In [ ]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Używane urządzenie:", device)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, down
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, down

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)
```

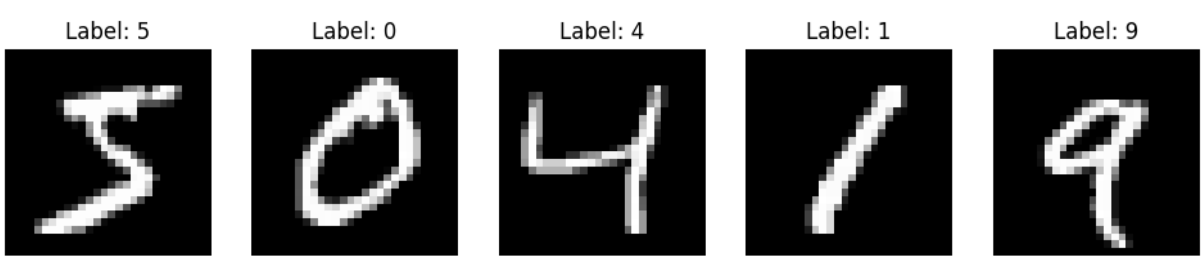
```
Używane urządzenie: cpu
```

```
100%|██████████| 9.91M/9.91M [00:05<00:00, 1.69MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 145kB/s]
100%|██████████| 1.65M/1.65M [00:01<00:00, 1.39MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 4.56MB/s]
```

Każdy przykład to cyfra od 0 do 9.

In [43]:

```python
fig, axes = plt.subplots(1, 5, figsize=(12, 3))
for i, ax in enumerate(axes):
    image, label = train_dataset[i]
    ax.imshow(image.squeeze(), cmap='gray')
    ax.set_title(f"Label: {label}")
    ax.axis('off')
plt.show()
```



In [44]:

```python
class SimpleMLP(nn.Module):
    def __init__(self):
        super(SimpleMLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
```

```
        return x
model_mlp = SimpleMLP().to(device)
optimizer = optim.Adam(model_mlp.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

To jest sieć, która bierze obrazek, zamienia w wektor, przepuszcza przez warstwy i mówi, jaka to cyfra. Potem ustawiamy optymalizator i funkcję strat, żeby ją trenować.

In [45]:
```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.dropout2(x)
        x = self.fc2(x)
        return x

model_cnn = SimpleCNN().to(device)
optimizer_cnn = optim.Adam(model_cnn.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

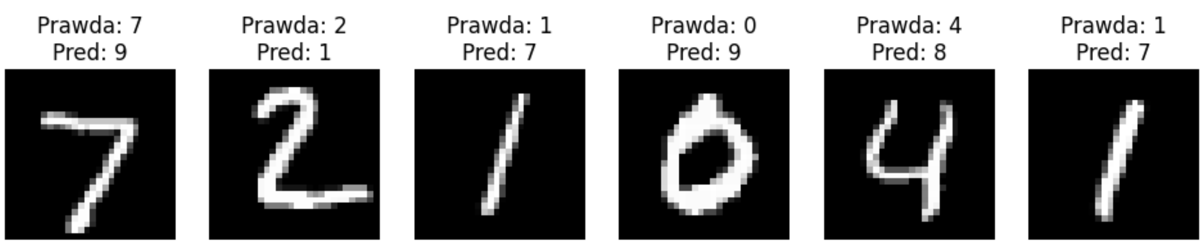
sieć CNN, która patrzy na obrazek jak na małą mapę, wyłapuje wzory przez te warstwy conv, trochę losowo wyłącza część neuronów, potem spłaszcza wszystko i przepuszcza przez zwykłe warstwy, żeby zgadnąć cyfrę.

In [46]:
```
def train(model, loader, optimizer, criterion, epochs=1):
    model.train()
    for epoch in range(epochs):
        total_loss = 0
        for data, target in loader:
            data, target = data.to(device), target.to(device)
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f"Epoch {epoch+1}, Loss: {total_loss/len(loader)}")

def test(model, loader, criterion):
    model.eval()
    correct = 0
```

```python
    test_loss = 0
    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    acc = 100. * correct / len(loader.dataset)
    print(f"Test Accuracy: {acc:.2f}%")
    return acc
```

In [47]:

```python
examples = enumerate(test_loader)
batch_idx, (example_data, example_targets) = next(examples)

with torch.no_grad():
    output = model_cnn(example_data.to(device))
fig, axes = plt.subplots(1, 6, figsize=(12, 3))
for i in range(6):
    ax = axes[i]
    ax.imshow(example_data[i][0], cmap='gray')
    pred = output.argmax(dim=1, keepdim=True)[i].item()
    ax.set_title(f"Prawda: {example_targets[i]}\nPred: {pred}")
    ax.axis('off')
plt.show()
```

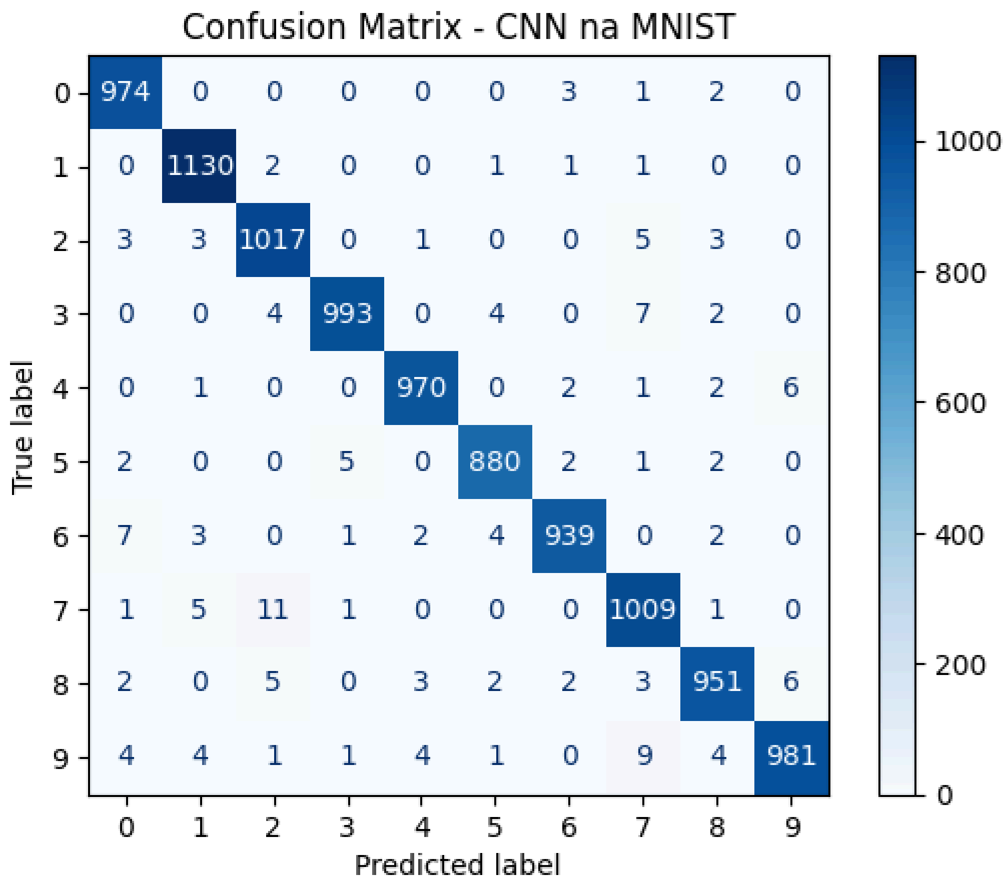


model najczęściej myli cyfry podobne wizualnie

In [49]:

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
y_true = []
y_pred = []
model_cnn.eval()
with torch.no_grad():
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model_cnn(data)
        preds = output.argmax(dim=1, keepdim=True).cpu().numpy()
        y_pred.extend(preds.flatten())
        y_true.extend(target.cpu().numpy())
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=list(range(10)))
plt.figure(figsize=(8,8))
disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix - CNN na MNIST")
plt.show()
```

```
<Figure size 800x800 with 0 Axes>
```



In [50]:

```
train(model_mlp, train_loader, optimizer, criterion, epochs=3)
acc_mlp = test(model_mlp, test_loader, criterion)
train(model_cnn, train_loader, optimizer_cnn, criterion, epochs=3)
acc_cnn = test(model_cnn, test_loader, criterion)
print("Porównanie:")
print(f"MLP Accuracy: {acc_mlp:.2f}%")
print(f"CNN Accuracy: {acc_cnn:.2f}%")
```

```
Epoch 1, Loss: 0.10733401530912753
Epoch 2, Loss: 0.09314091067199212
Epoch 3, Loss: 0.08228573606941285
Test Accuracy: 97.30%
Epoch 1, Loss: 0.096167398086473
Epoch 2, Loss: 0.07364081090781838
Epoch 3, Loss: 0.06064090667206095
Test Accuracy: 99.09%
Porównanie:
MLP Accuracy: 97.30%
CNN Accuracy: 99.09%
```

Podsumowanie wyników

- MLP (baseline) uzyskał 97.3% dokładności, co potwierdza, że nawet prosta sieć potrafi nauczyć się rozpoznawania cyfr.
- CNN osiągnął 99.1% dokładności, czyli znacznie lepszy wynik – sieć konwolucyjna lepiej uchwyciła cechy obrazów.
- Najwięcej pomyłek dotyczy cyfr podobnych wizualnie (np. 1 i 7, 4 i 9).