(X_train,y_train, validation_data=(X_test,y_test))
```



```
Trial 5 Complete [00h 00m 02s]
val_accuracy: 0.7857142686843872

Best val_accuracy So Far: 0.8116883039474487
Total elapsed time: 00h 00m 12s
```

```
tuner.get_best_hyperparameters()[0].values
```



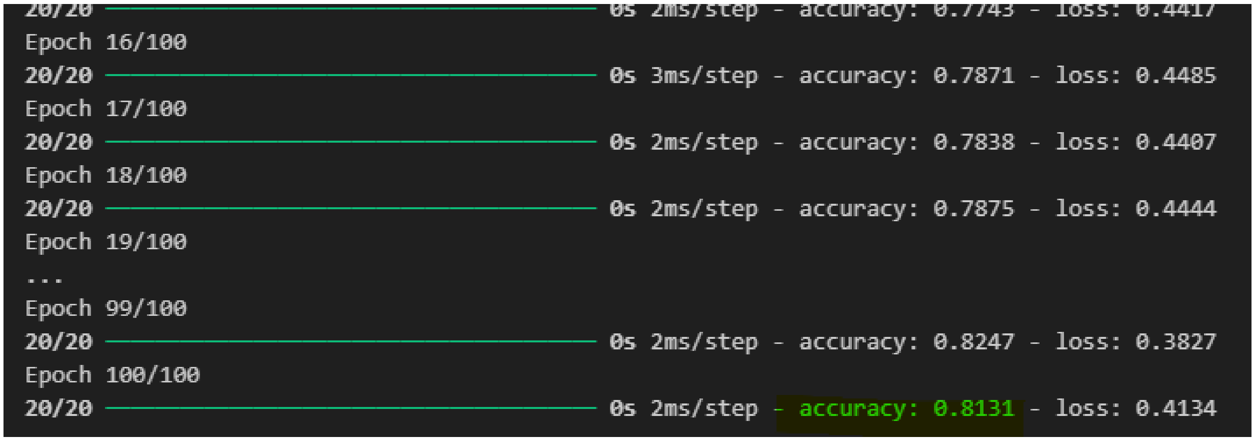
```
{'units': 112}
```

Best model:

```
model2= tuner.get_best_models(num_models=1)[0]
```

Fit again:

```
model2.fit(X_train,y_train,batch_size=32,epochs=100,initial_epoch=6, validation_data=(X_test,y_test))
```



```
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7743 - loss: 0.4417
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7871 - loss: 0.4485
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7838 - loss: 0.4407
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7875 - loss: 0.4444
Epoch 19/100
...
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8247 - loss: 0.3827
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8131 - loss: 0.4134
```

Select No. of layers:

```
def build_model(hp):
    model = Sequential()
    model.add(Dense(112, activation='relu', input_dim=8))

    for i in range(hp.Int('num_layers', min_value=1, max_value=10)):
        model.add(Dense(72, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])

    return model
```

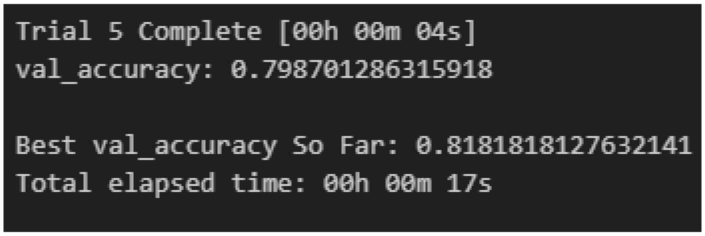
`for i in range(hp.Int('num_layers', min_value=1, max_value=10))`

- There will be 10 models with 1 to 10 layers

`hp.Int('num_layers', min_value=1, max_value=10):` This is where the magic of hyperparameter tuning comes in!

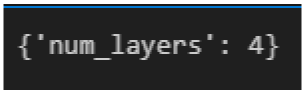
- ***`hp.Int` means pick a whole number (integer) for the hyperparameter named `'num_layers'`***
- `min_value=1, max_value=10` means the number of layers can be between 1 and 10
- **`for i in range(...)`**: This loop will run as many times as the number of layers chosen by hp.Int. For example, if hp.Int picks 3, the loop runs 3 times to add 3 layers.

```
tuner=kt.RandomSearch(build_model, objective='val_accuracy',max_trials=5,
directory= 'mydir', project_name= 'num_layers')

tuner.search(X_train,y_train, epochs=5, validation_data=(X_test,y_test))
```



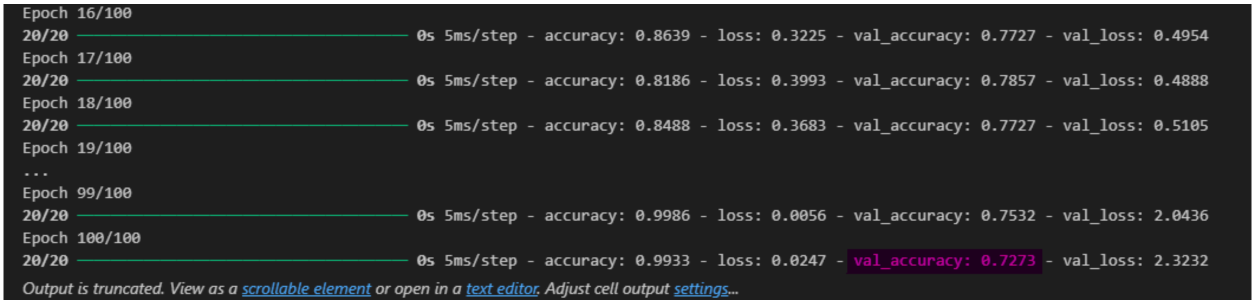
```
Trial 5 Complete [00h 00m 04s]
val_accuracy: 0.798701286315918

Best val_accuracy So Far: 0.8181818127632141
Total elapsed time: 00h 00m 17s
```

```
tuner.get_best_hyperparameters()[0].values
```



```
{'num_layers': 4}
```

```
model3 = tuner.get_best_models(num_models=1)[0]

model3.fit(X_train,y_train,epochs=100,initial_epoch=6,validation_data=(X_test,
y_test))
```



```
Epoch 16/100
20/20 ──────────────────────────────── 0s 5ms/step - accuracy: 0.8639 - loss: 0.3225 - val_accuracy: 0.7727 - val_loss: 0.4954
Epoch 17/100
20/20 ──────────────────────────────── 0s 5ms/step - accuracy: 0.8186 - loss: 0.3993 - val_accuracy: 0.7857 - val_loss: 0.4888
Epoch 18/100
20/20 ──────────────────────────────── 0s 5ms/step - accuracy: 0.8488 - loss: 0.3683 - val_accuracy: 0.7727 - val_loss: 0.5105
Epoch 19/100
...
Epoch 99/100
20/20 ──────────────────────────────── 0s 5ms/step - accuracy: 0.9986 - loss: 0.0056 - val_accuracy: 0.7532 - val_loss: 2.0436
Epoch 100/100
20/20 ──────────────────────────────── 0s 5ms/step - accuracy: 0.9933 - loss: 0.0247 - val_accuracy: 0.7273 - val_loss: 2.3232
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

👆 Validation Accuracy is decreasing

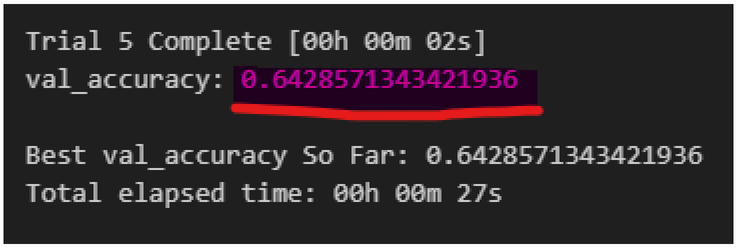
Now find of both No. of layers & Nodes

```
def build_model(hp):
  model = Sequential()
  counter = 0


  for i in range(hp.Int('num_layers',min_value=1,max_value=10)):

    if counter == 0:
      model.add(Dense(
        hp.Int('units'+str(i), min_value=8, max_value=128, step=8),
        activation= hp.Choice('activation' + str(i), values=['relu','tanh','sigmoid']),
                        input_dim=8
        ))
    else:
      model.add(Dense(hp.Int('units' + str(i), min_value=8, max_value=128,step=
8),activation= hp.Choice('activation' + str(i), values=['relu','tanh','sigmoid'])))
    counter += 1

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

    model.compile(optimizer=hp.Choice('optimizer',values=['rmsprop','adam','s
gd','nadam','adadelta']),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

  return model
```

Tuner Object:

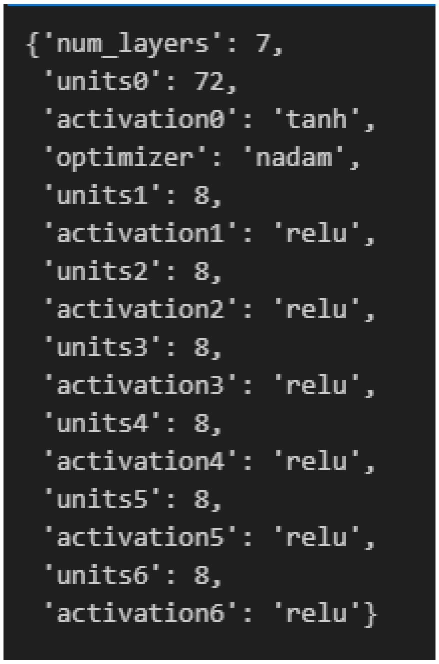
```
tuner = kt.RandomSearch(build_model,
                        objective='val_accuracy',
                        max_trials=5,
                        directory="mydir",
                        project_name='final')
```

```
tuner.search(X_train,y_train,epochs=5,validation_data=(X_test,y_test))
```



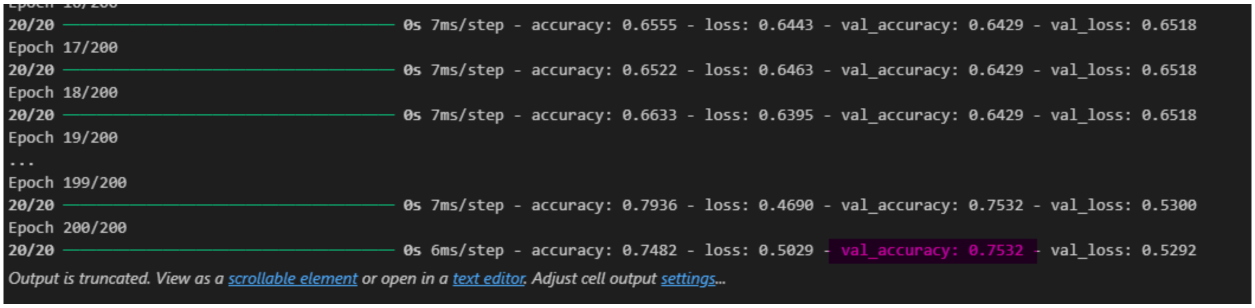
```
Trial 5 Complete [00h 00m 02s]
val_accuracy: 0.6428571343421936

Best val_accuracy So Far: 0.6428571343421936
Total elapsed time: 00h 00m 27s
```

```
tuner.get_best_hyperparameters()[0].values
```



```
{'num_layers': 7,
 'units0': 72,
 'activation0': 'tanh',
 'optimizer': 'nadam',
 'units1': 8,
 'activation1': 'relu',
 'units2': 8,
 'activation2': 'relu',
 'units3': 8,
 'activation3': 'relu',
 'units4': 8,
 'activation4': 'relu',
 'units5': 8,
 'activation5': 'relu',
 'units6': 8,
 'activation6': 'relu'}
```

```
model4= tuner.get_best_models(num_models=1)[0]
model4.fit(X_train,y_train,epochs=200,initial_epoch=6,validation_data=(X_test,
y_test))
```



```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6555 - loss: 0.6443 - val_accuracy: 0.6429 - val_loss: 0.6518
Epoch 17/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6522 - loss: 0.6463 - val_accuracy: 0.6429 - val_loss: 0.6518
Epoch 18/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6633 - loss: 0.6395 - val_accuracy: 0.6429 - val_loss: 0.6518
Epoch 19/200
...
Epoch 199/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7936 - loss: 0.4690 - val_accuracy: 0.7532 - val_loss: 0.5300
Epoch 200/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7482 - loss: 0.5029 - val_accuracy: 0.7532 - val_loss: 0.5292
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Add dropout layer:

- All code is same as above. just added 👇

```
model.add(Dropout(hp.Choice('dropout' + str(i), values=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9])))
```

```
def build_model(hp):

  model = Sequential()

  counter = 0

  for i in range(hp.Int('num_layers',min_value=1,max_value=10)):

    if counter == 0:

      model.add(Dense(hp.Int('units' + str(i), min_value=8, max_value=128,step=8),activation= hp.Choice('activation' + str(i), values=['relu','tanh','sigmoid']),input_dim=8))

      model.add(Dropout(hp.Choice('dropout' + str(i), values=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9])))
    else:

      model.add(Dense(hp.Int('units' + str(i), min_value=8, max_value=128,step=8),activation= hp.Choice('activation' + str(i), values=['relu','tanh','sigmoid'])))
      model.add(Dropout(hp.Choice('dropout' + str(i), values=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9])))

    counter+=1

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

  model.compile(optimizer=hp.Choice('optimizer',values=['rmsprop','adam','sgd','nadam','adadelta']),
                loss='binary_crossentropy',
                metrics=['accuracy'])

  return model
```

`counter = 0`

- This sets up a variable called counter and starts it at 0. It’s like a little counter to keep track of which layer we’re adding (first, second, third, etc.).
- We’ll use this to decide if we’re adding the first layer (which needs input_dim) or a later layer.

`'units' + str(i)` :

- This makes a unique name for each layer’s hyperparameter (like 'units0', 'units1', etc.), so the tuner can track them separately

Why `str(i)` ?

- It gives each layer a unique name (like 'units0', 'units1') so the tuner can pick different settings for each layer.

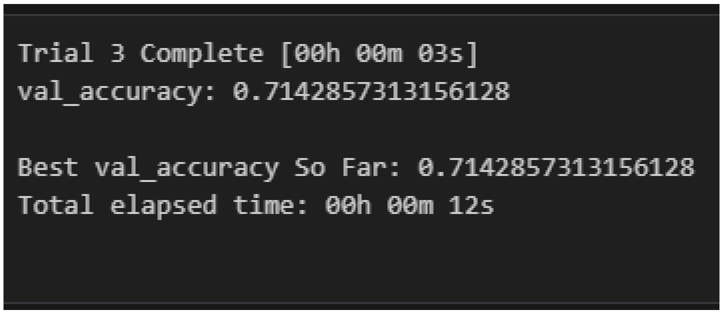
Can We Simplify Without `str(i)` ?

- Yes, but then all layers will have the same settings (same neurons, activation, dropout), which is less flexible.
- Simple answer → **NO**.

Create a tuner object:

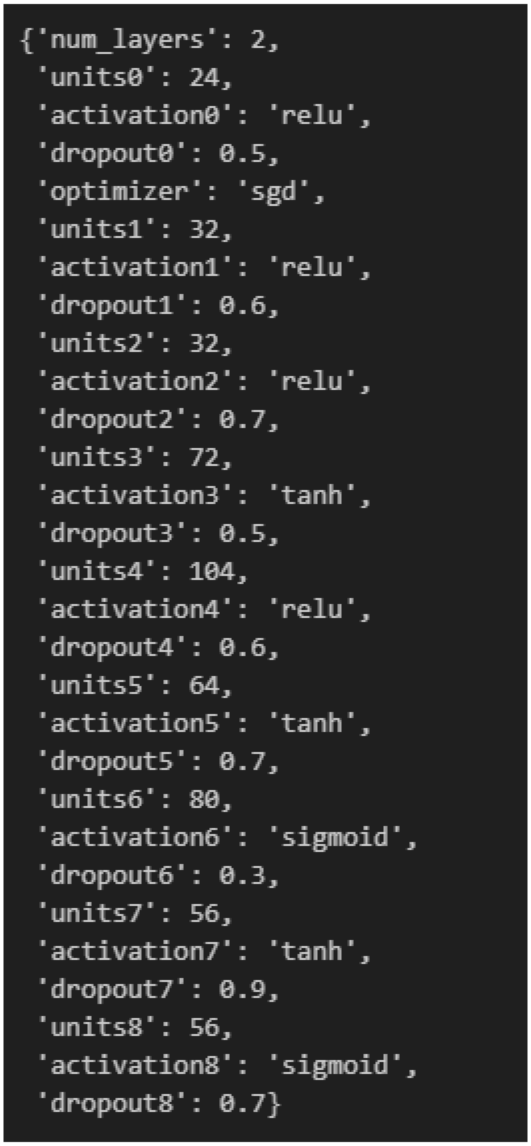
```
tuner = kt.RandomSearch(build_model,
            objective='val_accuracy',
            max_trials=3,
            directory='mydir',
            project_name='kuch bhi')
```

```
tuner.search(X_train,y_train,epochs=5,validation_data=(X_test,y_test))
```



```
Trial 3 Complete [00h 00m 03s]
val_accuracy: 0.7142857313156128

Best val_accuracy So Far: 0.7142857313156128
Total elapsed time: 00h 00m 12s
```

```
tuner.get_best_hyperparameters()[0].values
```

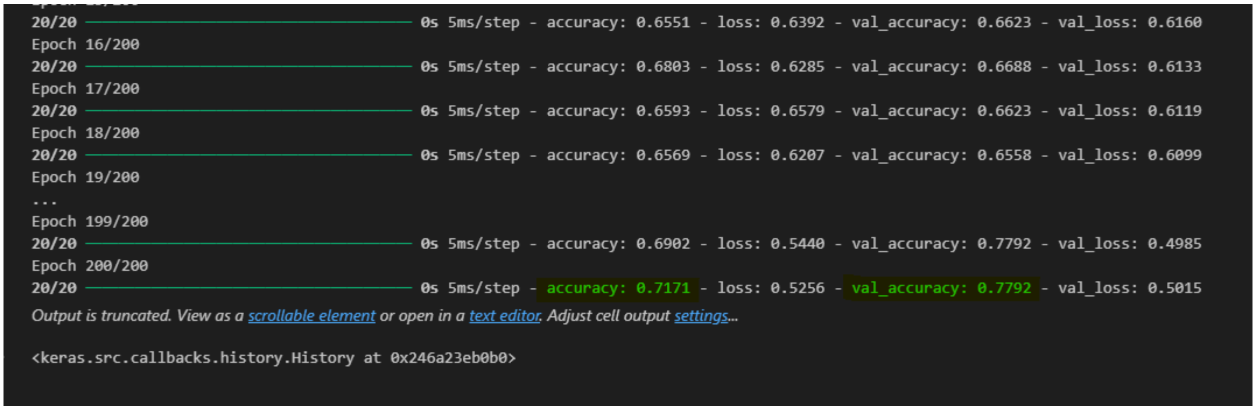


```
{'num_layers': 2,
 'units0': 24,
 'activation0': 'relu',
 'dropout0': 0.5,
 'optimizer': 'sgd',
 'units1': 32,
 'activation1': 'relu',
 'dropout1': 0.6,
 'units2': 32,
 'activation2': 'relu',
 'dropout2': 0.7,
 'units3': 72,
 'activation3': 'tanh',
 'dropout3': 0.5,
 'units4': 104,
 'activation4': 'relu',
 'dropout4': 0.6,
 'units5': 64,
 'activation5': 'tanh',
 'dropout5': 0.7,
 'units6': 80,
 'activation6': 'sigmoid',
 'dropout6': 0.3,
 'units7': 56,
 'activation7': 'tanh',
 'dropout7': 0.9,
 'units8': 56,
 'activation8': 'sigmoid',
 'dropout8': 0.7}
```

```
model = tuner.get_best_models(num_models=1)[0]
```

Fit

```
model.fit(X_train,y_train,epochs=200,initial_epoch=6,validation_data=(X_test,y_test))
```



```
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6551 - loss: 0.6392 - val_accuracy: 0.6623 - val_loss: 0.6160
Epoch 16/200
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6803 - loss: 0.6285 - val_accuracy: 0.6688 - val_loss: 0.6133
Epoch 17/200
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6593 - loss: 0.6579 - val_accuracy: 0.6623 - val_loss: 0.6119
Epoch 18/200
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6569 - loss: 0.6207 - val_accuracy: 0.6558 - val_loss: 0.6099
Epoch 19/200
...
Epoch 199/200
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6902 - loss: 0.5440 - val_accuracy: 0.7792 - val_loss: 0.4985
Epoch 200/200
20/20 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7171 - loss: 0.5256 - val_accuracy: 0.7792 - val_loss: 0.5015
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
<keras.src.callbacks.history.History at 0x246a23eb0b0>
```