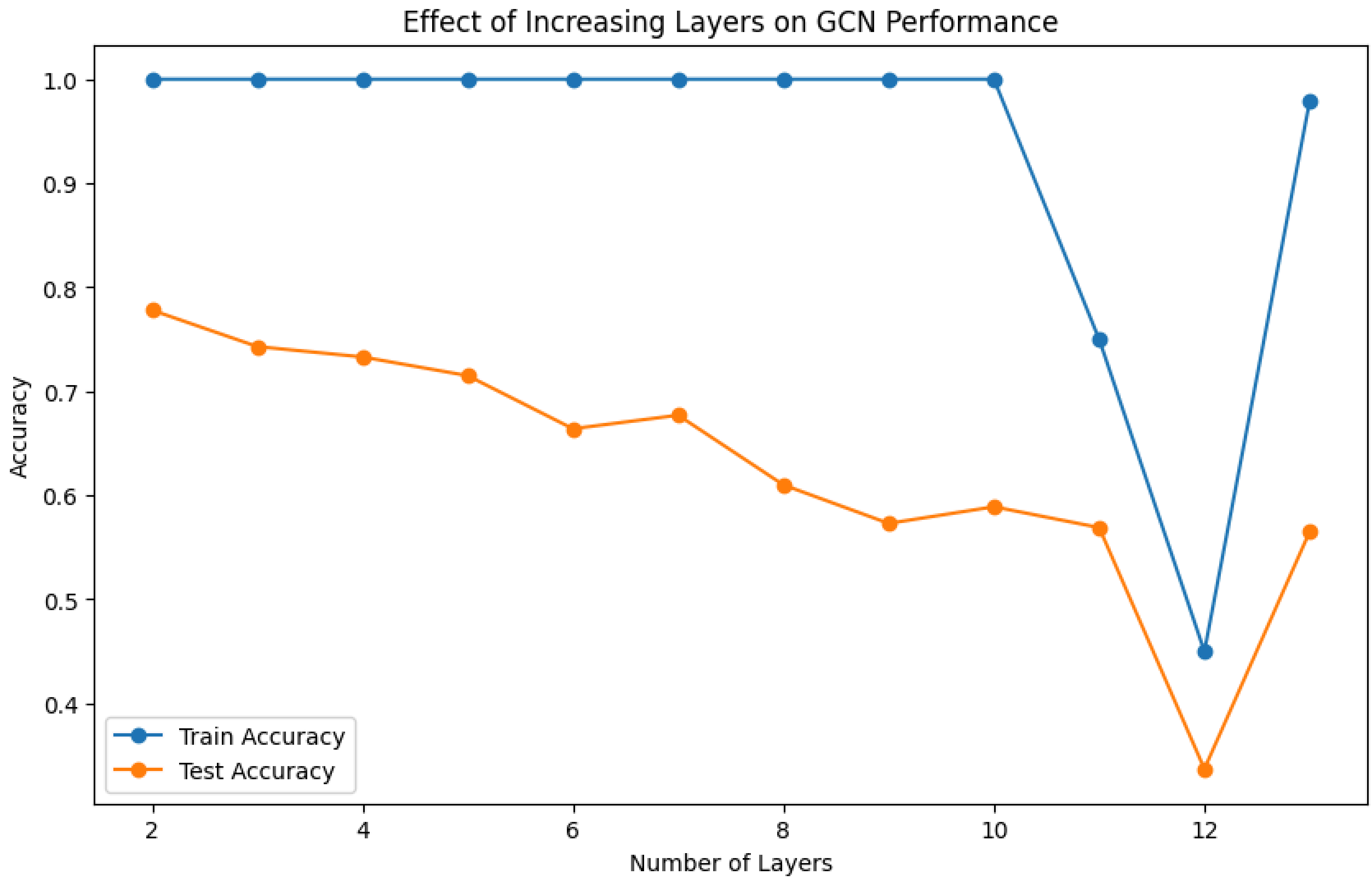
# Task 1

In [5]:
```
import torch
import torch_geometric
import networkx as nx
import matplotlib.pyplot as plt
from torch_geometric.datasets import Planetoid
from torch_geometric.nn import GCNConv, global_add_pool
from torch_geometric.transforms import NormalizeFeatures
from sklearn.manifold import TSNE
from sklearn.metrics import accuracy_score

dataset = Planetoid(root='/tmp/Cora', name='Cora', transform=NormalizeFeatures())

data = dataset[0]
```

In [11]:
```
import numpy as np

class GCN(torch.nn.Module):
    def __init__(self, hidden_channels, num_layers):
        super(GCN, self).__init__()

        self.convs = torch.nn.ModuleList()
        self.convs.append(GCNConv(dataset.num_node_features, hidden_channels))

        for _ in range(num_layers - 2):
            self.convs.append(GCNConv(hidden_channels, hidden_channels))

        self.convs.append(GCNConv(hidden_channels, hidden_channels))

        self.lin = torch.nn.Linear(hidden_channels, dataset.num_classes)

    def forward(self, x, edge_index):
        for conv in self.convs:
            x = conv(x, edge_index).relu()
        return self.lin(x)

def train_and_evaluate(num_layers, hidden_channels=64, epochs=200):
```

```
    model = GCN(hidden_channels=hidden_channels, num_layers=num_layers)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    criterion = torch.nn.CrossEntropyLoss()

    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        out = model(data.x, data.edge_index)
        loss = criterion(out[data.train_mask], data.y[data.train_mask])
        loss.backward()
        optimizer.step()

    model.eval()
    out = model(data.x, data.edge_index)
    pred = out.argmax(dim=1)

    train_acc = accuracy_score(data.y[data.train_mask].cpu(), pred[data.train_mask].cpu())
    test_acc = accuracy_score(data.y[data.test_mask].cpu(), pred[data.test_mask].cpu())
    return train_acc, test_acc

layer_counts = np.arange(2,14)
train_accuracies = []
test_accuracies = []

for num_layers in layer_counts:
    train_acc, test_acc = train_and_evaluate(num_layers=num_layers)
    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)
    print(f'Layers: {num_layers}, Train Accuracy: {train_acc:.4f}, Test Accuracy: {test_acc:.4f}')

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(layer_counts, train_accuracies, label='Train Accuracy', marker='o')
plt.plot(layer_counts, test_accuracies, label='Test Accuracy', marker='o')
plt.xlabel('Number of Layers')
plt.ylabel('Accuracy')
plt.title('Effect of Increasing Layers on GCN Performance')
plt.legend()
plt.show()
```

```
Layers: 2, Train Accuracy: 1.0000, Test Accuracy: 0.7780
Layers: 3, Train Accuracy: 1.0000, Test Accuracy: 0.7430
Layers: 4, Train Accuracy: 1.0000, Test Accuracy: 0.7330
Layers: 5, Train Accuracy: 1.0000, Test Accuracy: 0.7150
Layers: 6, Train Accuracy: 1.0000, Test Accuracy: 0.6640
Layers: 7, Train Accuracy: 1.0000, Test Accuracy: 0.6770
Layers: 8, Train Accuracy: 1.0000, Test Accuracy: 0.6100
Layers: 9, Train Accuracy: 1.0000, Test Accuracy: 0.5730
Layers: 10, Train Accuracy: 1.0000, Test Accuracy: 0.5890
Layers: 11, Train Accuracy: 0.7500, Test Accuracy: 0.5690
Layers: 12, Train Accuracy: 0.4500, Test Accuracy: 0.3370
Layers: 13, Train Accuracy: 0.9786, Test Accuracy: 0.5650
```

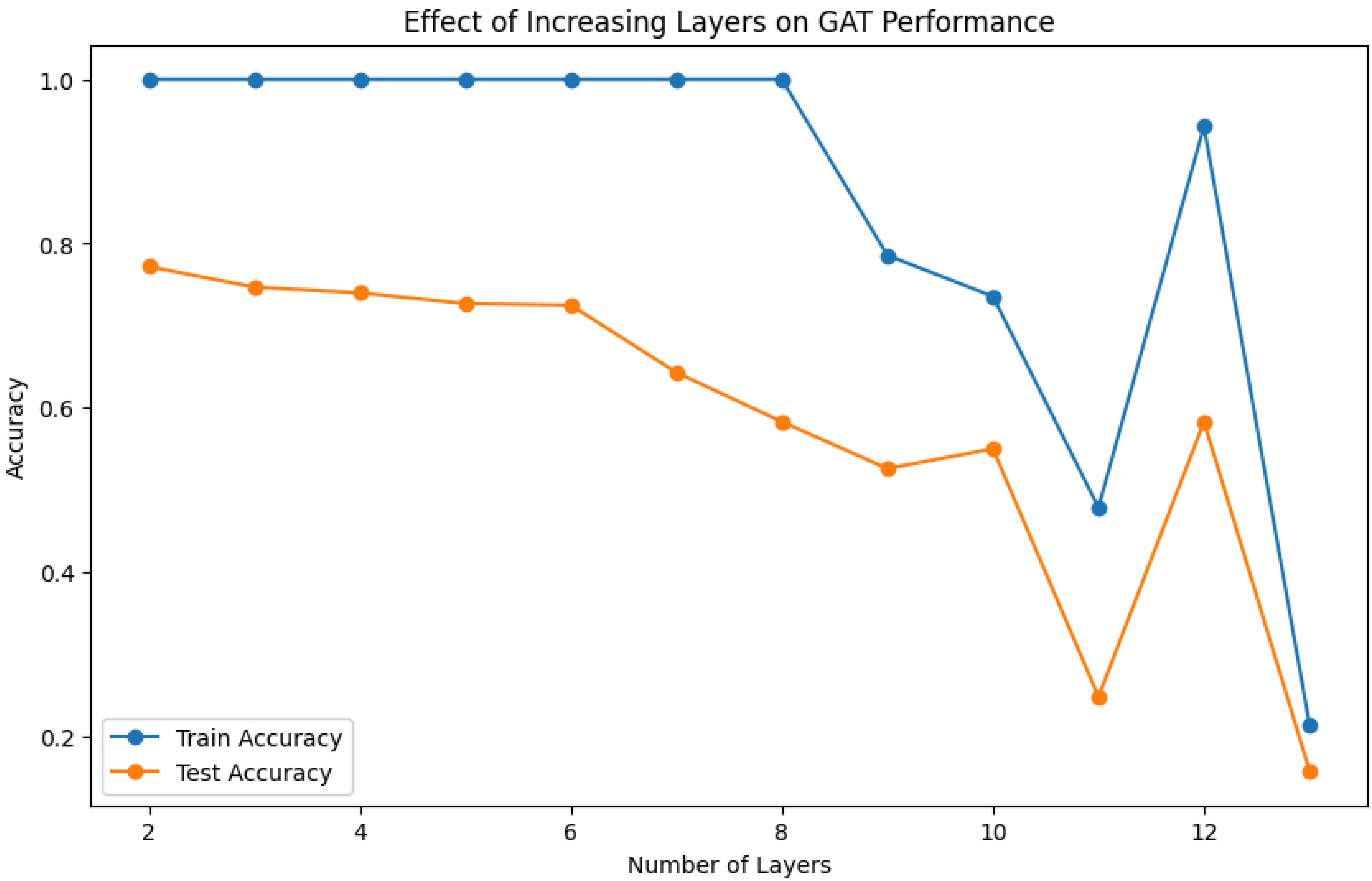



In [12]:
```
from torch_geometric.nn import GATConv

class GAT(torch.nn.Module):
    def __init__(self, hidden_channels, num_layers, heads=4):
        super(GAT, self).__init__()

        self.convs = torch.nn.ModuleList()
        self.convs.append(GATConv(dataset.num_node_features, hidden_channels, heads=heads, concat=True))
```

```
        for _ in range(num_layers - 2):
            self.convs.append(GATConv(hidden_channels * heads, hidden_channels, heads=heads, concat=True))

        self.convs.append(GATConv(hidden_channels * heads, hidden_channels, heads=1, concat=False))

        self.lin = torch.nn.Linear(hidden_channels, dataset.num_classes)

    def forward(self, x, edge_index):
        for conv in self.convs:
            x = conv(x, edge_index).relu()
        return self.lin(x)

def train_and_evaluate(num_layers, hidden_channels=64, heads=4, epochs=200):
    model = GAT(hidden_channels=hidden_channels, num_layers=num_layers, heads=heads)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    criterion = torch.nn.CrossEntropyLoss()

    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        out = model(data.x, data.edge_index)
        loss = criterion(out[data.train_mask], data.y[data.train_mask])
        loss.backward()
        optimizer.step()

    model.eval()
    out = model(data.x, data.edge_index)
    pred = out.argmax(dim=1)

    train_acc = accuracy_score(data.y[data.train_mask].cpu(), pred[data.train_mask].cpu())
    test_acc = accuracy_score(data.y[data.test_mask].cpu(), pred[data.test_mask].cpu())
    return train_acc, test_acc

layer_counts = np.arange(2, 14)
train_accuracies = []
test_accuracies = []

for num_layers in layer_counts:
    train_acc, test_acc = train_and_evaluate(num_layers=num_layers)
    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)
    print(f'Layers: {num_layers}, Train Accuracy: {train_acc:.4f}, Test Accuracy: {test_acc:.4f}')
```

```
plt.figure(figsize=(10, 6))
plt.plot(layer_counts, train_accuracies, label='Train Accuracy', marker='o')
plt.plot(layer_counts, test_accuracies, label='Test Accuracy', marker='o')
plt.xlabel('Number of Layers')
plt.ylabel('Accuracy')
plt.title('Effect of Increasing Layers on GAT Performance')
plt.legend()
plt.show()
```

```
Layers: 2, Train Accuracy: 1.0000, Test Accuracy: 0.7720
Layers: 3, Train Accuracy: 1.0000, Test Accuracy: 0.7470
Layers: 4, Train Accuracy: 1.0000, Test Accuracy: 0.7400
Layers: 5, Train Accuracy: 1.0000, Test Accuracy: 0.7270
Layers: 6, Train Accuracy: 1.0000, Test Accuracy: 0.7250
Layers: 7, Train Accuracy: 1.0000, Test Accuracy: 0.6430
Layers: 8, Train Accuracy: 1.0000, Test Accuracy: 0.5830
Layers: 9, Train Accuracy: 0.7857, Test Accuracy: 0.5260
Layers: 10, Train Accuracy: 0.7357, Test Accuracy: 0.5500
Layers: 11, Train Accuracy: 0.4786, Test Accuracy: 0.2480
Layers: 12, Train Accuracy: 0.9429, Test Accuracy: 0.5830
Layers: 13, Train Accuracy: 0.2143, Test Accuracy: 0.1580
```




There seems to be a overall decrease in accuracy for both models after adding extra layers. I suspect the reason is the aggregation of each node for each layer loses information.

# Task 2

Describe one thing you found interesting in the reading. Describe what it is in your own words and why you found it interesting.

I found the concencationation of k attention (multi attention) as a way to regularization as an interesting way to limit overfitting.

Describe one thing that you found difficult to understand. Try to be specific about what you don't think you understand.

Why do they concatenate instead of average all k elements every layer instead of the last layer in GAT architecture.

In [ ]:

```

```