1 Dataset Preparation

Organize your dataset in the following structure: dataset/ train/ class_1/ image1.jpg image2.jpg class_2/ image3.jpg image4.jpg

2 Load the Dataset

Use torchvision.datasets.ImageFolder to load the training and testing datasets. Apply the following transformations: Resize the images to 28 × 28 28×28. Convert the images to PyTorch tensors. Normalize the images with mean (0.5,0.5,0.5)and standard deviation (0.5,0.5,0.5).

3.Create DataLoaders Batch size: 4 Shuffle: True for training and False for testing.

4.Visualize a Batch

Write a function imshow to display a batch of images. Use torchvision.utils.make_grid to create a grid of images.

In [1]:
```
import torch
import os
import torchvision
import torchvision.transforms as transforms
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
# 1. Organize your dataset
# Assume the dataset is organized in 'dataset/train' and 'dataset/test' director

# 2. Load Dataset
# Define the transformations (ToTensor and Normalize)
# BEGIN SOLUTION
transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize with
    ])
# END SOLUTION
```

In [4]:
```
# Path to your dataset folder
dataset_dir = 'data/shapes'
```

In [6]:
```
# Set the path for your dataset
train_dir = './data/shapes/'
#test_dir = './dataset/test'
```

In [8]:
```
# Use ImageFolder to load the dataset from the directory
#root=train_dir: Specifies the directory containing training images.
trainset = torchvision.datasets.ImageFolder(root=train_dir, transform=transform)
#testset = torchvision.datasets.ImageFolder(root=test_dir, transform=transform)
```

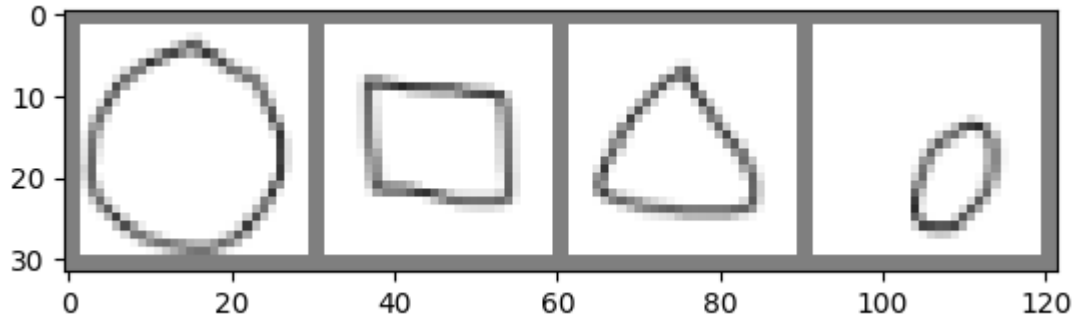
```
dataset/
    ├── train/
    │   ├── class_0/   # Folder for class 0 images
    │   │   ├── img_1.jpg
    │   │   ├── img_2.jpg
    │   ├── class_1/   # Folder for class 1 images
    │   │   ├── img_3.jpg
    │   │   ├── img_4.jpg
    ├── test/
        ├── class_0/
        ├── class_1/
```

In [10]:
```
# 3. Create Train and Test DataLoader
# BEGIN SOLUTION
batch_size = 4
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuff
# END SOLUTION
```

In [12]:
```
# 4. Display Sample Data
# Function to display an image
def imshow(img):

    img = img / 2 + 0.5  # Unnormalize the image (since we normalized it with me
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))  # Transpose to (H, W, C) for dis
    plt.show()
#is used to create an iterator from the trainloader,
# which is a PyTorch DataLoader.
dataiter = iter(trainloader) #Converts trainloader into an iterator.
images, labels = next(dataiter)# Retrieves the next batch of images and labels.
```

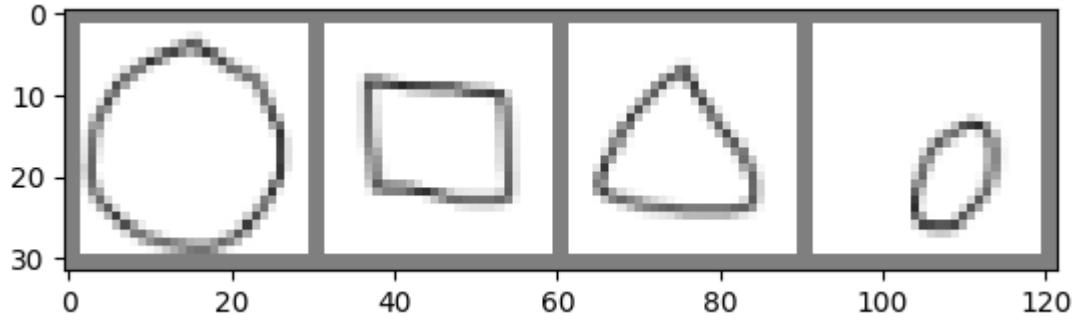
In [13]:
```
# Assertions for data batch
assert isinstance(images, torch.Tensor), "Images should be a PyTorch tensor."
assert isinstance(labels, torch.Tensor), "Labels should be a PyTorch tensor."
assert images.ndim == 4, "Images should have 4 dimensions (batch_size, channels,
assert labels.ndim == 1, "Labels should have 1 dimension (batch_size)."
assert images.shape[0] == labels.shape[0], "Number of images should match the nu

# Display images with labels
print("Labels: ", labels)  # Labels correspond to class indices
imshow(torchvision.utils.make_grid(images))
```

```
Labels:  tensor([0, 1, 2, 0])
```



In [14]:
```
# Get a batch of training data
# Display images
print("Labels: ", labels)
#BEGIN SOLUTION
#arranges multiple images into a single image grid.
# imshow() displays the images after unnormalization.
imshow(torchvision.utils.make_grid(images))
#END SOLUTION
```

```
Labels:  tensor([0, 1, 2, 0])
```



In [16]:
```
# 5. Device Check (optional)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
[NVSHARE][WARN]: Couldn't open file /var/run/secrets/kubernetes.io/serviceaccou
nt/namespace to read Pod namespace
[NVSHARE][INFO]: Successfully initialized nvshare GPU
[NVSHARE][INFO]: Client ID = 461b12af2a865081
```

In [17]:
```
# Print Dataset Statistics
print(f"Total images in the dataset: {len(trainset)}")
print(f"Number of classes: {len(trainset.classes)}")
print(f"Classes: {trainset.classes}")
```

```
Total images in the dataset: 301
Number of classes: 3
Classes: ['circles', 'squares', 'triangles']
```