Optical Character Recognition (OCR) in Computer Vision – Detailed Notes

---

🎯 Learning Objectives

By the end of these notes, students will be able to:

- Understand the concept and applications of OCR
- Explain the OCR pipeline with appropriate tools and techniques
- Apply preprocessing techniques to enhance text recognition
- Identify and handle challenges in handwritten text recognition
- Implement a basic OCR system using Python and OpenCV

---

1. What is OCR?

Optical Character Recognition (OCR) is a process of detecting and converting printed or handwritten text in images (scanned documents, photos, etc.) into machine-encoded text.

---

2. Applications of OCR

Area	Description
Document Digitization	Converting physical books, forms, or invoices into editable digital text
Banking	Processing cheques and forms
Law Enforcement	License plate recognition
Healthcare	Digitizing patient records
Accessibility	Text-to-speech apps for visually impaired people

---

3. OCR Pipeline

Step 1: **Image Preprocessing**

To prepare the image for text recognition:

- **Grayscale Conversion**: Reduces dimensionality by removing color.
- **Noise Removal**: Median or Gaussian blur to reduce small dots or distortion.
- **Thresholding**: Binary or Otsu's thresholding to separate foreground (text) from background.

- **Morphological Operations**: Dilation/Erosion to connect broken characters.
- **Skew Correction**: Align tilted text using Hough Transform or deskewing.

Step 2: **Text Detection**

Identify the regions containing text.

- **Classical methods**: Contours, Edge detection, MSER (Maximally Stable Extremal Regions)
- **Deep learning**: EAST (Efficient and Accurate Scene Text detector), CRAFT

Step 3: **Text Recognition**

Extract actual characters.

- **Tesseract OCR** (for printed text)
- **EasyOCR** (deep learning-based)
- **TrOCR** (transformer-based for printed and handwritten)

Step 4: **Postprocessing**

- Spell correction (using libraries like `pyspellchecker`)
- Regular expression filtering
- Language model assistance

---

4. Preprocessing Techniques to Enhance Text Recognition

Preprocessing improves the quality of the input image, which directly impacts OCR accuracy. Common techniques include:

Technique	Purpose
Grayscale conversion	Simplifies the image by reducing it to one channel
Thresholding (Otsu's, Adaptive)	Converts to binary, emphasizing text
Gaussian/Morphological blur	Removes noise while preserving edges
Deskewing	Corrects slanted text for better OCR
Dilation and Erosion	Refines character edges and fills gaps

---

5. ✍Challenges in Handling Handwritten Text

OCR with **handwritten text** is more complex than printed text due to variations in style, shape, and alignment. Two key challenges and solutions are:

Challenge 1: Variability in Writing Style

Problem: Different people write letters differently (e.g., 'a', 'g').

Solution:

- Use **deep learning models** like CNNs and RNNs trained on **large handwriting datasets** (e.g., IAM dataset).
- Data augmentation techniques to simulate variations.

Challenge 2: Overlapping and Touching Characters

Problem: Characters may not be clearly separated, leading to misclassification.

Solution:

- Use **connected component analysis** or **segmentation** to separate touching characters.
- Employ **CRNN (Convolutional Recurrent Neural Network)** or **CTC (Connectionist Temporal Classification)** loss to recognize sequences without character-level segmentation.

---

6. Python Code Example (Using Tesseract & OpenCV)

```
import cv2
import pytesseract

# Load and preprocess image
image = cv2.imread('handwritten_sample.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.medianBlur(gray, 3)
thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)[1]

# Run Tesseract
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'
custom_config = r'--oem 3 --psm 6'
text = pytesseract.image_to_string(thresh, config=custom_config)

print("Detected Text:", text)
```

---

7. Tools and Libraries

Tool	Use
OpenCV	Image preprocessing
Tesseract	OCR engine
EasyOCR	Deep learning-based text recognition
TrOCR	Transformer-based OCR
Pillow	Image processing
PyTesseract	Python wrapper for Tesseract

---

8. Summary

- OCR converts images of text into machine-readable characters.
- Preprocessing like thresholding, blurring, and deskewing is vital.
- Handwritten text recognition needs deep learning models due to high variability.
- Tools like Tesseract, EasyOCR, and OpenCV are essential for building OCR systems.

Tools and Libraries – With Code Samples

1. **OpenCV**

Purpose: Image preprocessing (grayscale conversion, thresholding, blurring, etc.)

```
import cv2

# Load image and convert to grayscale
image = cv2.imread('image.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

cv2.imshow("Thresholded Image", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

---

2. **Tesseract OCR**

Purpose: Optical character recognition engine for printed or scanned text.

```
import pytesseract
from PIL import Image
```

```
# Load image using Pillow
img = Image.open('sample.png')

# Run OCR
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'
text = pytesseract.image_to_string(img)
print("Extracted Text:", text)
```

📌 *Make sure Tesseract is installed on your system and added to the PATH.*

---

3. **EasyOCR**

Purpose: Deep learning-based OCR, supports over 80 languages, works well with handwritten and complex layouts.

```
import easyocr

# Create a Reader object
reader = easyocr.Reader(['en'])

# Perform OCR on image
result = reader.readtext('sample.jpg')

# Print results
for detection in result:
    print(detection[1])
```

Install with: `pip install easyocr`

---

4. **TrOCR (Transformer OCR - by Microsoft)**

Purpose: State-of-the-art OCR using transformer-based vision-text models.

```
from transformers import TrOCRProcessor, VisionEncoderDecoderModel
from PIL import Image
import torch

# Load model and processor
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-
handwritten')
model = VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-
handwritten')

# Load and preprocess image
image = Image.open("handwritten_sample.jpg").convert("RGB")
```

```
pixel_values = processor(images=image, return_tensors="pt").pixel_values

# Generate text
generated_ids = model.generate(pixel_values)
text = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
print("TrOCR Output:", text)
```

Install with: `pip install transformers`

---

5. **Pillow**

Purpose: Image loading, conversion, cropping, and filtering.

```
from PIL import Image, ImageFilter

# Load and apply filter
img = Image.open("text_sample.png")
filtered = img.filter(ImageFilter.SHARPEN)

filtered.show()
```

Pillow is often used with pytesseract or OpenCV.

---

6. **PyTesseract**

Purpose: Python wrapper for Tesseract OCR engine to integrate it in Python projects.

```
import pytesseract
import cv2

img = cv2.imread("doc.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# OCR
text = pytesseract.image_to_string(gray)
print("Detected Text:", text)
```

☑ Summary Table (Quick Reference)

Tool	Main Function	Python Package
OpenCV	Image processing & preprocessing	`opencv-python`
Tesseract	OCR engine for printed text	Install separately
EasyOCR	Deep learning-based OCR	`easyocr`

Tool	Main Function	Python Package
TrOCR	Transformer-based OCR (SOTA)	`transformers`
Pillow	Image manipulation	`Pillow`
PyTesseract	Interface between Tesseract and Python	`pytesseract`
