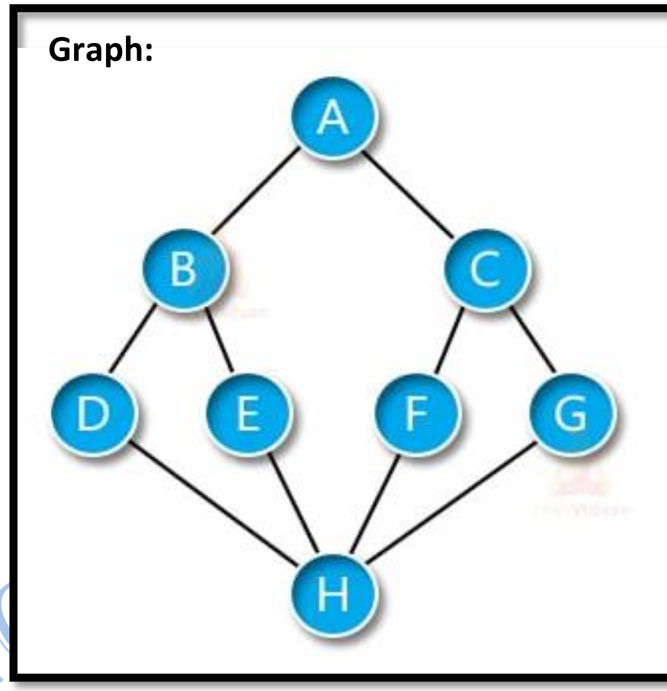
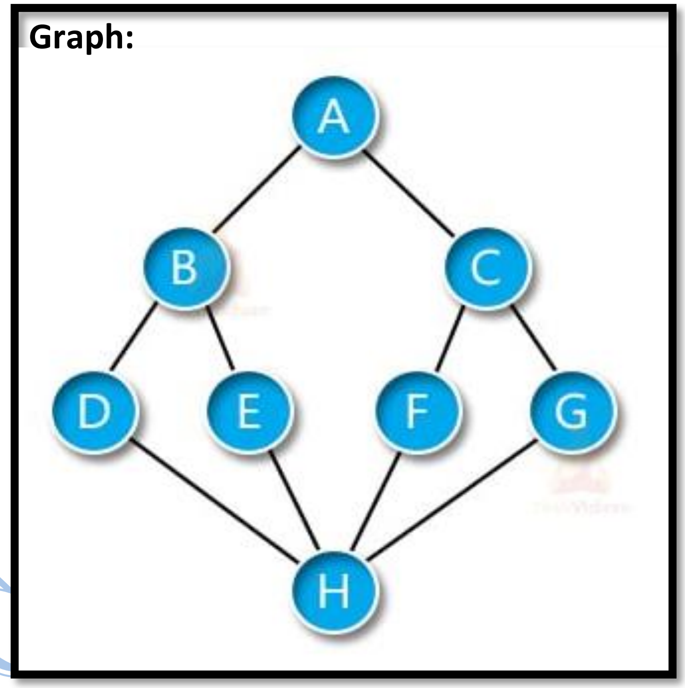
1. **Write a program to implement breadth first search using python.**




```
#Adjacency list
graph = {
        'A': ['B', 'C'],
        'B': ['D','E'],
        'C': ['F','G'],
        'D': ['H'],
        'E': ['H'],
        'F': ['H'],
        'G': ['H'],
        'H': []
        }
visited = []  # List to keep track of visited nodes.
queue = []   # Initialize a queue
def bfs(visited, graph, node):
   visited.append(node)
   queue.append(node)
   print("BFS traversal, start from node 'A': ")
   while queue:
      s = queue.pop(0)
      print(s, end=" ")
      for neighbour in graph[s]:
         if neighbour not in visited:
            visited.append(neighbour)
            queue.append(neighbour)

bfs(visited, graph, 'A')      # Driver Code
```

Output:
BFS traversal: start from node 'A':
A B C D E F G H

2. Write a Program to Implement Depth First Search using Python.




```
#Adjacency list
graph = {
        'A': ['B', 'C'],
        'B': ['D','E'],
        'C': ['F','G'],
        'D': ['H'],
        'E': ['H'],
        'F': ['H'],
        'G': ['H'],
        'H': []
        }
visited = set()  #Set to keep track of visited nodes.

def dfs(visited, graph, node):
    if node not in visited:
        print (node, end=" ")
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)


print("DFS traversal, start from node 'A': ")
dfs(visited, graph, 'A')   # Driver Code
```

Output:

DFS traversal, start from node 'A':

A B D H E C F G

3. **Write a Program to Implement 8-Puzzle problem using Python**

```
from collections import deque

# Helper function to find the position of the blank space (0)
def find_blank(state):
    for r in range(3):
        for c in range(3):
            if state[r][c] == 0:
                return r, c

# Check if the current state is the goal state
def is_goal(state):
    return state == goal_state

# Generate possible moves from the current state
def generate_moves(state):
    moves = []
    r, c = find_blank(state)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # up, down, left, right
    for dr, dc in directions:
        nr, nc = r + dr, c + dc
        if 0 <= nr < 3 and 0 <= nc < 3:
            new_state = [row[:] for row in state]  # Create a copy of the state
            new_state[r][c], new_state[nr][nc] = new_state[nr][nc], new_state[r][c]
            moves.append(new_state)
    return moves

# BFS algorithm to solve the 8-puzzle problem
def bfs(start_state):
    queue = deque([(start_state, [])])  # Queue stores (state, path)
    visited = set()

    visited.add(tuple(map(tuple, start_state)))  # Add the initial state to visited set
```

```
    while queue:
        current_state, path = queue.popleft()

        if is_goal(current_state):
            return path

        for move in generate_moves(current_state):
            move_tuple = tuple(map(tuple, move))  # Convert list to tuple to make it hashable
            if move_tuple not in visited:
                visited.add(move_tuple)
                queue.append((move, path + [move]))

    return None  # No solution found

# Function to print the state in a readable format
def print_state(state):
    for row in state:
        print(row)
    print()

# Code Begins
# Start state example
start_state = [[1, 2, 3], [4, 0, 5], [7, 8, 6]]
print("Start State:")
print_state(start_state)

# Define the goal state
goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

solution = bfs(start_state)

if solution:
    print("Solution found!")
    for step in solution:
        print_state(step)
```

```
else:
    print("No solution found!")
```

Output:

```
Start State:
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

Solution found!
[1, 2, 3]
[4, 5, 0]
[7, 8, 6]

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

4. Write a program to implement n-queens problem using python.

```
# Function to print the chessboard
def print_board(board):
    for i in range(N):
        for j in range(N):
            if board[i][j]==1:
              board[i][j]='Q'
            else:
              board[i][j]='*'
            print(board[i][j], end='  ')
        print()

# Function to check if it's safe to place a queen at board[row][col]
def is_safe(board, row, col, N):
    # Check the column
    for i in range(row):
        if board[i][col] == 1:
            return False

    # Check the upper left diagonal
    for i, j in zip(range(row - 1, -1, -1), range(col - 1, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check the upper right diagonal
    for i, j in zip(range(row - 1, -1, -1), range(col + 1, N)):
        if board[i][j] == 1:
            return False
    return True

# Function to solve N-Queens problem using backtracking
def solve_nqueens(board, row, N):
    # If all queens are placed, return True
    if row == N:
        return True
    # Try placing a queen in all columns for the current row
```

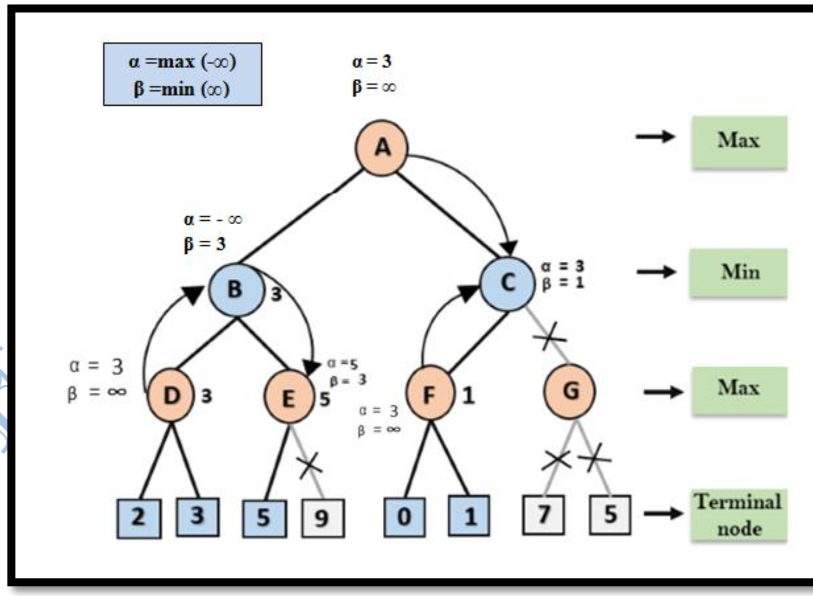
```
    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = 1        # Place the queen
            # Recur to place the next queen
            if solve_nqueens(board, row + 1, N):
                return True
            # If placing queen in board[row][col] doesn't lead to a solution, backtrack
            board[row][col] = 0

    return False  # If no place is found, return False

# Function to initialize the chessboard and call the solver
def nqueens(N):
    board = [[0 for _ in range(N)] for _ in range(N)]        # Initialize an empty
board

    if solve_nqueens(board, 0, N):
        print_board(board)   # If solution is found, print the board
    else:
        print("No solution exists")

N = 8
nqueens(N)
```

Output:

```
Q * * * * * * *
* * * * Q * * *
* * * * * * * Q
* * * * * Q * *
* * Q * * * * *
* * * * * * Q *
* Q * * * * * *
* * * Q * * * *
```

5. **Write a Program to Implement Alpha-Beta Pruning using Python.**

```
A, B = -1000, 1000

def AlphaBeta(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    best = A if maximizingPlayer else B
    for i in range(2):
        val = AlphaBeta(depth + 1, nodeIndex * 2 + i, not maximizingPlayer, values, alpha, beta)
        if maximizingPlayer:
            best = max(best, val)
            alpha = max(alpha, best)
        else:
            best = min(best, val)
            beta = min(beta, best)

        if beta <= alpha:
            break
    return best

values = [2, 3, 5, 9, 0, 1, 7, 5 ]
print("The optimal value is:",
AlphaBeta(0, 0, True, values, A, B))
```



Output:

The optimal value is: 3

6. **Write a program to implement forward chaining algorithm.**

```
symptoms = ["fever", "cough", "sore throat"]
diagnosis = {
    "f": "Flu",
    "fc": "Cold",
    "fcs": "COVID-19"
}
medicine = {
    "Flu": "Aspirin",
    "Cold": "Dolo 650",
    "COVID-19": "Paxlovid"
}

def main():
    print(f"\nSymptoms list: {symptoms}")
    facts = set()
    print("\nEnter symptoms in the given list (type 'done' when finished):")

    while True:
        symps = input("Symptom: ").lower().strip()
        if symps == 'done':
            break
        facts.add(symps)

    if 'fever' in facts:
        if 'cough' in facts and 'sore throat' in facts:
            Dx = diagnosis["fcs"]
        elif 'cough' in facts:
            Dx = diagnosis["fc"]
        else:
            Dx = diagnosis["f"]

else:
        Dx = None

    if Dx:
        print(f"\nPossible Diagnoses based on the symptoms: '{Dx}'")
        print(f"Recommended Medicine: {medicine[Dx]}")
```

```
 else:
        print("\nNo possible diagnosis based on the current symptoms")

if __name__ == "__main__":
    main()
```

Output:

Symptoms list: ['fever', 'cough', 'sore throat']

Enter symptoms in the given list (type 'done' when finished):
Symptom: fever
Symptom: cough
Symptom: done

Possible Diagnoses based on the symptoms: 'Cold'
Recommended Medicine: Dolo 650

7. Write a program to implement backward chaining algorithm.

```
Medicine = ["Aspirin","Dolo 650","Paxlovid"]
diagnosis = {
    "Aspirin":"Flu",
    "Dolo 650": "Cold",
    "Paxlovid": "COVID-19" }
symptoms = { "Flu" :"Fever",
    "Cold" :"Fever, Cough",
    "COVID-19" : "Fever, Cough, Sore throat"}

def main():
    print(f"\nMedicine list: {Medicine}")
    print("\nEnter Medicine name in the given list:")
    x = input()

    if x == 'Aspirin':
        Dx = diagnosis["Aspirin"]
    elif x == 'Dolo 650':
        Dx = diagnosis["Dolo 650"]
    elif x == 'Paxlovid':
        Dx = diagnosis["Paxlovid"]
    else:
        Dx = None

    if Dx:
        print(f"\nPossible Diagnoses based on the Medicine: '{Dx}'")
        print(f"Diagnoses symptoms: {symptoms[Dx]}")
    else:
        print("\nNo possible diagnosis based on the specified medicine")

if __name__ == "__main__":
    main()
```

Output:

```
Medicine list: ['Aspirin', 'Dolo 650', 'Paxlovid']
Enter Medicine name in the given list:
Dolo 650

Possible Diagnoses based on the Medicine: 'Cold'
Diagnoses symptoms: Fever, Cough
```

8. Write a program to implement KNN algorithm to classify Iris dataset. Print both correct and wrong predictions.

NOTE: Execute 8-12 programs in the 'colab google' online with internet.

Setp1: load iris dataset
Note: To copy this path: go to 'Google chrome' search 'iris dataset download' Then go to link 'gitHub' iris.csv -> right click on 'download' copy link address.
(we can also select any iris dataset link)

```
!wget https://archive.ics.uci.edu/static/public/53/iris.zip
```

```
!unzip iris.zip
```

Setp2: Code

```
# Import required libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
result=[]
# Load the Iris dataset       # copy correct file path from Dataset folder
iris_data = pd.read_csv('/content/iris.data')

X = iris_data.iloc[:, :-1].values   # Features
y = iris_data.iloc[:, -1].values    # Target variable


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=5)

# Create and train a KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the KNN classifier: {accuracy * 100:.2f}%')

# Print the correct and wrong predictions
for i in range(len(y_test)):
    if y_test[i] == y_pred[i]:
      result.append("Correct")
    else:
      result.append("Wrong")

status = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred, 'Result':result})
print(status)
```


Output:

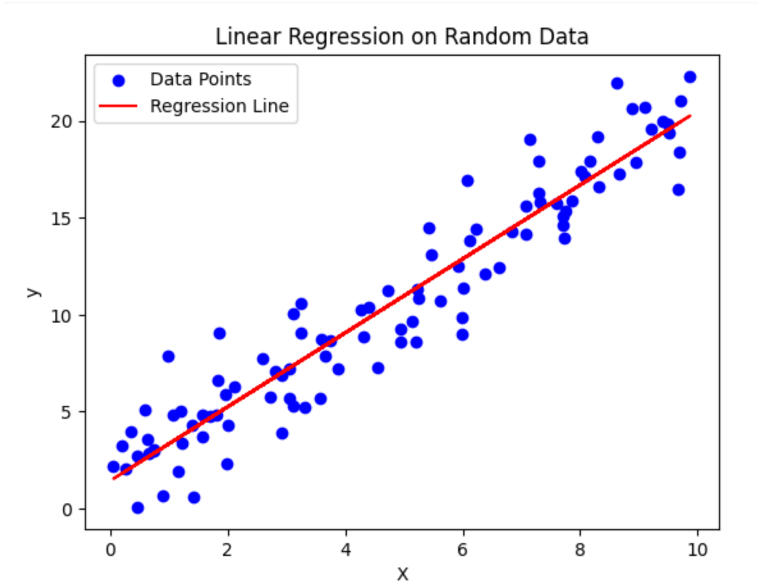
```
Accuracy of the KNN classifier: 95.56%
        Actual   Predicted   Result
0   Versicolor  Versicolor  Correct
1    Virginica   Virginica  Correct
2    Virginica   Virginica  Correct
3       Setosa      Setosa  Correct
4    Virginica   Virginica  Correct
5   Versicolor  Versicolor  Correct
6       Setosa      Setosa  Correct
7   Versicolor   Virginica    Wrong
8       Setosa      Setosa  Correct
9   Versicolor  Versicolor  Correct
10  Versicolor  Versicolor  Correct
11   Virginica   Virginica  Correct
12   Virginica   Virginica  Correct
13   Virginica   Virginica  Correct
14      Setosa      Setosa  Correct
15      Setosa      Setosa  Correct
16   Virginica   Virginica  Correct
17   Virginica   Virginica  Correct
18      Setosa      Setosa  Correct
19      Setosa      Setosa  Correct
20  Versicolor  Versicolor  Correct
21   Virginica   Virginica  Correct
22      Setosa      Setosa  Correct
```

```
23  Versicolor   Virginica    Wrong
24  Versicolor  Versicolor  Correct
25   Virginica   Virginica  Correct
26  Versicolor  Versicolor  Correct
27  Versicolor  Versicolor  Correct
28  Versicolor  Versicolor  Correct
29   Virginica   Virginica  Correct
30      Setosa      Setosa  Correct
31  Versicolor  Versicolor  Correct
32  Versicolor  Versicolor  Correct
33      Setosa      Setosa  Correct
34  Versicolor  Versicolor  Correct
35      Setosa      Setosa  Correct
36      Setosa      Setosa  Correct
37   Virginica   Virginica  Correct
38      Setosa      Setosa  Correct
39   Virginica   Virginica  Correct
40   Virginica   Virginica  Correct
41  Versicolor  Versicolor  Correct
42      Setosa      Setosa  Correct
43      Setosa      Setosa  Correct
44  Versicolor  Versicolor  Correct
```

9. Train a random data sample using linear regression model and plot the graph.

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate random data for training
np.random.seed(42)                          # Set seed for reproducibility
X = np.random.rand(100, 1) * 10             # 100 random data points between 0
and 10
y = 2 * X + 1 + np.random.randn(100, 1) * 2      # Linear relation with some
noise

# Create a Linear Regression model
model = LinearRegression()

# Train the model using the random data
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Plot the original data and the regression line
plt.scatter(X, y, color='blue', label='Data Points')  # Plotting the data points
plt.plot(X, y_pred, color='red', label='Regression Line')  # Plotting the regression
line
plt.title('Linear Regression on Random Data')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Output:


Linear Regression on Random Data
Data Points
Regression Line
y
X
0
5
10
15
20
0
2
4
6
8
10

10. Implement the naïve Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets.

Note: last four lines is optional

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Step 1: Load the dataset
# Assuming the CSV file has a header row with column names
data = pd.read_csv('/content/iris.data')    #copy correct file path from Dataset folder

# Step 2: Split the dataset into features (X) and target (y)
# Assume the last column is the target variable and the rest are features
X = data.iloc[:, :-1]  # Features (all columns except the last)
y = data.iloc[:, -1]   # Target (last column)

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=20)

# Step 4: Initialize and train the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Step 5: Predict on the test data
y_pred = model.predict(X_test)
# Step 6: Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Naive Bayes classifier: {accuracy * 100:.2f}%')

# Optionally, display predictions and actual values for comparison
print("\nPredictions vs Actual values:")
comparison = pd.DataFrame({'Predicted': y_pred, 'Actual': y_test})
print(comparison)
```

Output:

```
Accuracy of the Naive Bayes classifier: 93.33%

Predictions vs Actual values:
      Predicted      Actual
47       Setosa      Setosa
73   Versicolor  Versicolor
74   Versicolor  Versicolor
129   Virginica   Virginica
67   Versicolor  Versicolor
89   Versicolor  Versicolor
143   Virginica   Virginica
21       Setosa      Setosa
108   Virginica   Virginica
12       Setosa      Setosa
147   Virginica   Virginica
76   Versicolor  Versicolor
119  Versicolor   Virginica
35       Setosa      Setosa
28       Setosa      Setosa
122   Virginica   Virginica
13       Setosa      Setosa
58   Versicolor  Versicolor
114   Virginica   Virginica
57   Versicolor  Versicolor
50   Versicolor  Versicolor
149   Virginica   Virginica
111   Virginica   Virginica
20       Setosa      Setosa
72   Versicolor  Versicolor
81   Versicolor  Versicolor
98   Versicolor  Versicolor
34       Setosa      Setosa
104   Virginica   Virginica
133  Versicolor   Virginica
```

11. Demonstrate the working of SVM classifier for a suitable data set(e.g., iris dataset)

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset from the specified location
iris_data = pd.read_csv('/content/iris.data')

# Separate features (X) and target variable (y)
X = iris_data.iloc[:, :-1].values # Features
y = iris_data.iloc[:, -1].values # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=20)

# Step 3: Train the SVM classifier
svm_classifier = SVC(kernel='linear')  # Using a linear kernel
svm_classifier.fit(X_train, y_train)

# Step 4: Make predictions on the test set
y_pred = svm_classifier.predict(X_test)

# Step 5: Compute the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the SVM classifier: {accuracy * 100:.2f}%')


# Display confusion matrix
print("\nPredictions vs Actual values:")
comparison = pd.DataFrame({'Predicted': y_pred, 'Actual': y_test})
print(comparison)
```

Output:

```
Accuracy of the SVM classifier: 96.67%

Predictions vs Actual values:
     Predicted      Actual
0       Setosa      Setosa
1   Versicolor  Versicolor
2   Versicolor  Versicolor
3    Virginica   Virginica
4   Versicolor  Versicolor
5   Versicolor  Versicolor
6    Virginica   Virginica
7       Setosa      Setosa
8    Virginica   Virginica
9       Setosa      Setosa
10   Virginica   Virginica
11  Versicolor  Versicolor
12   Virginica   Virginica
13      Setosa      Setosa
14      Setosa      Setosa
15   Virginica   Virginica
16      Setosa      Setosa
17  Versicolor  Versicolor
18   Virginica   Virginica
19  Versicolor  Versicolor
20  Versicolor  Versicolor
21   Virginica   Virginica
22   Virginica   Virginica
23      Setosa      Setosa
24  Versicolor  Versicolor
25  Versicolor  Versicolor
26  Versicolor  Versicolor
27      Setosa      Setosa
28   Virginica   Virginica
29  Versicolor   Virginica
```

12. Build a sample binary image classification model (cat and dog).

Step1: Set: Runtime -> Change Runtime type -> GPU

Step2: load Cat vs Dog dataset

```
!wget --no-check-certificate \
   https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
   -O /tmp/cats_and_dogs_filtered.zip
```

```
!unzip /tmp/cats_and_dogs_filtered.zip
```

Setp3: Code

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Define constants
batch_size = 32
img_height = 150
img_width = 150
epochs = 10
# Create image data generators
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
'/content/cats_and_dogs_filtered/train',
target_size=(img_height, img_width),
batch_size=batch_size,
class_mode='binary'
)
validation_generator = validation_datagen.flow_from_directory(
'/content/cats_and_dogs_filtered/validation',
target_size=(img_height, img_width),
batch_size=batch_size,
class_mode='binary'
)
```

```
# Build a simple neural network model
model = Sequential([  Flatten(input_shape=(img_height, img_width, 3)),
Dense(128, activation='relu'), Dense(1, activation='sigmoid')
])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit( train_generator,
steps_per_epoch=train_generator.samples // batch_size,
epochs=epochs,   validation_data=validation_generator,
validation_steps=validation_generator.samples   // batch_size
)
```

Output:

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 6s 78ms/step - accuracy: 0.5477 - loss: 13.6614 - val_accuracy: 0.5746 - val_loss: 1.8352
Epoch 2/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.4375 - loss: 2.2653 - val_accuracy: 0.5706 - val_loss: 1.7689
Epoch 3/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 5s 83ms/step - accuracy: 0.5540 - loss: 2.9765 - val_accuracy: 0.5575 - val_loss: 1.2186
Epoch 4/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.5000 - loss: 1.3509 - val_accuracy: 0.5111 - val_loss: 2.9195
Epoch 5/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 5s 83ms/step - accuracy: 0.5651 - loss: 2.6707 - val_accuracy: 0.5423 - val_loss: 3.1215
Epoch 6/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.2812 - loss: 4.3689 - val_accuracy: 0.5716 - val_loss: 1.5851
Epoch 7/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 5s 76ms/step - accuracy: 0.6277 - loss: 1.6029 - val_accuracy: 0.5171 - val_loss: 2.4660
Epoch 8/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.6875 - loss: 1.1648 - val_accuracy: 0.5030 - val_loss: 3.6614
Epoch 9/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.5722 - loss: 2.0467 - val_accuracy: 0.5514 - val_loss: 1.3998
Epoch 10/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.5000 - loss: 1.2462 - val_accuracy: 0.5212 - val_loss: 2.4486
```