Name: Alvin James
Student ID: 23111190
Github link: https://github.com/RoronoaJames/OCR_CNN_PyTorch.git

The Power of OCR using CNN and PyTorch

Introduction

Optical Character Recognition (OCR) is a powerful technique that enables computers to extract text from images. This is particularly useful in applications like digitizing handwritten documents, automating data entry, and improving accessibility. In this tutorial, we will explore how Convolutional Neural Networks (CNNs) can be leveraged for OCR using PyTorch. Our focus is to demonstrate that even with image augmentations, OCR remains robust and effective. This tutorial will show how OCR works, why it is beneficial, and how data scientists can use it without relying on AI-generated solutions.

Why OCR Matters for Data Scientists?

OCR is a crucial tool in data science, enabling the extraction of meaningful text from scanned images, handwritten notes, and even vehicle license plates. As a data scientist, mastering OCR allows you to:

- **Automate Data Collection**: Extract text from images without manual entry.
- **Enhance Accessibility**: Convert physical documents into searchable digital formats.
- **Improve Efficiency**: Reduce human errors and speed up text-processing tasks.

While OCR has existed for decades, modern techniques using CNNs have significantly improved its accuracy and flexibility. With PyTorch, implementing an OCR system becomes intuitive, scalable, and highly customizable.

Why CNN and PyTorch for OCR?

OCR requires models that can recognize patterns and extract key features from images. CNNs are highly effective because they:

- Use **convolutional layers** to detect edges, curves, and shapes in text images.
- Retain spatial relationships in images, making them ideal for handwritten text recognition.
- Can be **trained on large datasets** to generalize well on unseen text samples.

PyTorch is chosen as the deep learning framework because:

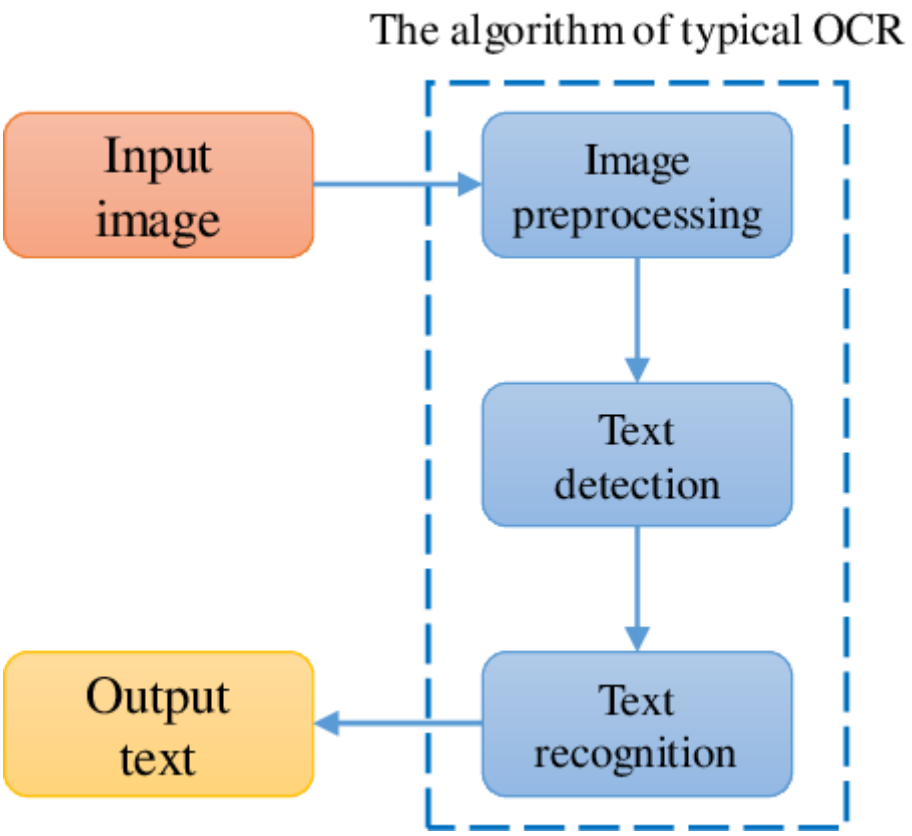
- It offers an **intuitive and flexible** platform for training neural networks.
- It supports **dynamic computation graphs**, making debugging easier.
- It has a vast ecosystem with pre-built **datasets, transformations, and optimizers**.

For those new to CNNs or PyTorch, these references provide in-depth learning:

- **CNNs**: Deep Learning with Python

- **PyTorch**: [PyTorch Official Documentation]

How does OCR work?



The OCR engine or OCR software works by using the following steps:

Image acquisition

A scanner reads documents and converts them to binary data. The OCR software analyzes the scanned image and classifies the light areas as background and the dark areas as text.

Pre-processing

The OCR software first cleans the image and removes errors to prepare it for reading. These are some of its cleaning techniques:

- Deskewing or tilting the scanned document slightly to fix alignment issues during the scan.
- Despeckling or removing any digital image spots or smoothing the edges of text images.
- Cleaning up boxes and lines in the image.

- Script recognition for multi-language OCR technology

Text recognition

The two main types of OCR algorithms or software processes that an OCR software uses for text recognition are called pattern matching and feature extraction.

Pattern matching

Pattern matching works by isolating a character image, called a glyph, and comparing it with a similarly stored glyph. Pattern recognition works only if the stored glyph has a similar font and scale to the input glyph. This method works well with scanned images of documents that have been typed in a known font.

Feature extraction

Feature extraction breaks down or decomposes the glyphs into features such as lines, closed loops, line direction, and line intersections. It then uses these features to find the best match or the nearest neighbor among its various stored glyphs.

Post-Processing

After analysis, the system converts the extracted text data into a computerized file. Some OCR systems can create annotated PDF files that include both the before and after versions of the scanned document.

OCR Implementation: MNIST Dataset

Dataset Overview

The MNIST dataset is a subset of a larger set available from NIST. The database is also widely used for training and testing in the field of machine learning. It is widely used for digit recognition and consists of:

- 60,000 training images
- 10,000 test images
- Grayscale 28x28 pixel images of handwritten digits (0-9)

To evaluate OCR performance, we train our model under two conditions:

1. **Without augmentations** (baseline performance)
2. **With augmentations** (evaluating robustness)

CNN Architecture for OCR

Model Architecture Used:

1. **Two Convolutional Layers**: Extract low and high-level features.
2. **Max-Pooling Layers**: Reduce the feature map size.
3. **Fully Connected Layers**: Convert extracted features into class probabilities.
4. **Softmax Activation**: Classify input images into digits (0-9).

Detailed CNN Components:

- **Activation Function**: ReLU (Rectified Linear Unit) is used to introduce non-linearity.
- **Pooling**: MaxPooling is applied to downsample the feature maps.
- **Dropout**: A dropout layer (50%) prevents overfitting.
- **Loss Function**: CrossEntropyLoss is used for multi-class classification.
- **Optimizer**: Adam optimizer is used for efficient gradient updates.

Implementing CNN in PyTorch

1. Define the CNN Model

```python
# Define CNN Model
class OCR_CNN(nn.Module):
    def __init__(self):
        super(OCR_CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 64 * 7 * 7)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

2. Train the Model

```
model = OCR_CNN()

criterion = nn.CrossEntropyLoss()
```

optimizer = optim.Adam(model.parameters(), lr=0.001)

```python
# Train the Model

def train(model, dataloader, criterion, optimizer, epochs=5):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        for images, labels in dataloader:
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch {epoch+1}, Loss: {running_loss/len(dataloader):.4f}")
```

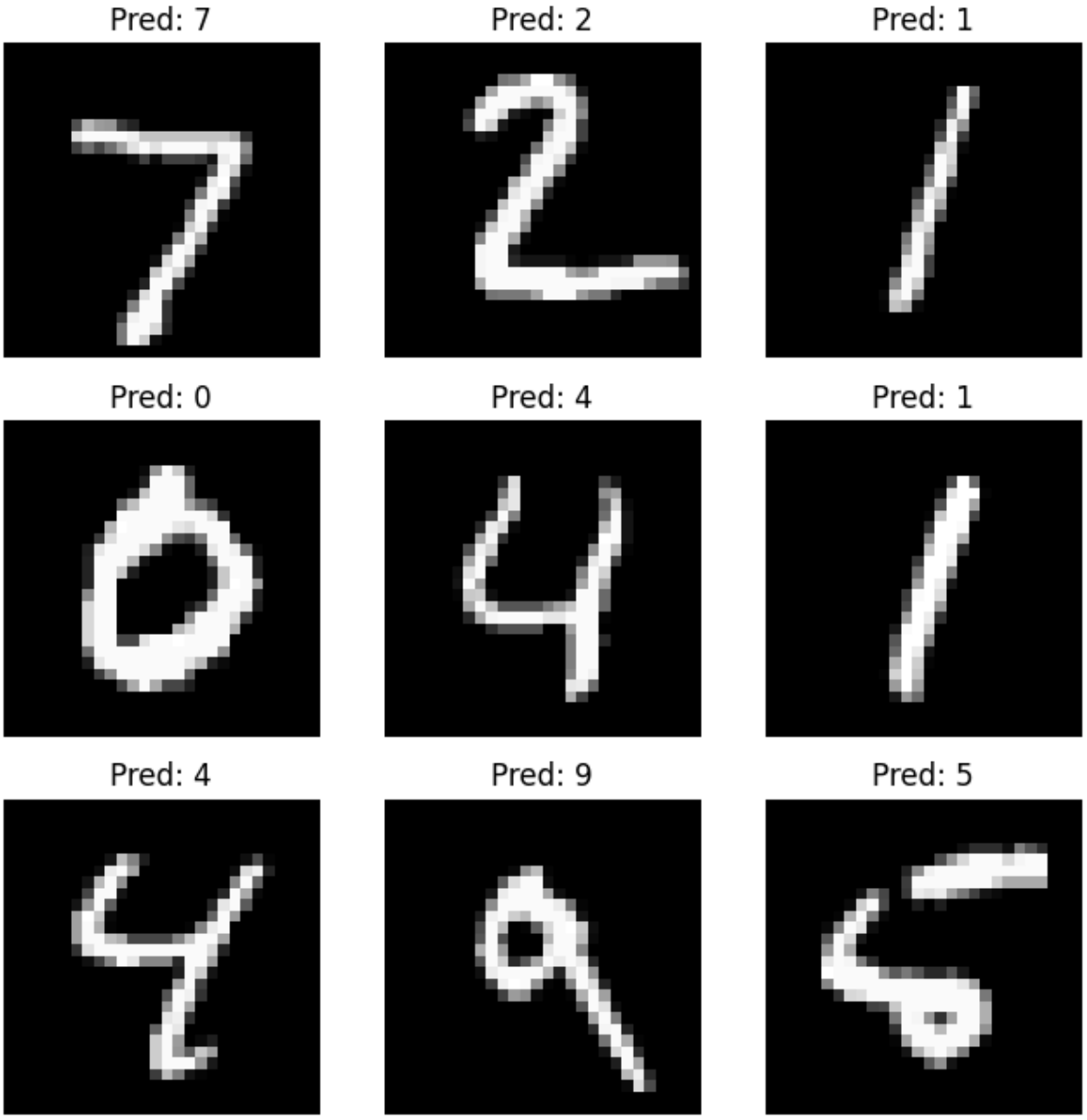
3. Evaluate Model Accuracy

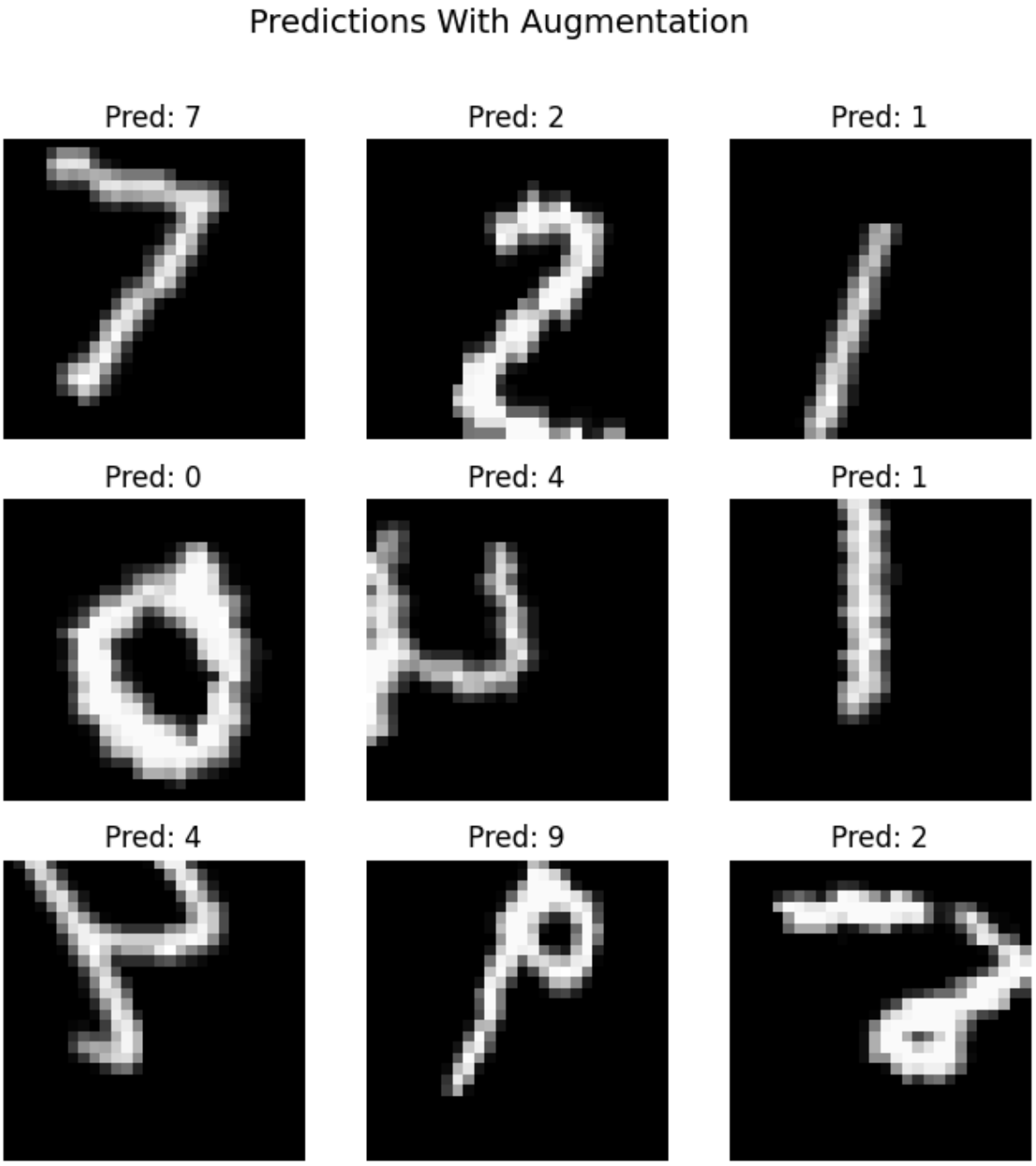
```python
# Evaluate Model

def evaluate(model, test_dataloader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_dataloader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print(f"Accuracy: {100 * correct / total:.2f}%")
```

Results & Observations

Prediction Results


Predictions Without Augmentation
Pred: 7
Pred: 2
Pred: 1
Pred: 0
Pred: 4
Pred: 1
Pred: 4
Pred: 9
Pred: 5



Above is an example of predictions made by our CNN model without and without applying augmentations:

Observations:

- The model correctly classified most of the digits.
- Some misclassifications still occur, indicating possible room for improvement through additional training or architectural modifications.

Accuracy Summary

- **Without Augmentations**: Model achieved **99.21% accuracy**.

- **With Augmentations**: Model accuracy dropped to **90.23%**.

Conclusion

This tutorial has demonstrated how CNNs can perform OCR effectively using PyTorch. By applying image augmentations, we have shown that OCR models can be trained to be resilient to distortions, making them highly useful in real-world applications.

References

1. Yann LeCun et al., "Gradient-Based Learning Applied to Document Recognition."
2. PyTorch Documentation: https://pytorch.org/docs/stable/index.html
3. MNIST Dataset: http://yann.lecun.com/exdb/mnist/