Solve a real world problem using CBM.

The MNIST dataset of handwritten digits. We can simulate the noise by adding random Gaussian noise to the clean training and testing images.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt

# ----------------------------
# 1. Load MNIST dataset
# ----------------------------
transform = transforms.Compose([transforms.ToTensor()])
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

# Convert images to tensors in [0,1]
train_images = train_data.data.float() / 255.0
test_images = test_data.data.float() / 255.0

# ----------------------------
# 2. Add Gaussian noise
# ----------------------------
def add_gaussian_noise(images, mean=0.0, std=0.3):
    noise = torch.randn_like(images) * std + mean
    noisy = torch.clamp(images + noise, 0., 1.)
    return noisy

noisy_train = add_gaussian_noise(train_images)
```

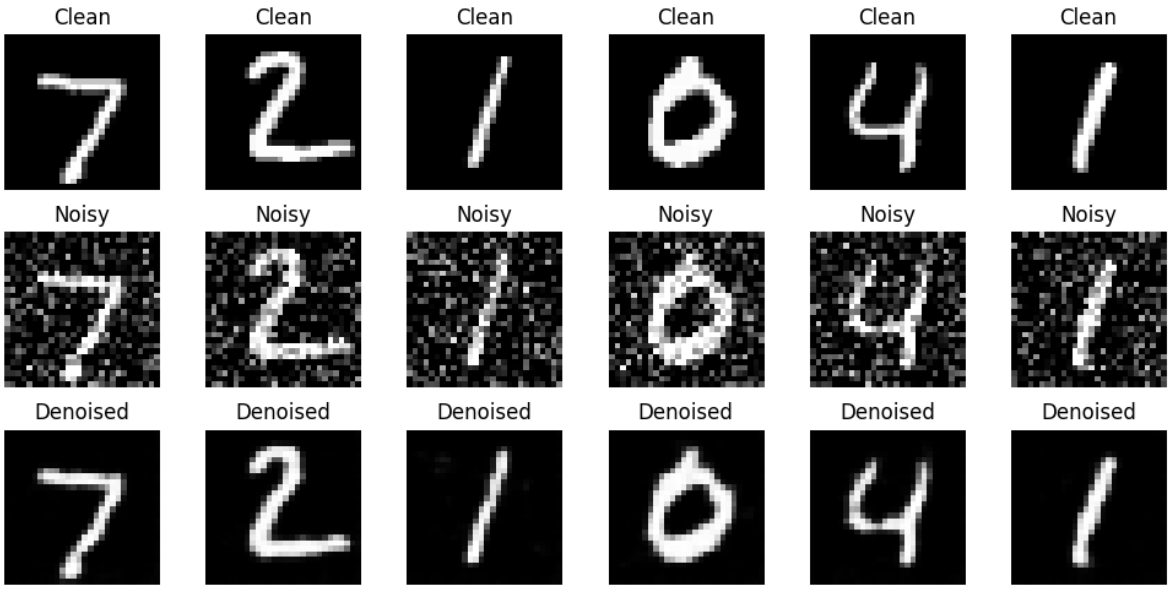
```
noisy_test = add_gaussian_noise(test_images)

# Wrap into dataset
train_dataset = TensorDataset(noisy_train.unsqueeze(1), train_images.unsqueeze(1))
test_dataset = TensorDataset(noisy_test.unsqueeze(1), test_images.unsqueeze(1))

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

# ------------------------------
# 3. Define a Convolutional Boltzmann-like Autoencoder
# ------------------------------
class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()
        # Encoder (feature learning, similar to CBM inference)
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=2, padding=1),  # 28x28 -> 14x14
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1), # 14x14 -> 7x7
            nn.ReLU(),
        )
        # Decoder (reconstruction, like visible layer reconstruction)
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(32, 16, 3, stride=2, output_padding=1, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 3, stride=2, output_padding=1, padding=1),
            nn.Sigmoid()  # keep output in [0,1]
        )

    def forward(self, x):
```

```
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

model = ConvAutoencoder()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# ----------------------------
# 4. Train the model
# ----------------------------
num_epochs = 5
for epoch in range(num_epochs):
    total_loss = 0
    for noisy_imgs, clean_imgs in train_loader:
        output = model(noisy_imgs)
        loss = criterion(output, clean_imgs)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss/len(train_loader):.4f}")

# ----------------------------
# 5. Visualize results
# ----------------------------
model.eval()
with torch.no_grad():
    noisy_imgs, clean_imgs = next(iter(test_loader))
    reconstructed = model(noisy_imgs)
```

```
# Plot examples
fig, axes = plt.subplots(3, 6, figsize=(10, 5))
for i in range(6):
    # Original
    axes[0, i].imshow(clean_imgs[i].squeeze(), cmap='gray')
    axes[0, i].set_title("Clean")
    axes[0, i].axis('off')

    # Noisy
    axes[1, i].imshow(noisy_imgs[i].squeeze(), cmap='gray')
    axes[1, i].set_title("Noisy")
    axes[1, i].axis('off')

    # Denoised
    axes[2, i].imshow(reconstructed[i].squeeze(), cmap='gray')
    axes[2, i].set_title("Denoised")
    axes[2, i].axis('off')

plt.tight_layout()
plt.show()
```

output:

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 37.8MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 1.04MB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 9.26MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 6.73MB/s]
Epoch [1/5], Loss: 0.0486
Epoch [2/5], Loss: 0.0068
Epoch [3/5], Loss: 0.0061
Epoch [4/5], Loss: 0.0059
Epoch [5/5], Loss: 0.0058
```


Clean
Clean
Clean
Clean
Clean
Clean
Noisy
Noisy
Noisy
Noisy
Noisy
Noisy
Denoised
Denoised
Denoised
Denoised
Denoised
Denoised