```
import tensorflow as tf
import numpy as np
```

```
from sklearn.datasets import load_breast_cancer
```

```
data = load_breast_cancer()
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_si
n, d = X_train.shape
print(f"Shape of the training data: {X_train.shape}")
```

```
Shape of the training data: (398, 30)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
import pandas as pd
cols = ['radius_mean', 'texture_mean', 'perimeter_mean',
           'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
           'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
           'radius_se', 'texture_se', 'perimeter_se', 'area_se',
           'smoothness_se', 'compactness_se', 'concavity_se',
           'concave points_se', 'symmetry_se', 'fractal_dimension_se',
           'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
           'smoothness_worst', 'compactness_worst', 'concavity_worst',
           'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']

df_train = pd.DataFrame(data=X_train, index=range(X_train.shape[0]), columns=cols)
df_train.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compa
0	0.010095	0.303994	0.007629	-0.113641	-0.511166	
1	0.080856	0.229315	0.092735	-0.043758	0.154891	
2	0.443151	-1.924333	0.432749	0.256770	0.549855	
3	-0.558822	-1.440123	-0.537784	-0.581256	0.712653	
4	-0.640904	-1.112499	-0.577473	-0.647401	1.321378	

```
def initialize_population(n=8):
  layers = [1, 2, 3]
  units = [8, 16, 32, 64]
```

```
  return [{
    'layers': layers[i % len(layers)],
    'units': [np.random.choice(units)  for i in range(layers[i % len(layers)])]
   } for i in range(n)]


population = initialize_population(16)
print(f'Initial Population: {len(population)} samples')
population
```

```
Initial Population: 16 samples
[{'layers': 1, 'units': [8]},
 {'layers': 2, 'units': [16, 8]},
 {'layers': 3, 'units': [8, 8, 64]},
 {'layers': 1, 'units': [32]},
 {'layers': 2, 'units': [8, 16]},
 {'layers': 3, 'units': [8, 16, 32]},
 {'layers': 1, 'units': [32]},
 {'layers': 2, 'units': [8, 16]},
 {'layers': 3, 'units': [8, 16, 8]},
 {'layers': 1, 'units': [8]},
 {'layers': 2, 'units': [8, 64]},
 {'layers': 3, 'units': [32, 64, 32]},
 {'layers': 1, 'units': [64]},
 {'layers': 2, 'units': [32, 8]},
 {'layers': 3, 'units': [16, 16, 64]},
 {'layers': 1, 'units': [64]}]
```

```
def fitness_function(training_accuracy, validation_accuracy):
    return (training_accuracy + 10 * validation_accuracy) / 11
```

```
def inject_fitness(population, model_callbacks):
  for i, _ in enumerate(population):
    train_accuracy, val_accuracy = model_callbacks[i].history['accuracy'][-1], mode
    population[i]['fitness'] = fitness_function(train_accuracy, val_accuracy)
```

```
def crossover(gene1, gene2):
  children = []
  for i in range(2):
    l = np.random.choice([gene1['layers'], gene2['layers']])
    u = [np.random.choice(gene1['units'] + gene2['units']) for _ in range(l)]
    children.append({'layers': l, 'units': u})
  return children
```

```
def get_next_generation(population, model_callbacks, n, mutation_rate=0.01):
  inject_fitness(population, model_callbacks)
  population.sort(key=lambda val: val['fitness'])
  fittest_genes = population[-n:]
  new_population = fittest_genes
  for i in range(n):
    for j in range(i + 1, n):
      gene1, gene2 = fittest_genes[i], fittest_genes[j]
      new_population += crossover(gene1, gene2)
      if np.random.rand() <= mutation_rate:
```

```
        new_population[-1] = mutation()
  return new_population
```

```
def mutation():
  return initialize_population(1)[0]
```

```
def build_model(gene):
  ip = tf.keras.layers.Input(shape=(d, ))
  for i, val in enumerate(gene['units']):
      x = tf.keras.layers.Dense(val, activation='relu')(ip if not i else x)
  x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
  return tf.keras.models.Model(ip, x)
```

```
def compile_models(models):
  for model in models:
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'
```

```
def train_models(models):
  model_callbacks = []
  for model in models:
    r = model.fit(X_train, y_train, validation_data=(X_test, y_test), verbose=0, ep
    model_callbacks.append(r)
  return model_callbacks
```

```
def get_max_fitness(model_callbacks):
  return max([fitness_function(r.history['accuracy'][-1], r.history['val_accuracy']
```

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

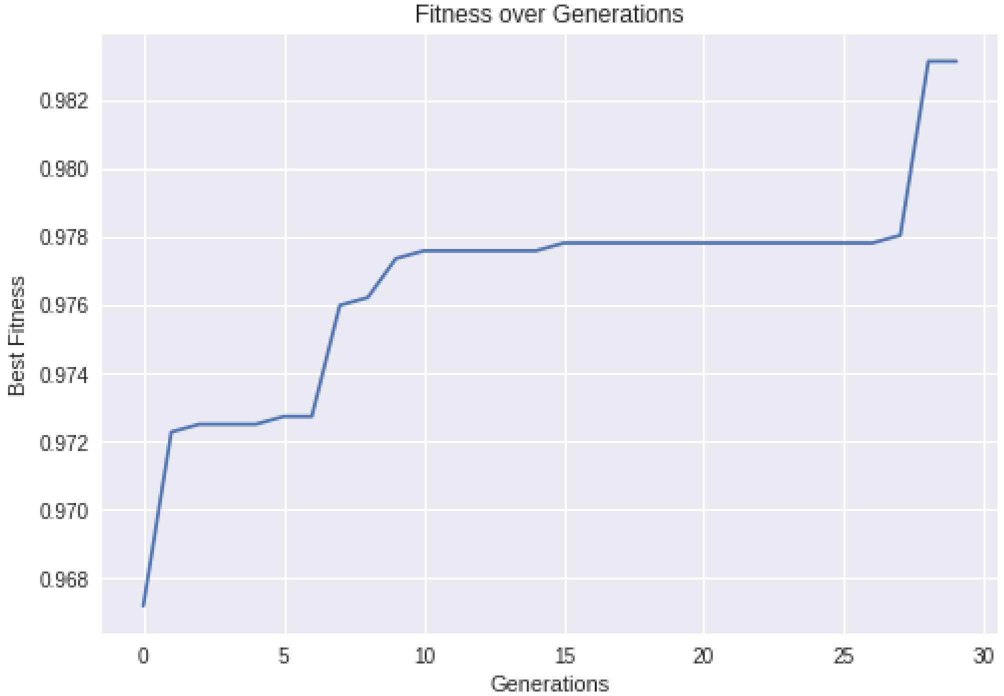
```
def main():
  population = initialize_population(16)
  fitness_for_generations = []
  num_iterations = 30
  for i in range(num_iterations):
    models = [build_model(gene) for gene in population]
    compile_models(models)
    model_callbacks = train_models(models)
    fitness_for_generations.append(get_max_fitness(model_callbacks))
    print(f"{i}", end=' ')
    population = get_next_generation(population, model_callbacks, 3, mutation_rate=
  fitness_for_generations.sort()
  print()
  return fitness_for_generations
```

```
def plot(fitness_for_generations):
  for i, val in enumerate(fitness_for_generations):
    print(f"Generation: {i}, Best Fitness: {val}")
  plt.xlabel('Generations')
  plt.ylabel('Best Fitness')
```

```
    plt.title('Fitness over Generations')
    plt.plot(fitness_for_generations)
```

```
f = main()
plot(f)
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
Generation: 0, Best Fitness: 0.9671884287487377
Generation: 1, Best Fitness: 0.9722763137383894
Generation: 2, Best Fitness: 0.9725047295743768
Generation: 3, Best Fitness: 0.9725047295743768
Generation: 4, Best Fitness: 0.9725047295743768
Generation: 5, Best Fitness: 0.9727331454103644
Generation: 6, Best Fitness: 0.9727331454103644
Generation: 7, Best Fitness: 0.9759937633167614
Generation: 8, Best Fitness: 0.9762221791527488
Generation: 9, Best Fitness: 0.9773642529140819
Generation: 10, Best Fitness: 0.9775926687500693
Generation: 11, Best Fitness: 0.9775926687500693
Generation: 12, Best Fitness: 0.9775926687500693
Generation: 13, Best Fitness: 0.9775926687500693
Generation: 14, Best Fitness: 0.9775926687500693
Generation: 15, Best Fitness: 0.9778210845860568
Generation: 16, Best Fitness: 0.9778210845860568
Generation: 17, Best Fitness: 0.9778210845860568
Generation: 18, Best Fitness: 0.9778210845860568
Generation: 19, Best Fitness: 0.9778210845860568
Generation: 20, Best Fitness: 0.9778210845860568
Generation: 21, Best Fitness: 0.9778210845860568
Generation: 22, Best Fitness: 0.9778210845860568
Generation: 23, Best Fitness: 0.9778210845860568
Generation: 24, Best Fitness: 0.9778210845860568
Generation: 25, Best Fitness: 0.9778210845860568
Generation: 26, Best Fitness: 0.9778210845860568
Generation: 27, Best Fitness: 0.9780495004220442
Generation: 28, Best Fitness: 0.983137385411696
Generation: 29, Best Fitness: 0.983137385411696
```



```
m = build_model({'layers': 1, 'units': [64]})
m.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
r = m.fit(X_train, y_train, validation_data=(X_test, y_test), verbose=1, epochs=100
```

```
Epoch 1/100
13/13 [==============================] - 0s 10ms/step - loss: 0.6682 - accurac
Epoch 2/100
13/13 [==============================] - 0s 3ms/step - loss: 0.3886 - accuracy
Epoch 3/100
13/13 [==============================] - 0s 3ms/step - loss: 0.2624 - accuracy
Epoch 4/100
13/13 [==============================] - 0s 3ms/step - loss: 0.1983 - accuracy
Epoch 5/100
13/13 [==============================] - 0s 4ms/step - loss: 0.1629 - accuracy
Epoch 6/100
13/13 [==============================] - 0s 3ms/step - loss: 0.1392 - accuracy
Epoch 7/100
13/13 [==============================] - 0s 3ms/step - loss: 0.1233 - accuracy
Epoch 8/100
13/13 [==============================] - 0s 3ms/step - loss: 0.1109 - accuracy
Epoch 9/100
13/13 [==============================] - 0s 4ms/step - loss: 0.1013 - accuracy
Epoch 10/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0933 - accuracy
Epoch 11/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0866 - accuracy
Epoch 12/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0813 - accuracy
Epoch 13/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0764 - accuracy
Epoch 14/100
13/13 [==============================] - 0s 4ms/step - loss: 0.0724 - accuracy
Epoch 15/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0688 - accuracy
Epoch 16/100
13/13 [==============================] - 0s 4ms/step - loss: 0.0656 - accuracy
Epoch 17/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0630 - accuracy
Epoch 18/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0605 - accuracy
Epoch 19/100
13/13 [==============================] - 0s 4ms/step - loss: 0.0582 - accuracy
Epoch 20/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0563 - accuracy
Epoch 21/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0545 - accuracy
Epoch 22/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0527 - accuracy
Epoch 23/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0511 - accuracy
Epoch 24/100
13/13 [==============================] - 0s 4ms/step - loss: 0.0497 - accuracy
Epoch 25/100
13/13 [==============================] - 0s 4ms/step - loss: 0.0485 - accuracy
Epoch 26/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0472 - accuracy
Epoch 27/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0461 - accuracy
Epoch 28/100
13/13 [==============================] - 0s 3ms/step - loss: 0.0450 - accuracy
Epoch 29/100
```

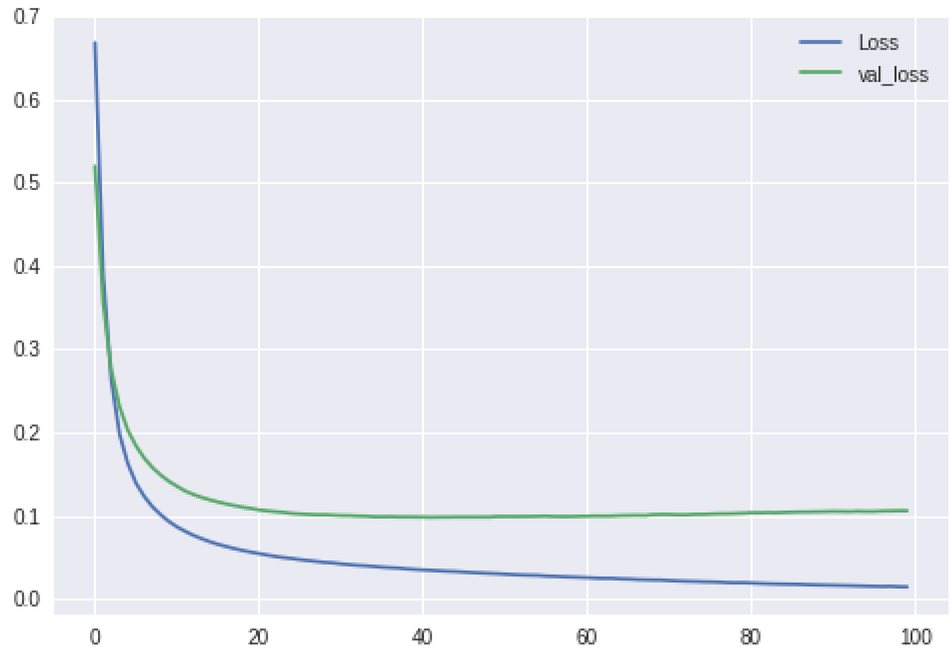
```
13/13 [==============================] - 0s 3ms/step - loss: 0.0439 - accuracy
Epoch 30/100
```

```
print(f"Train Accuracy: {m.evaluate(X_train, y_train)}")
print(f"Test Accuracy: {m.evaluate(X_test, y_test)}")
```

```
13/13 [==============================] - 0s 2ms/step - loss: 0.0139 - accuracy
Train Accuracy: [0.013913062401115894, 0.9949748516082764]
6/6 [==============================] - 0s 2ms/step - loss: 0.1060 - accuracy:
Test Accuracy: [0.10595281422138214, 0.9766082167625427]
```

```
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f82ac276240>
```




```
plt.plot(r.history['accuracy'], label='accuracy')
plt.plot(r.history['val_accuracy'], label='val_accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8284c89f60>
```