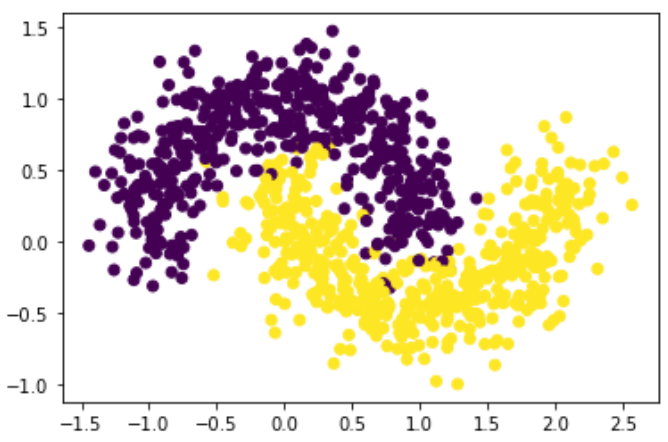
DNN- Experiment 7

- **SIA VASHIST**
- PRN: 20190802107

In [8]:
```
import torch
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

# Define the number of input and output neurons, as well as the size of the hidden layers
n_inputs = 2
n_outputs = 2
n_hidden1 = 64
n_hidden2 = 32
n_hidden3 = 16

# Define the model architecture
model = torch.nn.Sequential(
    torch.nn.Linear(n_inputs, n_hidden1),
    torch.nn.ReLU(),
    torch.nn.Linear(n_hidden1, n_hidden2),
    torch.nn.ReLU(),
    torch.nn.Linear(n_hidden2, n_hidden3),
    torch.nn.ReLU(),
    torch.nn.Linear(n_hidden3, n_outputs)
)
```

In [9]:
```
# Generate the Moon dataset for training and testing
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [10]:
```
# Visualize dataset
plt.scatter(X[:,0], X[:,1], c=y, cmap='viridis')
plt.show()
```



In [11]:
```
# Convert the data to tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)
```

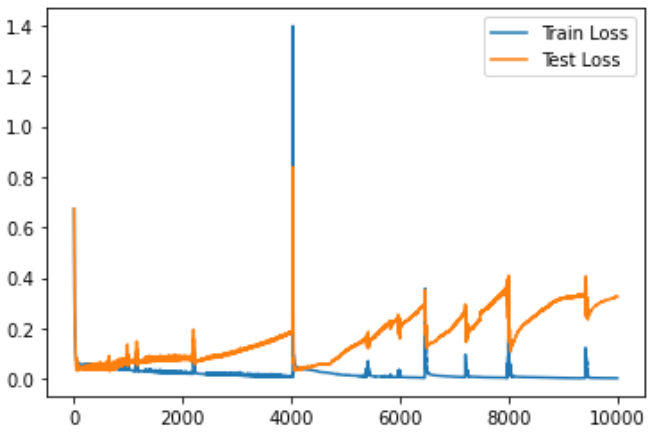
In [12]:
```
# Define the loss function and optimizer
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

In [13]:
```
# Train the model
n_epochs = 10000
train_losses = []
test_losses = []
for epoch in range(n_epochs):
    # Forward pass
    y_pred = model(X_train)
    train_loss = loss_fn(y_pred, y_train)
    test_loss = loss_fn(model(X_test), y_test)
    train_losses.append(train_loss)
    test_losses.append(test_loss)

    # Backward pass
    optimizer.zero_grad()
    train_loss.backward()
    optimizer.step()

    # Print the training and testing loss every 100 epochs
    if epoch % 100 == 0:
        print(f"Epoch {epoch}: Train Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}")
```

```
Epoch 0: Train Loss: 0.6728, Test Loss: 0.6735
Epoch 100: Train Loss: 0.0575, Test Loss: 0.0415
Epoch 200: Train Loss: 0.0566, Test Loss: 0.0492
Epoch 300: Train Loss: 0.0546, Test Loss: 0.0480
Epoch 400: Train Loss: 0.0524, Test Loss: 0.0457
Epoch 500: Train Loss: 0.0501, Test Loss: 0.0411
Epoch 600: Train Loss: 0.0504, Test Loss: 0.0587
Epoch 700: Train Loss: 0.0439, Test Loss: 0.0447
Epoch 800: Train Loss: 0.0396, Test Loss: 0.0572
Epoch 900: Train Loss: 0.0366, Test Loss: 0.0586
Epoch 1000: Train Loss: 0.0393, Test Loss: 0.0725
Epoch 1100: Train Loss: 0.0309, Test Loss: 0.0717
Epoch 1200: Train Loss: 0.0304, Test Loss: 0.0651
Epoch 1300: Train Loss: 0.0314, Test Loss: 0.0795
Epoch 1400: Train Loss: 0.0281, Test Loss: 0.0793
Epoch 1500: Train Loss: 0.0304, Test Loss: 0.0703
Epoch 1600: Train Loss: 0.0283, Test Loss: 0.0713
Epoch 1700: Train Loss: 0.0240, Test Loss: 0.0739
Epoch 1800: Train Loss: 0.0226, Test Loss: 0.0751
Epoch 1900: Train Loss: 0.0248, Test Loss: 0.0892
Epoch 2000: Train Loss: 0.0198, Test Loss: 0.0801
Epoch 2100: Train Loss: 0.0196, Test Loss: 0.0835
Epoch 2200: Train Loss: 0.0600, Test Loss: 0.0845
Epoch 2300: Train Loss: 0.0195, Test Loss: 0.0732
Epoch 2400: Train Loss: 0.0180, Test Loss: 0.0776
Epoch 2500: Train Loss: 0.0168, Test Loss: 0.0791
Epoch 2600: Train Loss: 0.0157, Test Loss: 0.0858
Epoch 2700: Train Loss: 0.0219, Test Loss: 0.0915
Epoch 2800: Train Loss: 0.0206, Test Loss: 0.0984
Epoch 2900: Train Loss: 0.0169, Test Loss: 0.1019
Epoch 3000: Train Loss: 0.0197, Test Loss: 0.1103
Epoch 3100: Train Loss: 0.0117, Test Loss: 0.1106
Epoch 3200: Train Loss: 0.0137, Test Loss: 0.1202
Epoch 3300: Train Loss: 0.0105, Test Loss: 0.1244
Epoch 3400: Train Loss: 0.0143, Test Loss: 0.1331
Epoch 3500: Train Loss: 0.0116, Test Loss: 0.1396
Epoch 3600: Train Loss: 0.0107, Test Loss: 0.1483
Epoch 3700: Train Loss: 0.0117, Test Loss: 0.1542
Epoch 3800: Train Loss: 0.0071, Test Loss: 0.1661
Epoch 3900: Train Loss: 0.0065, Test Loss: 0.1703
Epoch 4000: Train Loss: 0.0063, Test Loss: 0.1829
Epoch 4100: Train Loss: 0.0454, Test Loss: 0.0344
Epoch 4200: Train Loss: 0.0390, Test Loss: 0.0379
Epoch 4300: Train Loss: 0.0356, Test Loss: 0.0474
Epoch 4400: Train Loss: 0.0328, Test Loss: 0.0512
Epoch 4500: Train Loss: 0.0269, Test Loss: 0.0564
Epoch 4600: Train Loss: 0.0246, Test Loss: 0.0565
Epoch 4700: Train Loss: 0.0222, Test Loss: 0.0575
Epoch 4800: Train Loss: 0.0182, Test Loss: 0.0757
Epoch 4900: Train Loss: 0.0155, Test Loss: 0.0926
Epoch 5000: Train Loss: 0.0134, Test Loss: 0.1076
Epoch 5100: Train Loss: 0.0118, Test Loss: 0.1227
Epoch 5200: Train Loss: 0.0113, Test Loss: 0.1358
Epoch 5300: Train Loss: 0.0097, Test Loss: 0.1444
Epoch 5400: Train Loss: 0.0196, Test Loss: 0.1237
Epoch 5500: Train Loss: 0.0098, Test Loss: 0.1604
Epoch 5600: Train Loss: 0.0081, Test Loss: 0.1818
Epoch 5700: Train Loss: 0.0069, Test Loss: 0.1998
Epoch 5800: Train Loss: 0.0067, Test Loss: 0.2215
Epoch 5900: Train Loss: 0.0057, Test Loss: 0.2255
Epoch 6000: Train Loss: 0.0142, Test Loss: 0.2050
Epoch 6100: Train Loss: 0.0051, Test Loss: 0.2256
Epoch 6200: Train Loss: 0.0045, Test Loss: 0.2450
Epoch 6300: Train Loss: 0.0041, Test Loss: 0.2597
Epoch 6400: Train Loss: 0.0032, Test Loss: 0.2843
Epoch 6500: Train Loss: 0.0282, Test Loss: 0.1414
Epoch 6600: Train Loss: 0.0137, Test Loss: 0.1416
Epoch 6700: Train Loss: 0.0114, Test Loss: 0.1603
Epoch 6800: Train Loss: 0.0100, Test Loss: 0.1766
Epoch 6900: Train Loss: 0.0083, Test Loss: 0.1889
Epoch 7000: Train Loss: 0.0073, Test Loss: 0.2124
Epoch 7100: Train Loss: 0.0066, Test Loss: 0.2377
Epoch 7200: Train Loss: 0.0309, Test Loss: 0.2493
Epoch 7300: Train Loss: 0.0077, Test Loss: 0.1981
Epoch 7400: Train Loss: 0.0065, Test Loss: 0.2186
Epoch 7500: Train Loss: 0.0053, Test Loss: 0.2601
Epoch 7600: Train Loss: 0.0047, Test Loss: 0.2791
Epoch 7700: Train Loss: 0.0041, Test Loss: 0.2979
Epoch 7800: Train Loss: 0.0036, Test Loss: 0.3245
Epoch 7900: Train Loss: 0.0031, Test Loss: 0.3361
Epoch 8000: Train Loss: 0.0684, Test Loss: 0.1989
Epoch 8100: Train Loss: 0.0080, Test Loss: 0.1506
Epoch 8200: Train Loss: 0.0065, Test Loss: 0.1944
Epoch 8300: Train Loss: 0.0057, Test Loss: 0.2168
Epoch 8400: Train Loss: 0.0053, Test Loss: 0.2361
Epoch 8500: Train Loss: 0.0049, Test Loss: 0.2508
Epoch 8600: Train Loss: 0.0044, Test Loss: 0.2669
Epoch 8700: Train Loss: 0.0039, Test Loss: 0.2832
Epoch 8800: Train Loss: 0.0033, Test Loss: 0.2998
Epoch 8900: Train Loss: 0.0029, Test Loss: 0.3114
Epoch 9000: Train Loss: 0.0024, Test Loss: 0.3181
Epoch 9100: Train Loss: 0.0020, Test Loss: 0.3238
Epoch 9200: Train Loss: 0.0017, Test Loss: 0.3221
Epoch 9300: Train Loss: 0.0013, Test Loss: 0.3256
Epoch 9400: Train Loss: 0.0011, Test Loss: 0.3365
Epoch 9500: Train Loss: 0.0029, Test Loss: 0.2632
Epoch 9600: Train Loss: 0.0019, Test Loss: 0.2903
Epoch 9700: Train Loss: 0.0016, Test Loss: 0.3031
Epoch 9800: Train Loss: 0.0015, Test Loss: 0.3114
Epoch 9900: Train Loss: 0.0013, Test Loss: 0.3185
```

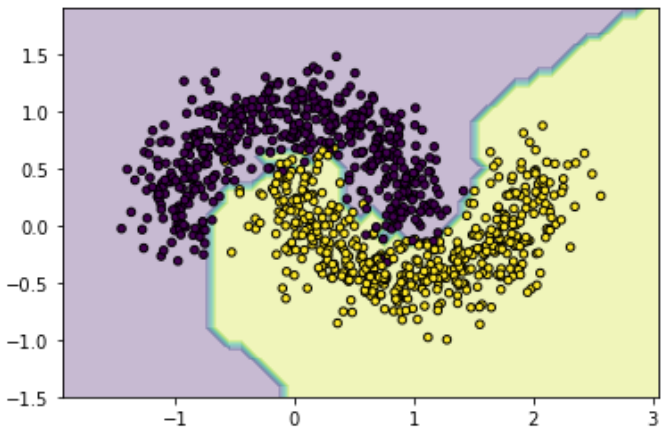
In [14]:

```python
# Plot the training and testing loss
plt.plot([i.detach().numpy() for i in train_losses], label='Train Loss')
plt.plot([i.detach().numpy() for i in test_losses], label='Test Loss')
plt.legend()
plt.show()
```



In [15]:
```
# Visualize the decision boundary
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = torch.meshgrid(torch.arange(x_min, x_max, 0.1), torch.arange(y_min, y_max, 0.1))
Z = model(torch.tensor(torch.stack([xx.ravel(), yy.ravel()], axis=1), dtype=torch.float32))
Z = torch.argmax(Z, dim=1)
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors='k')
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_8328\3957665579.py:5: UserWarning: To copy construct from a tensor, it is recommended to use so
urceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  Z = model(torch.tensor(torch.stack([xx.ravel(), yy.ravel()], axis=1), dtype=torch.float32))
```



Conclusion:

This defines a neural network with three hidden layers, using ReLU activation functions, which is trained on the "moons" dataset using cross-entropy loss and the Adam optimizer. The training and testing loss are plotted to assess the performance of the model. Finally, the decision boundary is visualized using a contour plot overlaid on the original dataset. This is a great example of a simple classification problem and how a neural network can be trained to solve it.