SHREE KRISHNA KANTH S

225229136

II MSc DATA SCIENCE - "A" ¶

PDL LAB 5 (TAMIL)

In [1]:
```python
import numpy as np

def load_data():
    # Load motivational quotes and demotivational quotes from files
    with open('Happy_Quotes.txt', 'r', encoding='utf-8') as f:
        Happy_Quotes = f.readlines()
    with open('Sad_Quotes.txt', 'r', encoding='utf-8') as f:
        Sad_Quotes = f.readlines()

    # Combine both classes of quotes and create labels (1 for motivational, 0 for demotivational)
    quotes = Happy_Quotes + Sad_Quotes
    labels = np.concatenate([np.ones(len(Happy_Quotes)), np.zeros(len(Sad_Quotes))])

    return quotes, labels

quotes, labels = load_data()
```

In [2]:

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

def preprocess_data(quotes):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(quotes)
    sequences = tokenizer.texts_to_sequences(quotes)
    vocab_size = len(tokenizer.word_index) + 1

    # Pad the sequences to have the same length
    max_sequence_length = max(len(seq) for seq in sequences)
    padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post')

    return padded_sequences, vocab_size

X, vocab_size = preprocess_data(quotes)
```

In [3]:

```
load_data()
```

Out[3]:
```
(['Sure, here are 20 happy quotes in Tamil:\n',
  '\n',
  '1. மகிழ்ச்சி ஒரு அமைதி புதிய ஒளி ஆகும்.\n',
  '2. எப்போதும் சந்தோஷமாக இருங்கள், வாழ்க்கை நலமாக இருக்கும்.\n',
  '3. சிரிப்பு ஒரு மிகுந்த அரிய மருத்துவம்.\n',
  '4. மகிழ்ச்சியால் மனிதர் சிறகுகள் விழுங்குகின்றன.\n',
  '5. மகிழ்ச்சியே மகிழ்ச்சி! அது உங்களுக்கு புதிய நாள் கொடுக்கும்.\n',
  '6. மகிழ்ச்சி எப்போதும் உங்களுக்கு நலமாக இருக்க வேண்டும்.\n',
  '7. சிரிப்பு வாழ்க்கைக்கு குடியும் அமையும்.\n',
  '8. மகிழ்ச்சி உங்கள் இரவுகளையும் நாள்களையும் சுவாரஸ்யமாக மாற்றும்.\n',
  '9. சிரிப்பு உங்கள் மனதில் இருக்கும்போது நீங்கள் உயிருக்குள் கிடைக்கும்.\n',
  '10. மகிழ்ச்சி உங்கள் காரியங்களில் வாழ்வின் ரகசியம் ஆகும்.\n',
  '11. மகிழ்ச்சி மனதில் உள்ளது மற்றும் உலகத்தின் உள்ளங்களைப் பகிரும்.\n',
  '12. சிரிப்பு எல்லா துறைகளின் மேலும் ஒளியாக உள்ளது.\n',
  '13. மகிழ்ச்சி மனிதரின் முகத்தில் ஒளியாக விளங்குகிறது.\n',
  '14. சிரிப்பு உன் வாழ்க்கைக்கு புது முகம் கொடுக்கும்.\n',
  '15. மகிழ்ச்சி நாள் நாளாக வருகின்றது.\n',
  '16. சிரிப்பு எப்போதும் அன்பு தருகின்றது.\n',
  '17. மகிழ்ச்சி ஒரு மகிழ்ச்சி மற்றும் அன்பு ஆகும்.\n',
  '18. சிரிப்பு ஒரு சந்தோஷம் மற்றும் மகிழ்ச்சியைத் தருகின்றது.\n',
  '19. மகிழ்ச்சி உன் மனதை புதுமாக செய்கிறது.\n',
  '20. சிரிப்பு உன்னை நலமாக உள்ளது மற்றும் உங்கள் சுவாரஸ்யத்தை அதிகரிக்கின்றது.\n',
  '\n',
  '1. விருத்தத்தின் பேரின்பில் தான் பலமுடியும், அதன் அடிப்படையில் உனக்கு நோய் வந்துவிட்டால் உன் நிலைக்கு மிகுந்த பல
முடியும்.\n',
  '\n',
  '2. மனம் துன்பத்தை எண்ணி நிறுத்த வந்தால், அந்த விருத்தம் அதிகமாக உனக்கு அர்த்தம் ஆகும்.\n',
  '\n',
  '3. விருத்தத்தின் அடிப்படையில் பரிகாரங்கள் என்னும் பொருள்கள் மிகுந்த சிந்தனைகளை ஏற்படுத்தும்.\n',
  '\n',
  '4. குணமான நடனம் கொடுக்க விடும், ஆனால் அதன் பின் நீ விருத்தத்தை இழந்தாய் போகலாம்.\n',
  '\n',
  '5. உனக்கு விருத்தம் எப்போதும் கிடைக்க வந்தால், அந்த விருத்தம் உனக்கு நலமாக இருக்கும்.\n',
  '\n',
  '6. விருத்தம் வந்து விட்டால் நீ துன்பமாகிறாய், அதன் காரணமாக நீ பிரியமாகிறாய்.\n',
  '\n',
  '7. மனம் வருத்தம் கொடுத்தால் துன்பம் பெருகின்றது, ஆனால் அதன் பின் நீ விருத்தத்தை இழந்திட வேண்டும்.\n',
  '\n',
  '8. குணமான விருத்தத்துக்கு நீயே காரணம், அது எப்போதும் நீயே மறைக்கும்.\n',
  '\n',
  '9. உனக்கு நிலைத்து நிறுத்த முடியாத துன்பங்கள் எப்போதும் உன்னை மெல்ல விடும்.\n',
  '\n',
  '10. விருத்தம் வந்த போது துன்பம் வருவது எப்போதும் அடையவில்லை, அதன் பின் நீ விருத்தம் இழந்தாய் போகலாம்.\n',
  '\n',
  '11. நீ மனம் வருத்தம் கொடுத்தால் நீ பிரியமாகிறாய், ஆனால் அதன் பின் நீ விருத்தத்தை இழந்திட வேண்டும்.\n',
  '\n',
```

```
 '12. விருத்தத்தின் பின் உனக்கு எந்த நலமும் இல்லை, அதன் பேரில் உனக்கு நிலைத்து நிறுத்த முடியும்.\n',
 '\n',
 '13. விருத்தம் கொடுத்து மிகுந்த பயன் கிடைக்கின்றது, ஆனால் அதன் பின் உன்னை நிலைத்து நிறுத்த முடியாது.\n',
 '\n',
 '14. விருத\n',
 '\n',
 '◌தம் கொடுத்து வந்தால் துன்பம் பெருகின்றது, ஆனால் அதன் பின் நீ விருத்தத்தை இழந்தாய் போகலாம்.\n',
 '\n',
 '15. நீ விருத்தம் கொடுத்து வாழ்கின்றாய், அது எப்போதும் நீயே மறைக்கும்.\n',
 '\n',
 '16. விருத்தம் கொடுக்க விட்டால் நீ துன்பமாகிறாய், அதன் காரணமாக நீ நிலைத்து நிறுத்த வேண்டும்.\n',
 '\n',
 '17. விருத்தத்துக்கு உன் கையில் சிறந்த வாய்க்குத்துணை அந்த விருத்தம் எப்போதும் உனக்கு அர்த்தமாக இருக்கும்.\n',
 '\n',
 '18. விருத்தத்துக்கு நீ பலம் தரக்கூடிய மகிழ்ச்சி, அது எப்போதும் நீயே அடைகின்றது.\n',
 '\n',
 '19. உனக்கு விருத்தம் வந்தால் நீ நிலைத்து நிறுத்த முடியாத துன்பங்கள் எப்போதும் உன்னை மெல்ல விடும்.\n',
 '\n',
 '20. விருத்தம் எப்போதும் நீயே காரணமாகிறது, அது எப்போதும் நீயே விருத்தமாக இருக்கும்.\n',
 '\n'],
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]))
```

In [4]:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

def create_model(nodes=32, layers=1):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=nodes, input_length=X.shape[1]))

    for _ in range(layers):
        model.add(LSTM(nodes, return_sequences=True))
    model.add(LSTM(nodes))

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

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

In [5]:

```python
from sklearn.model_selection import train_test_split
import time

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, model.count_params(), end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1]

# Create a dictionary to store the results for each configuration
results = {}

# Evaluate each configuration and store the components
for nodes in nodes_list:
    for layers in layers_list:
        train_accuracy, test_accuracy, num_params, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
            "Test Accuracy": test_accuracy,
            "Parameters Learned": num_params,
            "Running Time": running_time
        }

# Find the best configuration based on the highest testing accuracy
best_config = max(results, key=lambda x: results[x]["Test Accuracy"])

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("# Parameters Learned:", results[best_config]["Parameters Learned"])
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
print("Running Time:", results[best_config]["Running Time"])
```

```
2/2 [==============================] - 1s 21ms/step - loss: 0.4862 - accuracy: 0.8462
1/1 [==============================] - 0s 29ms/step - loss: 0.5544 - accuracy: 0.6923
Epoch 1/5
1/1 [==============================] - 4s 4s/step - loss: 0.6927 - accuracy: 0.5577
Epoch 2/5
1/1 [==============================] - 1s 1s/step - loss: 0.6840 - accuracy: 0.6538
Epoch 3/5
1/1 [==============================] - 1s 1s/step - loss: 0.7700 - accuracy: 0.3846
Epoch 4/5
1/1 [==============================] - 1s 1s/step - loss: 0.6668 - accuracy: 0.7885
Epoch 5/5
1/1 [==============================] - 1s 1s/step - loss: 0.6274 - accuracy: 0.6538
2/2 [==============================] - 1s 61ms/step - loss: 0.5885 - accuracy: 0.6538
1/1 [==============================] - 0s 65ms/step - loss: 0.6676 - accuracy: 0.6154
BEST CONFIGURATION: (512, 1)
Components of the Best Configuration:
# Parameters Learned: 4289025
Training Accuracy: 0.8461538553237915
Testing Accuracy: 0.692307710647583
Running Time: 4.612245082855225
```

In [6]:

```
from sklearn.model_selection import train_test_split
import time

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, model.count_params(), end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1, 2, 3, 4, 5]

# Create a dictionary to store the results for each configuration
results = {}

# Evaluate each configuration and store the components
for nodes in nodes_list:
    if nodes == 32:  # Evaluate all layers for node 32
        for layers in layers_list:
            train_accuracy, test_accuracy, num_params, running_time = train_and_evaluate_model(nodes, layers)
            results[(nodes, layers)] = {
                "Train Accuracy": train_accuracy,
                "Test Accuracy": test_accuracy,
                "Parameters Learned": num_params,
                "Running Time": running_time
            }
    else:  # Only evaluate the first layer for other nodes
        layers = 1
        train_accuracy, test_accuracy, num_params, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
            "Test Accuracy": test_accuracy,
            "Parameters Learned": num_params,
            "Running Time": running_time
        }

# Find the best configuration based on the highest testing accuracy
best_config = max(results, key=lambda x: results[x]["Test Accuracy"])
```

```python

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("# Parameters Learned:", results[best_config]["Parameters Learned"])
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
print("Running Time:", results[best_config]["Running Time"])

```

```
2/2 [==============================] - 1s 21ms/step - loss: 0.4618 - accuracy: 0.8846
1/1 [==============================] - 0s 33ms/step - loss: 0.5423 - accuracy: 0.6923
Epoch 1/5
1/1 [==============================] - 5s 5s/step - loss: 0.6912 - accuracy: 0.5962
Epoch 2/5
1/1 [==============================] - 1s 1s/step - loss: 0.6874 - accuracy: 0.6538
Epoch 3/5
1/1 [==============================] - 1s 1s/step - loss: 0.7737 - accuracy: 0.3654
Epoch 4/5
1/1 [==============================] - 1s 1s/step - loss: 0.6681 - accuracy: 0.8462
Epoch 5/5
1/1 [==============================] - 1s 1s/step - loss: 0.6280 - accuracy: 0.6538
2/2 [==============================] - 1s 72ms/step - loss: 0.5805 - accuracy: 0.6538
1/1 [==============================] - 0s 74ms/step - loss: 0.6617 - accuracy: 0.6154
BEST CONFIGURATION: (512, 1)
Components of the Best Configuration:
# Parameters Learned: 4289025
Training Accuracy: 0.8846153616905212
Testing Accuracy: 0.692307710647583
Running Time: 5.205701112747192
```

In [7]:

```python
from sklearn.model_selection import train_test_split
import time

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, model.count_params(), end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1, 2, 3, 4, 5]

# Create a dictionary to store the results for each configuration
results = {}

# Find the best configuration
best_accuracy = 0
best_config = None

# Evaluate each configuration and store the components
for nodes in nodes_list:
    if nodes == 32:  # Evaluate all layers for node 32
        for layers in layers_list:
            train_accuracy, test_accuracy, num_params, running_time = train_and_evaluate_model(nodes, layers)
            results[(nodes, layers)] = {
                "Train Accuracy": train_accuracy,
                "Test Accuracy": test_accuracy,
                "Parameters Learned": num_params,
                "Running Time": running_time
            }
            if test_accuracy > best_accuracy:
                best_accuracy = test_accuracy
                best_config = (nodes, layers)
    else:  # Only evaluate the first layer for other nodes
        layers = 1
        train_accuracy, test_accuracy, num_params, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
```

```
            "Test Accuracy": test_accuracy,
            "Parameters Learned": num_params,
            "Running Time": running_time
        }
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_config = (nodes, layers)

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("# Parameters Learned:", results[best_config]["Parameters Learned"])
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
print("Running Time:", results[best_config]["Running Time"])

```

```
2/2 [==============================] - 1s 22ms/step - loss: 0.4941 - accuracy: 0.8269
1/1 [==============================] - 0s 31ms/step - loss: 0.5579 - accuracy: 0.6154
Epoch 1/5
1/1 [==============================] - 4s 4s/step - loss: 0.6906 - accuracy: 0.6538
Epoch 2/5
1/1 [==============================] - 1s 1s/step - loss: 0.6839 - accuracy: 0.6538
Epoch 3/5
1/1 [==============================] - 1s 1s/step - loss: 0.7548 - accuracy: 0.3654
Epoch 4/5
1/1 [==============================] - 1s 1s/step - loss: 0.6656 - accuracy: 0.8269
Epoch 5/5
1/1 [==============================] - 1s 1s/step - loss: 0.6203 - accuracy: 0.6538
2/2 [==============================] - 1s 60ms/step - loss: 0.5655 - accuracy: 0.6538
1/1 [==============================] - 0s 64ms/step - loss: 0.6532 - accuracy: 0.6154
BEST CONFIGURATION: (6, 1)
Components of the Best Configuration:
# Parameters Learned: 1687
Training Accuracy: 0.6538461446762085
Testing Accuracy: 0.6153846383094788
Running Time: 2.949039936065674
```

In [8]:
```
from sklearn.model_selection import train_test_split
import time

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1, 2, 3, 4, 5]

# Create a dictionary to store the results for each configuration
results = {}

# Find the best configuration
best_accuracy = 0
best_config = None

# Evaluate each configuration and store the components
for nodes in nodes_list:
    if nodes == 32:  # Evaluate all layers for node 32
        for layers in layers_list:
            train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
            results[(nodes, layers)] = {
                "Train Accuracy": train_accuracy,
                "Test Accuracy": test_accuracy,
                "Running Time": running_time
            }
            if test_accuracy > best_accuracy:
                best_accuracy = test_accuracy
                best_config = (nodes, layers)
    else:  # Only evaluate the first layer for other nodes
        layers = 1
        train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
            "Test Accuracy": test_accuracy,
```

```python
            "Running Time": running_time
        }
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_config = (nodes, layers)

# Print the results for all configurations
for config, values in results.items():
    print("Node: {}, Layers: {}".format(config[0], config[1]))
    print("Training Accuracy:", values["Train Accuracy"])
    print("Testing Accuracy:", values["Test Accuracy"])
    print("Running Time:", values["Running Time"])
    print("-" * 30)

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
print("Running Time:", results[best_config]["Running Time"])
```

```
------------------------------
Node: 1024, Layers: 1
Training Accuracy: 0.6538461446762085
Testing Accuracy: 0.6153846383094788
Running Time: 8.345974683761597
------------------------------
BEST CONFIGURATION: (512, 1)
Components of the Best Configuration:

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [8], in <cell line: 64>()
     62 print("BEST CONFIGURATION:", best_config)
     63 print("Components of the Best Configuration:")
---> 64 print("# Parameters Learned:", results[best_config]["Parameters Learned"])
     65 print("Training Accuracy:", results[best_config]["Train Accuracy"])
     66 print("Testing Accuracy:", results[best_config]["Test Accuracy"])

KeyError: 'Parameters Learned'
```

In [10]:

```
from sklearn.model_selection import train_test_split
import time
import matplotlib.pyplot as plt

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1, 2, 3, 4, 5]

# Create a dictionary to store the results for each configuration
results = {}

# Find the best configuration
best_accuracy = 0
best_config = None

# Evaluate each configuration and store the components
for nodes in nodes_list:
    if nodes == 32:  # Evaluate all layers for node 32
        for layers in layers_list:
            train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
            results[(nodes, layers)] = {
                "Train Accuracy": train_accuracy,
                "Test Accuracy": test_accuracy,
                "Running Time": running_time
            }
            if test_accuracy > best_accuracy:
                best_accuracy = test_accuracy
                best_config = (nodes, layers)
    else:  # Only evaluate the first layer for other nodes
        layers = 1
        train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
```

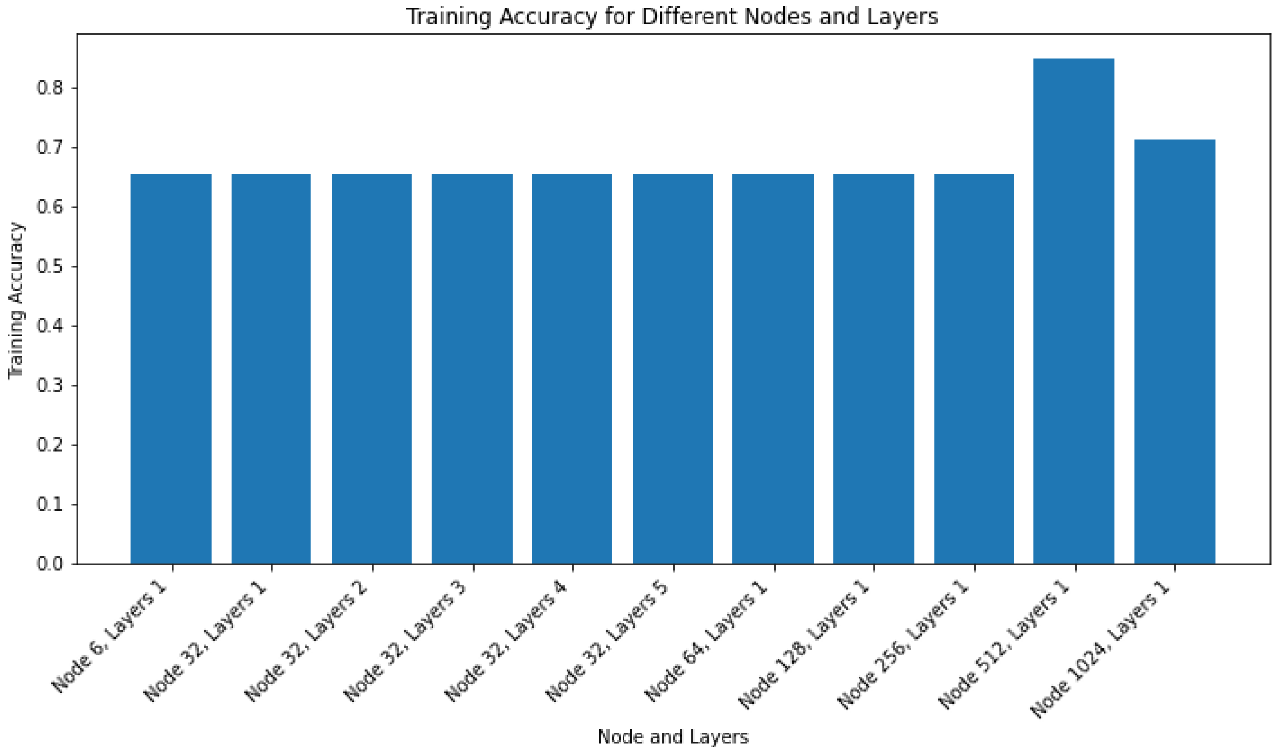
```python
            "Test Accuracy": test_accuracy,
            "Running Time": running_time
        }
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_config = (nodes, layers)

# Print the results for all configurations
for config, values in results.items():
    node, layers = config
    print("Node: {}, Layers: {}".format(node, layers))
    print("Training Accuracy:", values["Train Accuracy"])
    print("Testing Accuracy:", values["Test Accuracy"])
    print("Running Time:", values["Running Time"])
```

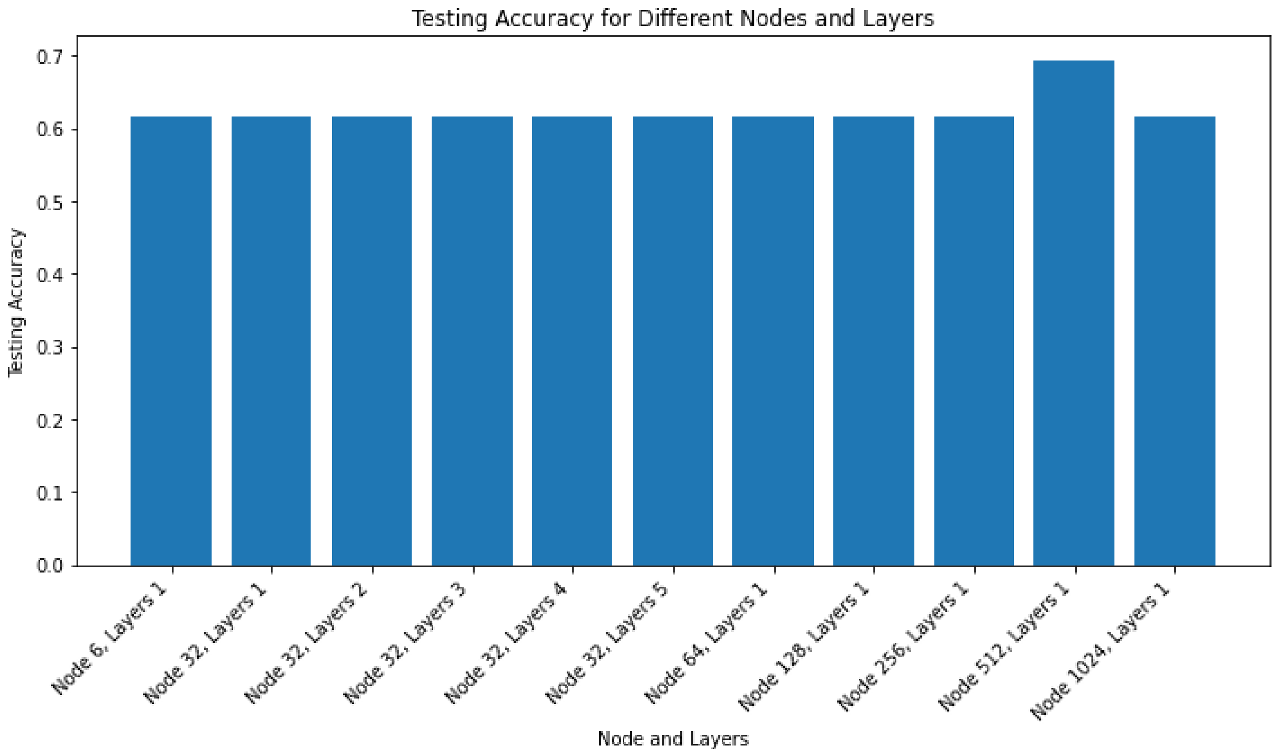
```
Node: 64, Layers: 1
Training Accuracy: 0.6538461446762085
Testing Accuracy: 0.6153846383094788
Running Time: 3.1060950756073
Node: 128, Layers: 1
Training Accuracy: 0.6538461446762085
Testing Accuracy: 0.6153846383094788
Running Time: 3.153494119644165
Node: 256, Layers: 1
Training Accuracy: 0.6538461446762085
Testing Accuracy: 0.6153846383094788
Running Time: 3.463649034500122
Node: 512, Layers: 1
Training Accuracy: 0.8461538553237915
Testing Accuracy: 0.692307710647583
Running Time: 4.425907373428345
Node: 1024, Layers: 1
Training Accuracy: 0.7115384340286255
Testing Accuracy: 0.6153846383094788
Running Time: 8.258821249008179
```

In [12]:

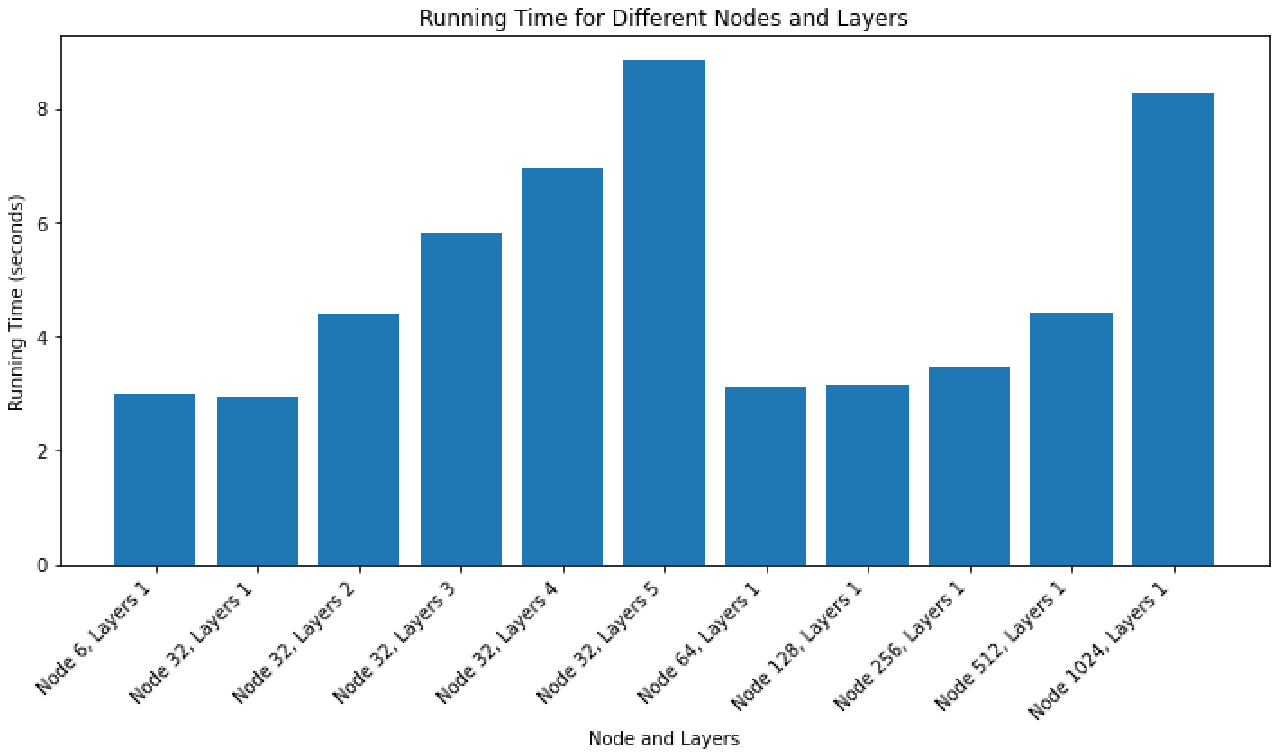
```
# Plot the graphs for Training Accuracy, Testing Accuracy, and Running Time for all nodes with their layers
x_labels = ["Node {}, Layers {}".format(node, layers) for node, layers in results.keys()]
train_accuracies = [values["Train Accuracy"] for values in results.values()]
test_accuracies = [values["Test Accuracy"] for values in results.values()]
running_times = [values["Running Time"] for values in results.values()]

# Plot the bar graph for Training Accuracy
plt.figure(figsize=(10, 6))
plt.bar(x_labels, train_accuracies)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Training Accuracy")
plt.title("Training Accuracy for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Plot the bar graph for Testing Accuracy
plt.figure(figsize=(10, 6))
plt.bar(x_labels, test_accuracies)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Testing Accuracy")
plt.title("Testing Accuracy for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Plot the bar graph for Running Time
plt.figure(figsize=(10, 6))
plt.bar(x_labels, running_times)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Running Time (seconds)")
plt.title("Running Time for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
```

```
42 print("Running Time:", results[best_config]["Running Time"])
```





Testing Accuracy for Different Nodes and Layers
Testing Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Node 6, Layers 1
Node 32, Layers 1
Node 32, Layers 2
Node 32, Layers 3
Node 32, Layers 4
Node 32, Layers 5
Node 64, Layers 1
Node 128, Layers 1
Node 256, Layers 1
Node 512, Layers 1
Node 1024, Layers 1
Node and Layers




```
BEST CONFIGURATION: (512, 1)
Components of the Best Configuration:
Training Accuracy: 0.8461538553237915
Testing Accuracy: 0.692307710647583
Running Time: 4.425907373428345
```

In [13]:

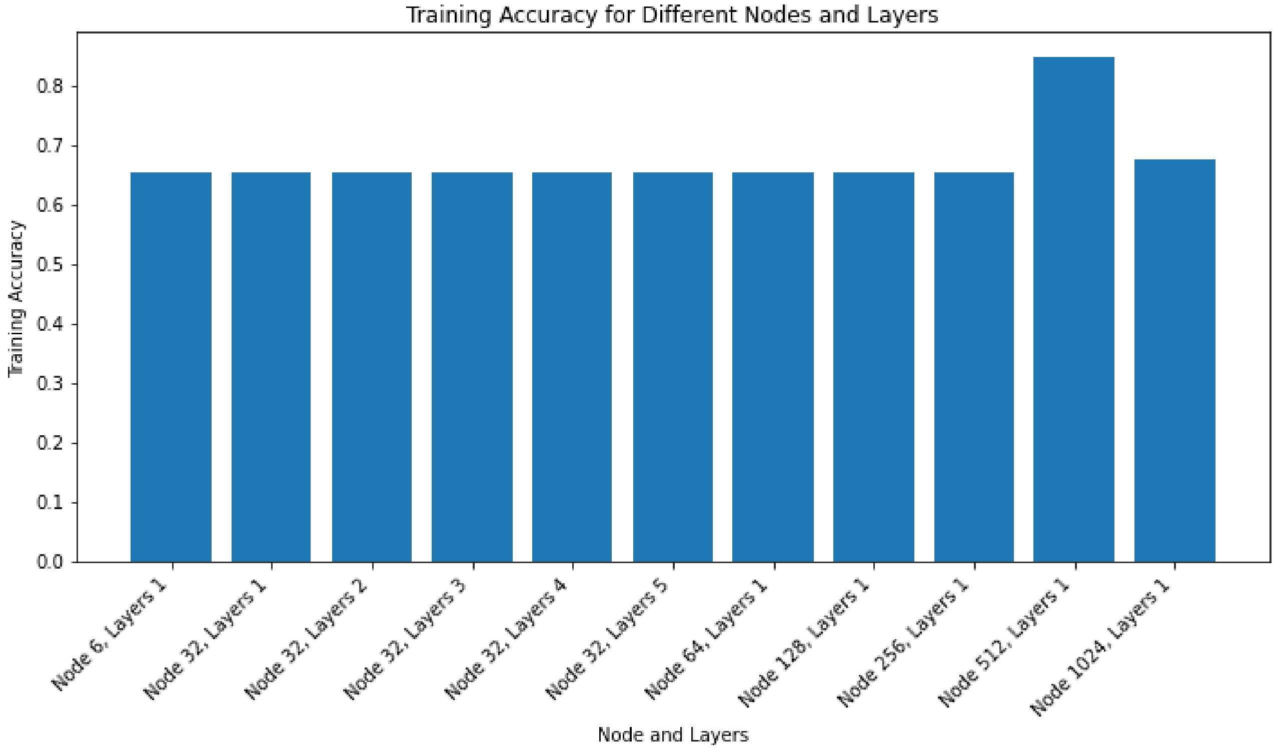
```
from sklearn.model_selection import train_test_split
import time
import matplotlib.pyplot as plt

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1, 2, 3, 4, 5]

# Create a dictionary to store the results for each configuration
results = {}

# Find the best configuration
best_accuracy = 0
best_config = None

# Evaluate each configuration and store the components
for nodes in nodes_list:
    if nodes == 32:  # Evaluate all layers for node 32
        for layers in layers_list:
            train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
            results[(nodes, layers)] = {
                "Train Accuracy": train_accuracy,
                "Test Accuracy": test_accuracy,
                "Running Time": running_time
            }
            if test_accuracy > best_accuracy:
                best_accuracy = test_accuracy
                best_config = (nodes, layers)
    else:  # Only evaluate the first layer for other nodes
        layers = 1
        train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
```

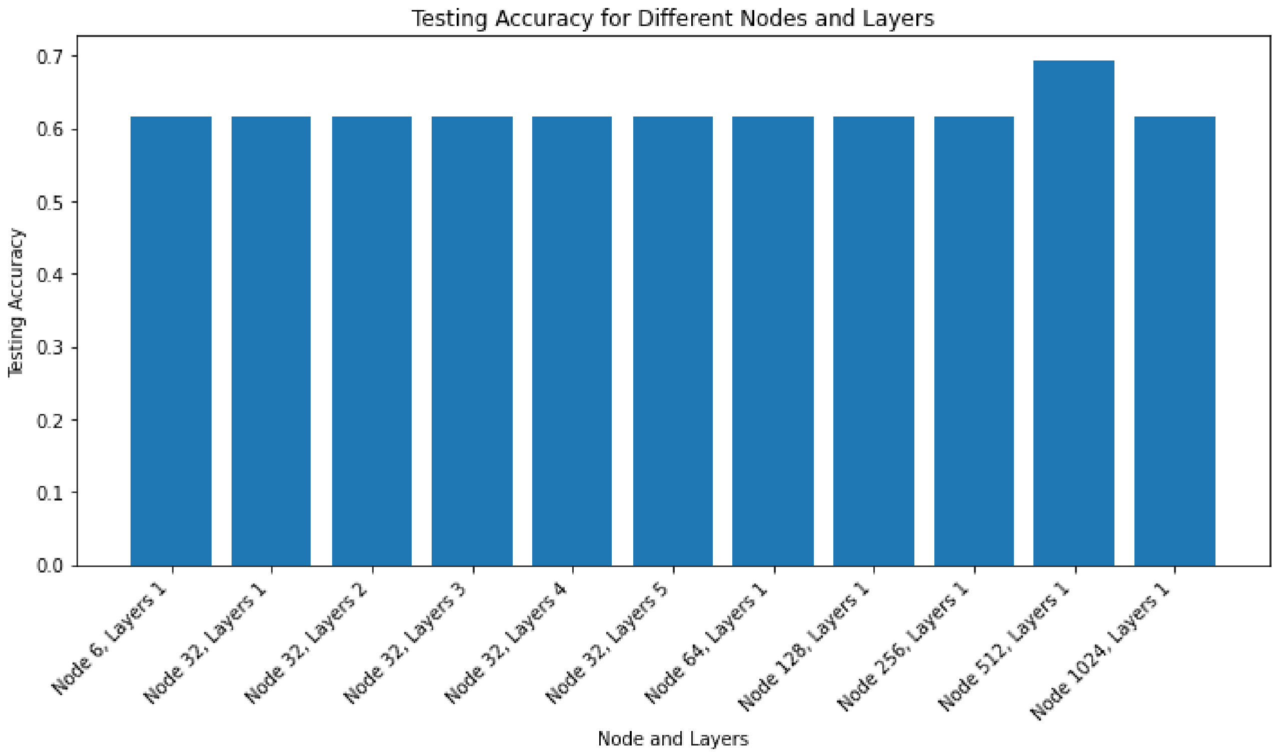
```python
            "Test Accuracy": test_accuracy,
            "Running Time": running_time
        }
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_config = (nodes, layers)

# Print the results for all configurations
for config, values in results.items():
    node, layers = config
    print("Node: {}, Layers: {}".format(node, layers))
    print("Training Accuracy:", values["Train Accuracy"])
    print("Testing Accuracy:", values["Test Accuracy"])
    print("Running Time:", values["Running Time"])
    print("-" * 30)
```

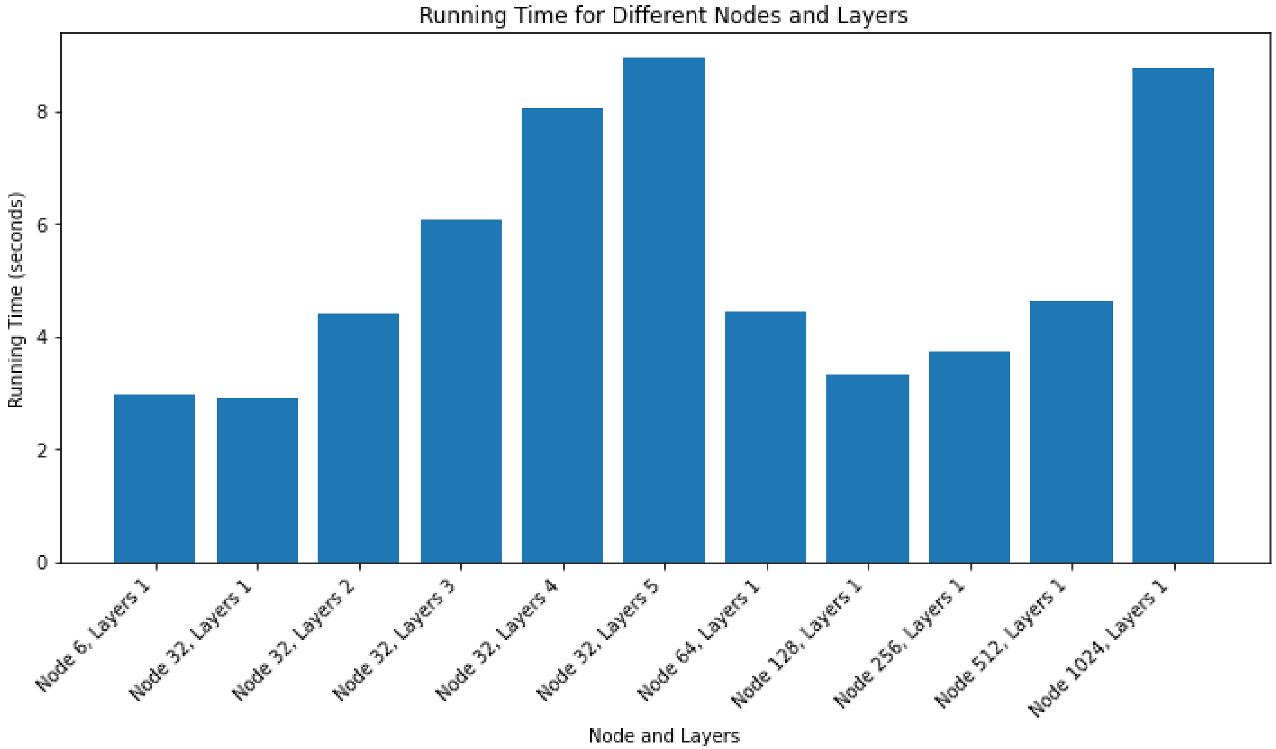
```
Epoch 3/5
1/1 [==============================] - 0s 115ms/step - loss: 0.6449 - accuracy: 0.6538
Epoch 4/5
1/1 [==============================] - 0s 112ms/step - loss: 0.6287 - accuracy: 0.6538
Epoch 5/5
1/1 [==============================] - 0s 112ms/step - loss: 0.6189 - accuracy: 0.6538
2/2 [==============================] - 1s 14ms/step - loss: 0.5802 - accuracy: 0.6538
1/1 [==============================] - 0s 23ms/step - loss: 0.6474 - accuracy: 0.6154
Epoch 1/5
1/1 [==============================] - 3s 3s/step - loss: 0.6942 - accuracy: 0.4423
Epoch 2/5
1/1 [==============================] - 0s 328ms/step - loss: 0.6539 - accuracy: 0.6538
Epoch 3/5
1/1 [==============================] - 0s 328ms/step - loss: 0.6501 - accuracy: 0.6538
Epoch 4/5
1/1 [==============================] - 0s 333ms/step - loss: 0.6019 - accuracy: 0.6538
Epoch 5/5
1/1 [==============================] - 0s 323ms/step - loss: 0.5536 - accuracy: 0.6731
2/2 [==============================] - 1s 29ms/step - loss: 0.4299 - accuracy: 0.8462
1/1 [==============================] - 0s 39ms/step - loss: 0.4959 - accuracy: 0.6923
```

In [14]:
```
# Plot the graphs for Training Accuracy, Testing Accuracy, and Running Time for all nodes with their layers
x_labels = ["Node {}, Layers {}".format(node, layers) for node, layers in results.keys()]
train_accuracies = [values["Train Accuracy"] for values in results.values()]
test_accuracies = [values["Test Accuracy"] for values in results.values()]
running_times = [values["Running Time"] for values in results.values()]

# Plot the bar graph for Training Accuracy
plt.figure(figsize=(10, 6))
plt.bar(x_labels, train_accuracies)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Training Accuracy")
plt.title("Training Accuracy for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Plot the bar graph for Testing Accuracy
plt.figure(figsize=(10, 6))
plt.bar(x_labels, test_accuracies)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Testing Accuracy")
plt.title("Testing Accuracy for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Plot the bar graph for Running Time
plt.figure(figsize=(10, 6))
plt.bar(x_labels, running_times)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Running Time (seconds)")
plt.title("Running Time for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
print("Running Time:", results[best_config]["Running Time"])
```

43


Training Accuracy for Different Nodes and Layers
Training Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Node 6, Layers 1
Node 32, Layers 1
Node 32, Layers 2
Node 32, Layers 3
Node 32, Layers 4
Node 32, Layers 5
Node 64, Layers 1
Node 128, Layers 1
Node 256, Layers 1
Node 512, Layers 1
Node 1024, Layers 1
Node and Layers


Testing Accuracy for Different Nodes and Layers
Testing Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Node 6, Layers 1
Node 32, Layers 1
Node 32, Layers 2
Node 32, Layers 3
Node 32, Layers 4
Node 32, Layers 5
Node 64, Layers 1
Node 128, Layers 1
Node 256, Layers 1
Node 512, Layers 1
Node 1024, Layers 1
Node and Layers




```
BEST CONFIGURATION: (512, 1)
Components of the Best Configuration:
Training Accuracy: 0.8461538553237915
Testing Accuracy: 0.692307710647583
Running Time: 4.621887445449829
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: