

Image Classification Project

Dataset Acquisition, Splitting, Data Augmentation, Training, and Testing

```
In [1]: import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms as T

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    classification_report, confusion_matrix, ConfusionMatrixDisplay
)
```

```
In [2]: SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.set_num_threads(2)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Device:', device)
```

Device: cpu

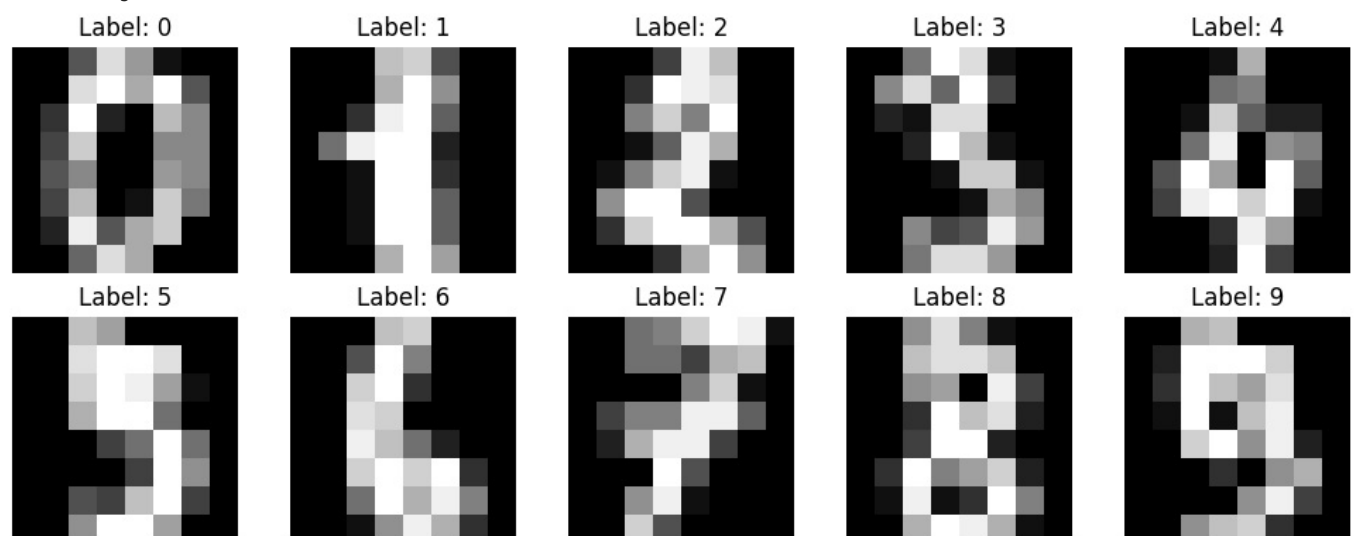
1. Acquire and inspect the image dataset

```
In [3]: digits = load_digits()
X = digits.images.astype(np.float32)
y = digits.target.astype(np.int64)
class_names = [str(i) for i in digits.target_names]

print('Images shape:', X.shape)
print('Labels shape:', y.shape)
print('Classes:', class_names)
print('Pixel range:', X.min(), 'to', X.max())

fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i, ax in enumerate(axes.flat):
    ax.imshow(X[i], cmap='gray')
    ax.set_title(f'Label: {y[i]}')
    ax.axis('off')
plt.tight_layout()
plt.show()
```

Images shape: (1797, 8, 8)
Labels shape: (1797,)
Classes: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
Pixel range: 0.0 to 16.0



2. Split the dataset

```
In [4]: # 70% training, 15% validation, 15% testing, with stratification.
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, random_state=SEED, stratify=y
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, random_state=SEED, stratify=y_temp
)

split_summary = pd.DataFrame({
    'Split': ['Training', 'Validation', 'Testing'],
    'Images': [len(X_train), len(X_val), len(X_test)],
    'Percentage': [len(X_train)/len(X)*100, len(X_val)/len(X)*100, len(X_test)/len(X)*100]
})
display(split_summary)
```

	Split	Images	Percentage
0	Training	1257	69.949917
1	Validation	270	15.025042
2	Testing	270	15.025042

3. Apply data augmentation

```
In [5]: class DigitsDataset(Dataset):
    def __init__(self, images, labels, transform=None):
        self.images = images
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        # Scale 0-16 grayscale values to 0-255 uint8 for PIL-compatible transforms.
        image = (self.images[idx] / 16.0 * 255.0).astype(np.uint8)
        image = np.expand_dims(image, axis=-1)
        if self.transform:
            image = self.transform(image)
        return image, int(self.labels[idx])

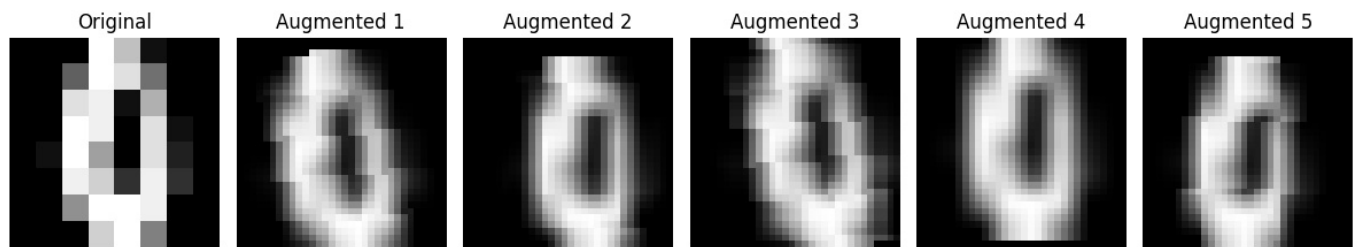
train_transform = T.Compose([
    T.ToPILImage(),
    T.Resize((32, 32)),
    T.RandomRotation(12),
    T.RandomAffine(degrees=0, translate=(0.08, 0.08), scale=(0.90, 1.10)),
    T.ToTensor(),
    T.Normalize(mean=[0.5], std=[0.5])
])

eval_transform = T.Compose([
    T.ToPILImage(),
    T.Resize((32, 32)),
    T.ToTensor(),
    T.Normalize(mean=[0.5], std=[0.5])
])

train_dataset = DigitsDataset(X_train, y_train, train_transform)
val_dataset = DigitsDataset(X_val, y_val, eval_transform)
test_dataset = DigitsDataset(X_test, y_test, eval_transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=0)

# Show one original image and several randomly augmented versions.
fig, axes = plt.subplots(1, 6, figsize=(12, 2.5))
axes[0].imshow(X_train[0], cmap='gray')
axes[0].set_title('Original')
axes[0].axis('off')
for i in range(1, 6):
    aug, label = train_dataset[0]
    axes[i].imshow(aug.squeeze().numpy(), cmap='gray')
    axes[i].set_title(f'Augmented {i}')
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```



4. Define two CNN models

```
In [6]: class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 64),
            nn.ReLU(),
            nn.Dropout(0.30),
            nn.Linear(64, num_classes)
        )

    def forward(self, x):
        return self.classifier(self.features(x))

class DeeperCNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 16, 3, padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1, 1))
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Dropout(0.35),
            nn.Linear(64, num_classes)
        )

    def forward(self, x):
        return self.classifier(self.features(x))

print(SimpleCNN())
print(DeeperCNN())
```

```

SimpleCNN(
    (features): Sequential(
      (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
      (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): ReLU()
      (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
      (0): Flatten(start_dim=1, end_dim=-1)
      (1): Linear(in_features=2048, out_features=64, bias=True)
      (2): ReLU()
      (3): Dropout(p=0.3, inplace=False)
      (4): Linear(in_features=64, out_features=10, bias=True)
    )
)

DeeperCNN(
    (features): Sequential(
      (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, bias=True, track_running_stats=True)
      (2): ReLU()
      (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (4): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, bias=True, track_running_stats=True)
      (6): ReLU()
      (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (8): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (9): ReLU()
      (10): AdaptiveAvgPool2d(output_size=(1, 1))
    )
    (classifier): Sequential(
      (0): Flatten(start_dim=1, end_dim=-1)
      (1): Dropout(p=0.35, inplace=False)
      (2): Linear(in_features=64, out_features=10, bias=True)
    )
)

```

5. Train the models

```

In [7]: def run_epoch(model, loader, criterion, optimizer=None):
    training = optimizer is not None
    model.train() if training else model.eval()

    total_loss, correct, total = 0.0, 0, 0
    with torch.set_grad_enabled(training):
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            if training:
                optimizer.zero_grad()
                outputs = model(images)
                loss = criterion(outputs, labels)
                if training:
                    loss.backward()
                    optimizer.step()

            total_loss += loss.item() * images.size(0)
            correct += (outputs.argmax(1) == labels.sum()).item()
            total += images.size(0)

    return total_loss / total, correct / total

def train_model(model, epochs=2):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.5)

    history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}
    for epoch in range(epochs):
        train_loss, train_acc = run_epoch(model, train_loader, criterion, optimizer)
        val_loss, val_acc = run_epoch(model, val_loader, criterion)
        scheduler.step()

        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['val_loss'].append(val_loss)
        history['val_acc'].append(val_acc)

    print(f'Epoch {epoch+1:02d}/{epochs} | '
          f'train loss={train_loss:.4f}, train acc={train_acc:.4f} | ')

```

```

        f'val loss={val_loss:.4f}, val acc={val_acc:.4f}')

    return model, history

```

```

In [8]: models_to_train = {
        'SimpleCNN': SimpleCNN(),
        'DeeperCNN': DeeperCNN()
    }

    trained_models = {}
    histories = {}

    for name, model in models_to_train.items():
        print()
        print('=' * 70)
        print('Training', name)
        trained_models[name], histories[name] = train_model(model, epochs=2)

=====
Training SimpleCNN
Epoch 01/2 | train loss=2.2452, train acc=0.1695 | val loss=2.0052, val acc=0.4259
Epoch 02/2 | train loss=1.7440, train acc=0.4272 | val loss=1.0179, val acc=0.8074

=====
Training DeeperCNN
Epoch 01/2 | train loss=2.3052, train acc=0.1138 | val loss=2.2900, val acc=0.1889
Epoch 02/2 | train loss=2.2424, train acc=0.2021 | val loss=2.2265, val acc=0.1815

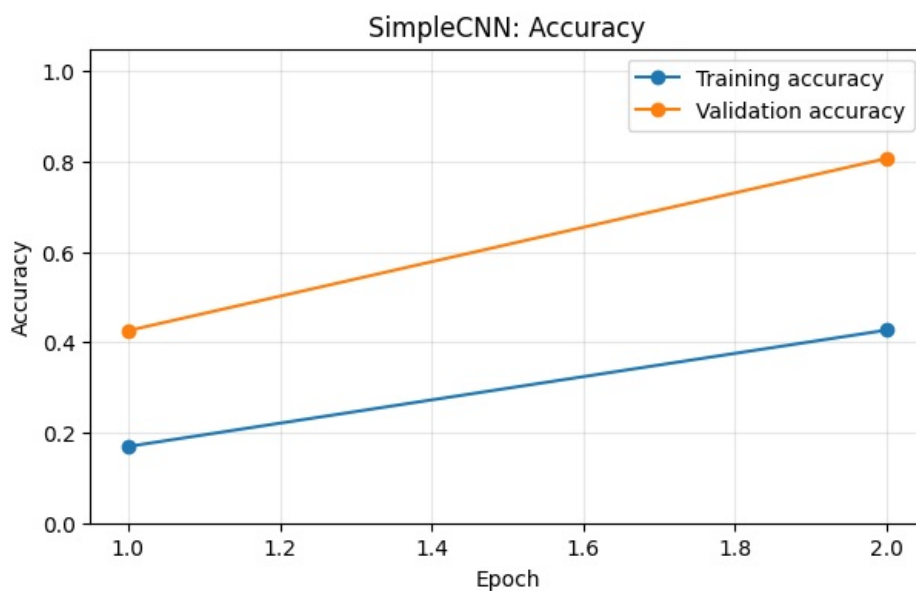
```

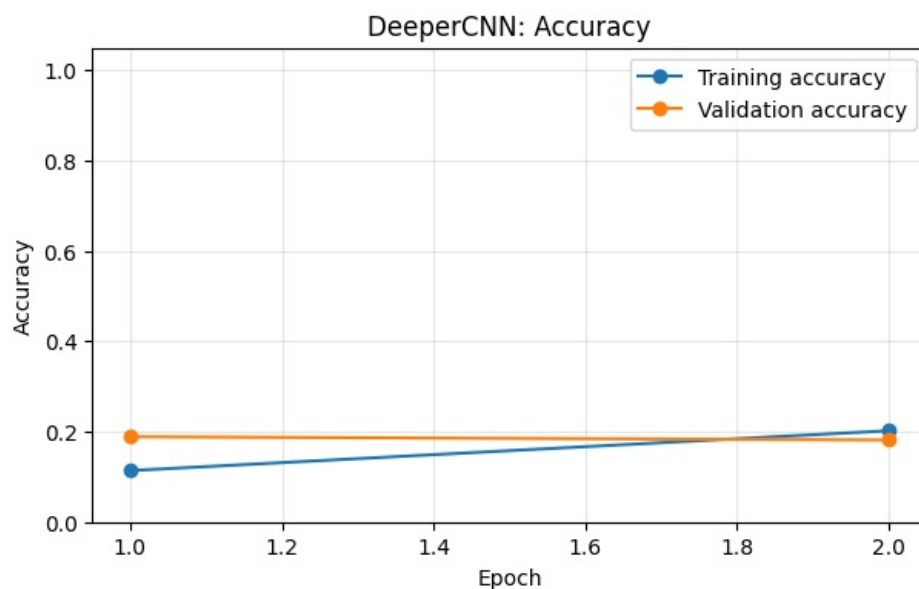
6. Plot training histories

```

In [9]: for name, history in histories.items():
        epochs = range(1, len(history['train_acc']) + 1)
        plt.figure(figsize=(7, 4))
        plt.plot(epochs, history['train_acc'], marker='o', label='Training accuracy')
        plt.plot(epochs, history['val_acc'], marker='o', label='Validation accuracy')
        plt.title(f'{name}: Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.ylim(0, 1.05)
        plt.grid(True, alpha=0.3)
        plt.legend()
        plt.show()

```





7. Test and compare the models

```
In [10]: def predict_model(model, loader):
    model.eval()
    y_true, y_pred = [], []
    with torch.no_grad():
        for images, labels in loader:
            outputs = model(images.to(device))
            preds = outputs.argmax(1).cpu().numpy()
            y_true.extend(labels.numpy())
            y_pred.extend(preds)
    return np.array(y_true), np.array(y_pred)

results = []
all_predictions = {}

for name, model in trained_models.items():
    y_true, y_pred = predict_model(model, test_loader)
    all_predictions[name] = (y_true, y_pred)
    results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_true, y_pred),
        'Precision (weighted)': precision_score(y_true, y_pred, average='weighted', zero_division=0),
        'Recall (weighted)': recall_score(y_true, y_pred, average='weighted', zero_division=0),
        'F1-score (weighted)': f1_score(y_true, y_pred, average='weighted', zero_division=0)
    })

results_df = pd.DataFrame(results).sort_values('Accuracy', ascending=False).reset_index(drop=True)
display(results_df.style.format({c: '{:.4f}' for c in results_df.columns if c != 'Model'}))
```

	Model	Accuracy	Precision (weighted)	Recall (weighted)	F1-score (weighted)
0	SimpleCNN	0.8259	0.8389	0.8259	0.8097
1	DeeperCNN	0.1444	0.0895	0.1444	0.0705

```
In [11]: for name, (y_true, y_pred) in all_predictions.items():
    print()
    print('=' * 70)
    print(name)
```

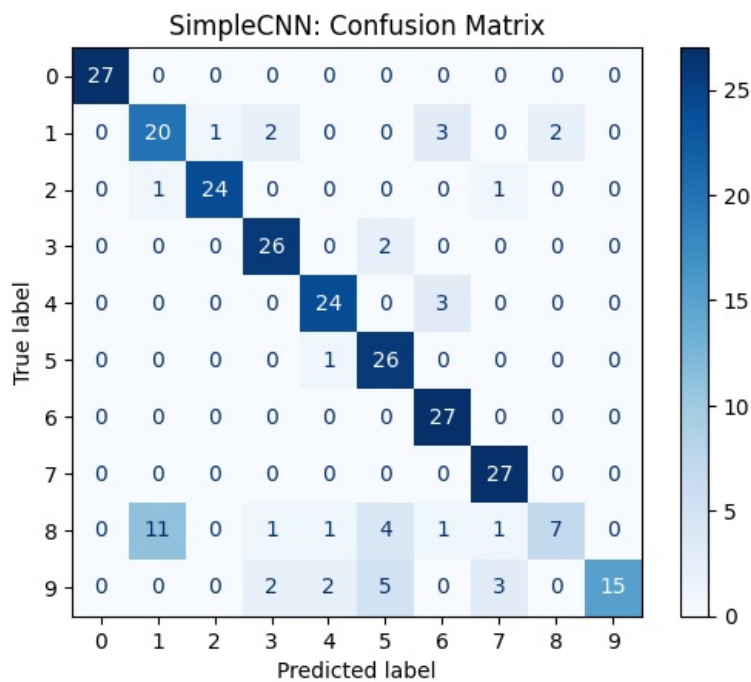
```
print(classification_report(y_true, y_pred, target_names=class_names, zero_division=0))
```

```
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap='Blues', values_format='d')
plt.title(f'{name}: Confusion Matrix')
plt.show()
```

=====

SimpleCNN

	precision	recall	f1-score	support
0	1.00	1.00	1.00	27
1	0.62	0.71	0.67	28
2	0.96	0.92	0.94	26
3	0.84	0.93	0.88	28
4	0.86	0.89	0.87	27
5	0.70	0.96	0.81	27
6	0.79	1.00	0.89	27
7	0.84	1.00	0.92	27
8	0.78	0.27	0.40	26
9	1.00	0.56	0.71	27
accuracy			0.83	270
macro avg	0.84	0.82	0.81	270
weighted avg	0.84	0.83	0.81	270



=====

DeeperCNN

	precision	recall	f1-score	support
0	0.67	0.22	0.33	27
1	0.14	1.00	0.25	28
2	0.00	0.00	0.00	26
3	0.08	0.18	0.11	28
4	0.00	0.00	0.00	27
5	0.00	0.00	0.00	27
6	0.00	0.00	0.00	27
7	0.00	0.00	0.00	27
8	0.00	0.00	0.00	26
9	0.00	0.00	0.00	27
accuracy			0.14	270
macro avg	0.09	0.14	0.07	270
weighted avg	0.09	0.14	0.07	270

