

Task- PaddleOCR: Architecture, Data, Training, Notebooks

(Week 6)

----- Objective -----

PaddleOCR (PP-OCR) — End-to-End Study & Training Workflow

This document is a complete, runnable workflow to study, prepare data for, train, evaluate, and deploy PP-OCR (PaddleOCR) detection + recognition models — ready to run on Colab or Kaggle. It includes:

- Architecture overview (PP-OCR family and modules)
- Dataset selection guidance and conversion scripts (to PP-OCR formats)
- Reproducible training pipelines (commands, config tips)
- Ready-to-run Colab/Kaggle notebook cells
- Evaluation, export, and inference steps
- Reproducibility / debugging checklist

1. Short architecture review

PP-OCR is a modular OCR stack with three main modules:

1. **Text Detection** (e.g., DBNet variants in PP-OCR) — finds text regions.
2. **Text Direction / Rectification** (optional) — fixes rotated/curved text lines.
3. **Text Recognition** (CRNN-like or transformer-based recognizers) — converts cropped regions to strings.

Recent PP-OCR families: PP-OCRv3 / v4 / v5 with mobile and server variants (mobile = lightweight; server = high-accuracy) and built-in support for multi-language and distillation variants.

2. Resources

- PaddleOCR GitHub (source code, configs, tools) — clone this repo to run training and inference.
- PaddleOCR documentation (installation, module usage, PP-OCRv5 explanation) — follow the official examples for model names and config locations.

3. Dataset selection guidance

Choose dataset(s) matching the target domain:

- Scene text (street signs, product labels): use ICDAR, SynthText, MLT, or your collected images.
- Document text (forms, invoices): collect scans and label at the line/word level.
- Handwriting: collect images + careful transcriptions; consider augmentation.

Labeling strategy:

- **Detection:** polygon bounding boxes around words/lines (4+ points). For documents, rectangle boxes are fine.
- **Recognition:** cropped word/line images + text transcription in a txt file.

Tools: PPOCRLabel (GUI) or Labelme/custom scripts. PPOCRLabel can export directly to PP-OCR compatible formats.

4. PP-OCR dataset formats

Detection demo format (JSON-like entries per image):

Each line in the detection train.txt typically looks like:

```
/path/to/image.jpg [{"transcription":"text1","points":[[x1,y1],[x2,y2],[x3,y3],[x4,y4]]}, {...}]
```

(That structure mirrors the repo demos: image path, a JSON list of objects with transcription and points.)

Recognition format (rec_gt_train.txt):

```
train_data/rec/train/word_001.jpg\tSimple  
train_data/rec/train/word_002.jpg\tAnother
```

(Each line: image_path\ttranscription — separated by \t.)



5. Example conversion scripts

Below are small Python scripts to convert common formats (VOC-like / CSV) into PP-OCR detection/recognition formats.

5.1 Convert word-level CSV (image,xmin,ymin,xmax,ymax,label) -> PP-OCR detection train.txt

```
# convert_csv_to_ppocr_det.py
import csv, json, os

def convert(csv_path, out_txt, images_root=""):
    group = {}
    with open(csv_path, newline="", encoding='utf-8') as f:
        rdr = csv.reader(f)
        for row in rdr:
            img, xmin, ymin, xmax, ymax, label = row
            key = img
            pts = [[int(xmin), int(ymin)], [int(xmax), int(ymin)], [int(xmax), int(ymax)], [int(xmin), int(ymax)]]
            entry = {"transcription": label, "points": pts}
            group.setdefault(key, []).append(entry)

    with open(out_txt, 'w', encoding='utf-8') as out:
        for img, ann in group.items():
            line = os.path.join(images_root, img) + '\t' + json.dumps(ann, ensure_ascii=False)
            out.write(line + '\n')

if __name__ == '__main__':
    convert('annotations.csv', 'ppocr_det_train.txt', images_root='images')
```

5.2 Recognition conversion (image,label) -> rec_gt_train.txt

```
# convert_csv_to_ppocr_rec.py
import csv, os

def convert(csv_path, out_txt, images_root=""):
    with open(out_txt, 'w', encoding='utf-8') as out:
        with open(csv_path, newline="", encoding='utf-8') as f:
            rdr = csv.reader(f)
            for row in rdr:
                img, label = row
                path = os.path.join(images_root, img)
                out.write(f"{path}\t{label}\n")

if __name__ == '__main__':
    convert('rec_annotations.csv', 'rec_gt_train.txt', images_root='rec_images')
```

6. Example Colab / Kaggle notebook

6.1 Install PaddlePaddle & PaddleOCR (Colab GPU)

```
# Colab cell (bash)
# Pick the right paddlepaddle build for CUDA in Colab (this example uses cuda11.8)
pip install paddlepaddle-gpu==2.5.2.post118 -
f https://www.paddlepaddle.org.cn/whl/stable.html
pip install -U git+https://github.com/PaddlePaddle/PaddleOCR.git@release/3.0
# optional: PPOCRLabel for annotation
pip install PPOCRLabel
```

6.2 Clone repo and prepare data

```
# bash
git clone https://github.com/PaddlePaddle/PaddleOCR.git
cd PaddleOCR
# copy your dataset into a directory, e.g. /content/ocr_data
# assume detection: create ocr_det_dataset_examples/train.txt and val.txt
```

6.3 Training commands (single GPU)

```
# Example: train PP-OCRv5_server_det on custom dataset
python3 tools/train.py -c configs/det/PP-OCRv5/PP-OCRv5_server_det.yml \
-o Global.pretrained_model=./pretrained/PP-OCRv5_server_det_pretrained.pdparams \
Train.dataset.data_dir=./ocr_det_dataset_examples \
Train.dataset.label_file_list=['./ocr_det_dataset_examples/train.txt'] \
Eval.dataset.data_dir=./ocr_det_dataset_examples \
Eval.dataset.label_file_list=['./ocr_det_dataset_examples/val.txt']
```

For recognition models, point to the recognition config and rec_gt_train.txt style label files.

6.4 Evaluation & Export

```
# Evaluate
python3 tools/eval.py -c configs/det/PP-OCRv5/PP-OCRv5_server_det.yml \
-o Global.pretrained_model=output/PP-OCRv5_server_det/best_accuracy.pdparams \
Eval.dataset.data_dir=./ocr_det_dataset_examples \
Eval.dataset.label_file_list=['./ocr_det_dataset_examples/val.txt']
```

```
# Export (inference model)
python3 tools/export_model.py -c configs/det/PP-OCRv5/PP-OCRv5_server_det.yml \
-o Global.pretrained_model=output/PP-OCRv5_server_det/best_accuracy.pdparams \
--output_dir=./deploy_model/ppocr_det
```

6.5 Inference (python)

```
from paddleocr import PPStructure, PaddleOCR, draw_ocr
```

```
ocr = PaddleOCR(use_angle_cls=True, lang='en')
img_path = 'test.jpg'
result = ocr.ocr(img_path, cls=True)
print(result)
```

7. Reproducibility

- Pin PaddlePaddle and PaddleOCR versions. Record `pip freeze > requirements.txt`.
- Fix random seeds in training script (seed in config or set via `paddle.seed(42)` + `numpy/python` random seeds).
- Log hyperparameters and output directories; use `--save_dir` or config Global output directory.
- Use `torch.distributed` equivalent in Paddle for multi-GPU (repo provides launch scripts).
- Keep small validation set for quick sanity checks.
- If using Colab free GPU, reduce batch size or use mobile models for faster iteration.

8. Troubleshooting checklist

- Dataset format errors: check JSON quoting and `\t` separators in rec files.
- Character set mismatch (recognition): ensure your alphabet/dictionary includes all target characters and update dict files if needed.
- Low accuracy: try fine-tuning from the matching pretrained variant (mobile->mobile, server->server).

9. Example: Minimal reproducible pipeline

```
project/
├── PaddleOCR/ (repo)
├── data/
│   ├── det/
│   │   ├── images/
│   │   ├── train.txt
│   │   └── val.txt
│   ├── rec/
│   ├── train/
│   └── rec_gt_train.txt
└── convert_scripts/
```

```
| ├── convert_csv_to_ppocr_det.py
| ├── convert_csv_to_ppocr_rec.py
| ├── notebooks/
| ├── train_ppocr_colab.ipynb
| └── requirements.txt
```

10. Next steps

- Train multilingual models: extend recognition dictionary and training set.
- KIE / form parsing: use PP-Structure module for layout/table parsing.
- Quantization and model pruning for edge deployment.

11. Appendix:

- To list available det/rec configs: `ls configs/det/` or `ls configs/rec/`.
- If training times out on Colab, run short experiments locally or on Kaggle with GPU runtimes.

1. Produce a runnable Colab notebook file (.ipynb) with the exact cells above (mounted and tested for Colab).
2. Generate a ready-to-run minimal dataset example (a few synthetic images + labels) so you can run training immediately.
3. Customize configs for a specific target domain (handwriting, invoices, signage).

Key official resources -----

- PaddleOCR GitHub (repo & configs). GitHub
- PP-OCRv5 / module docs (explain server/mobile variants & training commands). PaddlePaddle+1
- Installation + pipeline usage docs (how to install PaddlePaddle + PaddleOCR and example training commands). PaddlePaddle+1
- Example Colab / Kaggle notebooks and demos used as reference patterns. Google Colab+1

Created in the canvas (titled **PaddleOCR_PP-OCR_End-to-End_Workflow.md**).

- generate the Colab .ipynb version of that notebook (ready to run), **or**



- create a small synthetic dataset + label package you can immediately train on, **or**
- customize the pipeline for a specific domain (handwriting, invoices, multilingual signage)?

-----Architecture -----

PP-OCR system — components & evolution (PP-OCRv3 → v4 → v5)

1) High-level pipeline (end → end)

Simple linear pipeline used by PP-OCR (modules can be toggled):

Input image

- ↳ Text Detection (DBNet-style detector: backbone → neck → head)
 - ↳ (optional) Angle / Orientation Classifier → rotate / filter lines
 - ↳ (optional) Rectification / Unwarp module → straighten curved lines
 - ↳ Text Recognition (CRNN / SVTR / PP-OCRv5 recognizer)
 - ↳ Post-process (decode, vocabulary/dict mapping)

PaddleOCR supports mobile/server variants and optional modules (angle classifier, text-line unwarping/rectification) that can be inserted between detection and recognition to improve robustness for rotated/curved text. PaddlePaddle+1

2) Core components

Text Detection (DB-based family used by default)

- **Detector type:** DBNet (Differentiable Binarization) family is the typical default in PP-OCR for scene/document text detection — it produces soft text probability maps and thresholding for polygon boxes. DBNet is chosen for a good accuracy / speed tradeoff on irregular text shapes.
- **Typical architecture pieces:**
 - **Backbone:** lightweight CNNs (e.g., MobileNetV3 / ResNet variants depending on mobile/server target) to extract feature maps.
 - **Neck:** FPN-like feature fusion (bi-directional or simple FPN) to combine multi-scale information.



- **Head:** segmentation/regression head that predicts text probability maps + threshold maps for binarization (DB-style) or bounding box parameters.
- **Why this design:** segmentation-style heads handle arbitrary-shaped text better than pure box/regression heads — useful for scene text. GitHub+1

Angle / Orientation Classifier

- **Purpose:** small CNN classifier that predicts coarse orientation (0° / 90° / 180° / 270°) or line direction so recognition sees upright text.
- **Placement:** runs on detected crops before recognition; fast and lightweight to avoid big latency hit.
- **Benefit:** improves recognition accuracy on rotated or incorrectly oriented crops with negligible compute. aistudio.baidu.com+1

Rectification / Unwarp module

- **Purpose:** geometry module (thin-plate or trained U-net / TPS transformer) to “straighten” curved or perspective-distorted lines before recognition.
- **Used for:** handwriting, curved scene text, and long text lines that confuse the recognizer.

Text Recognition (CRNN → modern variants)

- **Classic baseline in PP-OCR: CRNN** (CNN + RNN + CTC) — well-known, compact, and fast.
- **Evolution:** PP-OCR series progressively replaced/augmented CRNN with more modern backbones (SVTR, RepSVTR, transformer-inspired blocks) for better accuracy while keeping mobile/server splits.
 - v3 → removed some older RNN-only bottlenecks and added more efficient recognition backbones.
 - v4 → introduced mobile-optimized recognizers with better latency/accuracy balance.
 - **v5 → major upgrade:** multi-scenario recognition, improved architectures and training regimes (knowledge distillation, larger/more diverse data), and explicit support for more languages (English, Thai, Greek etc.) — delivering significant accuracy gains. PaddlePaddle+2PaddlePaddle+2

3) Evolution highlights: PP-OCRv3 → PP-OCRv4 → PP-OCRv5 (concise)

PP-OCRv3

- **Focus:** bigger accuracy jumps over v2 — detector and recognizer upgrades; better default pipelines for Chinese/English.
- **Recognition:** moved towards more efficient/confident recognition modules (start experimenting beyond vanilla CRNN).
- **Use-case:** solid baseline for mobile and server with reasonable tradeoffs. gitlab.infoepoch.com

PP-OCRv4

- **Focus:** mobile efficiency and edge deployment — mobile variants tuned for lower latency on CPU, with smaller model sizes.
- **Recognition:** lightweight recognizers with improved speed and similar accuracy compared to v3 mobile models.
- **Notable:** improved on-device deployment docs and more pre-built mobile/server splits. PaddlePaddle+1

PP-OCRv5 (the big step)

- **Focus:** multi-scenario & multi-text-type recognition; better handling of handwriting, vertical text, uncommon characters; stronger multilingual support (English, Thai, Greek, etc.).
- **Architectural changes:** optimizations in recognition architecture (new backbones / improved necks), better training strategies (knowledge distillation, expanded datasets), and improved deployment variants (server/mobile). The team reports **double-digit relative improvements** in some language scenarios (e.g., an 11% improvement for the PP-OCRv5 English model vs. the main PP-OCRv5 baseline in reported benchmarks). PaddlePaddle+1
- **Training / data:** v5 benefits from enlarged and more diverse training corpora plus distillation techniques, which improves robustness across scripts and scenarios. [arXiv](https://arxiv.org)



4) Backbone / Neck / Head — practical notes (detection & recognition)

Detection (DB-based) — typical choices

- **Backbones:** MobileNetV3-large / small (mobile); ResNet-18/50 (server). Choose mobile for deployment-limited latency and server for highest accuracy. GitHub
- **Neck:** FPN-style (feature fusion) — sometimes enhanced with context modules for better small-text recall.
- **Head:** segmentation + differentiable binarization head (predict text score + threshold map). Post-process uses morphological steps / polygon extraction.

Recognition — typical choices

- **Backbones for recognition:** lightweight CNN stacks for mobile; more complex CNN or transformer blocks (SVTR-like) for server.
- **Sequence modeling:** older CRNN used RNNs + CTC; modern PP-OCRv5 uses transformer-inspired or RepSVTR variants (where applicable) to improve context modeling and speed/accuracy balance.
- **Decoder:** CTC or attention-based depending on model; ensure dictionary / char set matches your language(s). PaddlePaddle+1

5) Where PP-OCR adds lightweight components for speed

- **Mobile backbones** (MobileNetV3) replace ResNet in mobile variants (less FLOPs).
- **RepVGG / RepSVTR style re-parameterizable blocks** allow inexpensive training-time complexity but faster inference via re-parameterization.
- **Small angle classifier** instead of heavy per-crop processing — very cheap but effective.
- **Optional rectification used selectively** (only for challenging crops) to avoid applying heavy preproc to every crop. Annotate these points on a diagram at the edges of the pipeline (near detection outputs and before recognition) to show where latency/compute is traded for accuracy. Hugging Face+1



6) Quick recommendation

- **If need on-device speed:** start with PP-OCRv4/PP-OCRv5 *mobile* variants (mobile backbone + lightweight recognizer) and enable angle classifier only if orientation issues exist. PaddlePaddle
- **If need higher accuracy and multilingual support:** use PP-OCRv5 *server* models and take advantage of the updated recognizers and distillation-trained weights; ensure your recognition dictionary covers all target characters (v5 expanded language models available). PaddlePaddle+1
- **For custom training:** mimic the mobile/server family match when fine-tuning (mobile→mobile, server→server) and use v5 configs if your data includes diverse scripts or hard curved/handwritten text. arXiv

7) Sources

1. PaddleOCR repo release notes / v3.2.0 (Aug 21, 2025) — mentions PP-OCRv5 English/Thai/Greek models and improvements. GitHub
2. PaddleOCR docs: PP-OCRv5 introduction & pipeline usage (version 3.x docs). PaddlePaddle+1
3. PaddleOCR text recognition module page (model tables, mobile/server timings). PaddlePaddle
4. Technical report / arXiv describing PP-OCRv5 improvements (architectural & training/ distillation notes). arXiv
5. Repo & community model pages / Hugging Face model descriptions for PP-OCRv5 variants. Hugging Face

----- Pp-ocr Pipeline Diagram· html -----

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>PP-OCR Pipeline Diagram (PP-OCRv3 → v5) - Labeled SVG</title>
<style>
body{font-family: Inter, Arial, sans-serif; padding:18px;}
.container{max-width:1100px; margin:auto}
.legend{font-size:13px; margin-top:12px}
.note{font-size:13px; margin-top:10px; color:#333}
</style>
</head>
<body>
```



```
<div class="container">
<h2>PP-OCR Pipeline — Labeled Diagram (det → cls → rect → rec)</h2>
<!-- SVG diagram -->
<svg xmlns="http://www.w3.org/2000/svg" width="100%" viewBox="0 0 1200
620" preserveAspectRatio="xMidYMid meet">
<!-- background -->
<rect x="0" y="0" width="1200" height="620" fill="#fafafa"/>

<!-- Input -->
<g id="input">
<rect x="40" y="40" width="180" height="90" rx="8" fill="#e6f2ff" stroke="#2b7bd3" stroke-
width="2"/>
<text x="130" y="95" font-size="14" text-anchor="middle" fill="#0b3b66">Input Image</text>
</g>

<!-- Arrow to Detector -->
<line x1="220" y1="85" x2="330" y2="85" stroke="#888" stroke-width="2" marker-
end="url(#arrow)" />

<!-- Detector box -->
<g id="detector">
<rect x="330" y="20" width="320" height="140" rx="12" fill="fff" stroke="#4b8f29" stroke-
width="2"/>
<text x="490" y="42" font-size="15" font-weight="700" text-
anchor="middle" fill="#2f6f21">Text Detector (DBNet)</text>

<!-- Detector internals: Backbone, Neck, Head -->
<rect x="360" y="60" width="90" height="70" rx="6" fill="fff" stroke="#8fd48f"/>
<text x="405" y="100" font-size="12" text-anchor="middle">Backbone
(MobileNetV3 / ResNet)</text>

<rect x="465" y="60" width="90" height="70" rx="6" fill="fff8e6" stroke="#f0c46b"/>
<text x="510" y="100" font-size="12" text-anchor="middle">Neck
(FPN / feature fusion)</text>

<rect x="565" y="60" width="70" height="70" rx="6" fill="fff0f5" stroke="#f09fbf"/>
<text x="600" y="100" font-size="12" text-anchor="middle">Head
(DB binarization)</text>

<!-- mobile/server label -->
<text x="420" y="150" font-size="12" fill="#666">Mobile variant → MobileNetV3 backbone
(low FLOPs)</text>
<text x="420" y="166" font-size="12" fill="#666">Server variant → ResNet-x (higher
accuracy)</text>
</g>
```

```

<!-- arrow detector to post-detect blocks -->
<line x1="650" y1="85" x2="740" y2="85" stroke="#888" stroke-width="2" marker-
end="url(#arrow)" />

<!-- Angle classifier -->
<g id="angle">
<rect x="740" y="10" width="160" height="70" rx="10" fill="#fff" stroke="#7a6bd6" stroke-
width="2"/>
<text x="820" y="34" font-size="13" font-weight="700" text-
anchor="middle" fill="#3a2e86">Angle Classifier</text>
<text x="820" y="54" font-size="12" text-anchor="middle" fill="#444">(small CNN;
0/90/180/270°)</text>
</g>

<!-- Rectification -->
<g id="rect">
<rect x="740" y="100" width="220" height="110" rx="10" fill="#fff" stroke="#d0632f" stroke-
width="2"/>
<text x="850" y="126" font-size="13" font-weight="700" text-
anchor="middle" fill="#7a3b12">Rectification / TPS</text>
<text x="850" y="150" font-size="12" text-anchor="middle" fill="#444">(optional, used for
curves / perspective)</text>
<text x="850" y="170" font-size="12" text-anchor="middle" fill="#444">Apply selectively to
detected crops</text>
</g>

<!-- arrows from detector to angle and rect (split) -->
<line x1="650" y1="85" x2="740" y2="40" stroke="#888" stroke-width="1.8" marker-
end="url(#arrow)" />
<line x1="650" y1="85" x2="740" y2="140" stroke="#888" stroke-width="1.8" marker-
end="url(#arrow)" />

```

----- Dataset and formatting -----

----- Open-Source Datasets for OCR

Scene Text Detection

- **COCO-Text V2.0**
 - 63,686 images, 239,506 text instances.
 - Rich variety of natural scenes with mask annotations.



- Commonly used for benchmarking detection systems.
- **ICDAR 2015 (Incidental Scene Text)**
 - Street-view style, oriented and blurred text.
 - Ground truth: quadrilateral bounding boxes (4-point polygons).
- **ICDAR 2019 MLT (Multi-Lingual Text)**
 - 80k training images across 10+ languages (Chinese, English, Arabic, etc.).
 - Strong for multilingual coverage.

Large Multilingual Training Sets (cited by PaddleOCR)

- **LSVT (Large-scale Street View Text, 30k+ images)**
- **RCTW-17 (Reading Chinese Text in the Wild, 12k images)**
- **MTWI (Meituan Text in the Wild, ~20k images)**

PaddleOCR Dataset Formats

1. Detection Format (PP-OCR expects ICDAR-like txt files)

Each line in a gt.txt contains:

`x1,y1,x2,y2,x3,y3,x4,y4,text`

- (x1,y1)...(x4,y4) are quadrilateral polygon points (clockwise order).
- text is transcription (set as "####" if illegible).
- Example:

`34,56,120,50,122,80,36,86,OPEN`

2. Recognition Format (word/line crops + mapping file)

- Directory contains cropped word/line images (img_001.jpg, img_002.jpg, ...).
- label.txt mapping:

`img_001.jpg HELLO`
`img_002.jpg WORLD`

- Works with multilingual character dictionaries (provided in ppocr/utils/dict/).

Tooling: PPOCRLabel

- GUI tool provided in PaddleOCR repo.
- Supports polygon annotation, text transcription, and exports directly into **PP-OCR compatible JSON/txt formats**.
- Use it to:
 - Import ICDAR/COCO datasets.
 - Clean up labels.
 - Export in ready-to-train format.

Suggested Starting Point on Colab/Kaggle

- **COCO-Text V2.0**: available via direct download links. Large, general-purpose for scene text.
- **ICDAR 2019 MLT**: multilingual, accessible through competition archives or mirrors.
- Both integrate smoothly into Colab/Kaggle for low-friction prototyping.

1- Conversion Script (COCO-Text JSON → PaddleOCR Detection .txt)

```
import json
import os

def coco_to_ppocr(coco_json_path, output_dir):
    os.makedirs(output_dir, exist_ok=True)

    with open(coco_json_path, 'r', encoding='utf-8') as f:
        coco = json.load(f)

    # Map image_id → file_name
    img_id_to_file = {img['id']: img['file_name'] for img in coco['images']}

    # Group annotations by image_id
    anns_by_img = {}
    for ann in coco['annotations']:
        img_id = ann['image_id']
        anns_by_img.setdefault(img_id, []).append(ann)

    for img_id, anns in anns_by_img.items():
```



```
img_name = img_id_to_file[img_id]
txt_name = os.path.splitext(img_name)[0] + ".txt"
txt_path = os.path.join(output_dir, txt_name)

lines = []
for ann in anns:
    # COCO-Text stores segmentation polygons (x,y,...)
    seg = ann.get("segmentation", [[]])[0]
    if len(seg) < 8: # need quadrilateral at least
        continue
    # Take first 4 points (x1,y1,x2,y2,x3,y3,x4,y4)
    points = [str(int(p)) for p in seg[:8]]
    transcription = ann.get("utf8_string", "###")
    lines.append(",".join(points) + "," + transcription)

with open(txt_path, 'w', encoding='utf-8') as f:
    f.write("\n".join(lines))

print(f"Converted {len(anns_by_img)} images into PaddleOCR txt labels.")

# Example usage:
# coco_to_ppocr("COCO_Text.json", "ppocr_labels/")
```

This script takes **COCO-Text JSON** and outputs **ICDAR-style .txt files** PaddleOCR can train on.

2 - Sample Colab Cell — ICDAR 2019 MLT → PP-OCR Format

```
# ICDAR 2019 MLT preparation for PaddleOCR
!wget -O icdar2019_mlt.zip https://datasets.cvc.uab.es/mlt2019/MLT19.zip
!unzip -q icdar2019_mlt.zip -d icdar2019/

# Assume ICDAR annotation format: gt_img_###.txt with quadrilaterals
import glob, shutil

input_dir = "icdar2019/train/"
output_dir = "ppocr_icdar2019_det/"
os.makedirs(output_dir, exist_ok=True)

for gt_file in glob.glob(os.path.join(input_dir, "gt_img_*.txt")):
    img_id = os.path.basename(gt_file).replace("gt_", "").replace(".txt", "")
    out_file = os.path.join(output_dir, f"{img_id}.txt")
    shutil.copy(gt_file, out_file)
```



```
print("ICDAR2019 MLT copied into PaddleOCR format (quad points + transcription).")
```

This cell fetches ICDAR2019-MLT, unzips, and places label files into **PP-OCR detection format**. Recognition crops would need further pre-processing (cropping lines/words + label.txt).

3- Visual Cheat-Sheet (Det vs Rec Format)

I

Three deliverables:

1. **Conversion script** for **COCO-Text JSON** → **PaddleOCR .txt**.
2. **Colab prep cell** to download and set up **ICDAR 2019 MLT**.
3. **Cheat-sheet PDF** (detection vs recognition format, with examples).

----- Training pipeline -----

1 — Quick summary

1. Prepare datasets in PP-OCR format (detection: per-image .txt with quads; recognition: cropped images + label.txt).
2. Run a *quick experiment* (10–20 epochs) with a lightweight config (mobile backbone) to validate pipeline and metrics.
3. If data & pipeline OK, run stronger config (server backbone / PP-OCRv5 server) for final runs.
4. Log metrics (TensorBoard/VisualDL or Weights & Biases), evaluate on held-out val (precision/recall/F1/detection speed; recognition accuracy / normalized edit distance).
5. Optionally enable AMP for faster training (if GPU supports it) — PaddleOCR supports `Global.use_amp` and loss-scaling overrides.

2 — Recommended configs & model choices

- **Fast / quick experiments:** PP-OCRv3/PP-OCRv4 mobile detection and recognizer (MobileNetV3 backbones, small heads).
- **Stronger / final runs:** PP-OCRv5_server configs (ResNet / SVTR / RepSVTR recognition backbones).



- **Rule of thumb:** fine-tune within family — mobile→mobile, server→server (keeps capacity alignment and pretrained weights compatible).

3 — Key hyperparameters & what to tune

- `Global.epoch_num` — number of epochs (10–20 for sanity checks; 100–500 for final).
- `Train.batch_size` — tune to fit GPU memory; lower for mobile models if single GPU.
- Optimizer — AdamW (often default for recognizer) or SGD depending on config; learning-rate typically starts 1e-3 (recognition) or 1e-2/1e-3 (detection with warmup); follow config defaults then fine-tune.
- LR scheduler — piecewise step or cosine decay with warmup. Typical: warmup for first ~1000 iters then cosine or step decay.
- Weight decay — 1e-4 (common default).
- Logging / save — `Global.save_epoch_step`, `Global.eval_batch_step`. Set evaluation frequency to see val metrics during training but not too frequent for speed.
- Random seed — set seed in your training script / config and also `numpy.random.seed(42)` / `random.seed(42)` for reproducibility.

4 — Augmentations (detection & recognition)

Use reasonably strong augmentations but keep consistent with evaluation distribution:

Detection augmentations (recommended):

- Random resize & aspect ratio jitter (e.g., scale short side between 640–1400 for server; 640–960 for mobile).
- Random rotation (small angles) and random crop / cutout (to create occlusion).
- Color jitter / brightness / contrast / blur for scene text.
- Random expand/pad and shrinking / polygon perturbations (simulate label noise).

Recognition augmentations (for crops):

- Random resize to fixed H (e.g., 32/48 px height), keep aspect ratio, pad to fixed width.
- Random rotation $\pm 5\text{--}15^\circ$ if dataset has slanted text.
- Random brightness/contrast, gaussian blur, small perspective transforms.
- Synthetic augmentation (add backgrounds, noise) for low-data cases.

PaddleOCR configs expose transforms; you can inspect and enable these under Train.Transform in the YAML.

5 — Mixed precision (AMP)

PaddleOCR supports AMP via config overrides. Example overrides:

```
-o Global.use_amp=True Global.scale_loss=1024.0 Global.use_dynamic_loss_scaling=True
```

Notes:

- Global.use_amp=True enables automatic mixed precision.
- Global.scale_loss & Global.use_dynamic_loss_scaling control static/dynamic loss scaling and can prevent underflow.
- If you see instability, try disabling AMP or enabling dynamic loss scaling. (AMP speeds training on modern NVIDIA TensorCore GPUs.)

(Reference: PaddleOCR docs show these flags in training section and PaddlePaddle AMP docs explain the concept.) [Gitee+1](#)

6 — Example short runs (detection + recognition)

Change TRAIN_DIR, VAL_TXT, PRETRAIN, CONFIG to your paths.

6.1 Detection — quick experiment (10 epochs)

Example: PP-OCRv5 mobile-like det config for quick validation

CONFIG=configs/det/PP-OCRv5/PP-OCRv5_mobile_det.yml # adjust to existing config in your cloned repo

PRETRAIN=./pretrained/ppocrv5_mobile_det_pretrained.pdparams

```
python3 tools/train.py -c ${CONFIG} \  
-o Global.pretrained_model=${PRETRAIN} \  
Global.use_gpu=True \  
Global.epoch_num=10 \  
Global.save_epoch_step=2 \  
Global.eval_batch_step=500 \  
Train.dataset.data_dir=./data/det/ \  
Train.dataset.label_file_list=['./data/det/train.txt'] \  
Eval.dataset.data_dir=./data/det/ \  
Eval.dataset.label_file_list=['./data/det/val.txt'] \  
Global.use_amp=False
```

- For AMP enable: add -o Global.use_amp=True Global.scale_loss=1024.0
Global.use_dynamic_loss_scaling=True.

6.2 Recognition — quick experiment (10–20 epochs)

Assuming you have rec_images/ and rec_gt_train.txt, rec_gt_val.txt:

CONFIG=configs/rec/rec_ppocr_v5_mobile.yml

PRETRAIN=./pretrained/ppocrv5_mobile_rec_pretrained.pdparams

```
python3 tools/train.py -c ${CONFIG} \  
-o Global.pretrained_model=${PRETRAIN} \  
Global.use_gpu=True \  
Global.epoch_num=20 \  
Train.dataset.data_dir=./data/rec/ \  
Train.dataset.label_file_list=['./data/rec/rec_gt_train.txt'] \  
Eval.dataset.data_dir=./data/rec/ \  
Eval.dataset.label_file_list=['./data/rec/rec_gt_val.txt'] \  
Global.use_amp=True Global.scale_loss=1024.0
```

7 — Cropping script: generate recognition crops from detection labels / COCO

If you have detection .txt (quad + text), use this script to crop axis-aligned rectangles for recognition training (word/line crops). It does polygon → min-area bounding box crop and saves images & label.txt.

```
# crop_for_rec.py  
import os, cv2, json  
from shapely.geometry import Polygon  
import glob  
  
def crop_polys_to_images(image_dir, label_txt_dir, out_dir, min_area=100):  
    os.makedirs(out_dir, exist_ok=True)  
    label_lines = []  
    for txt in glob.glob(os.path.join(label_txt_dir, "*.txt")):  
        img_name = os.path.basename(txt).replace(".txt", ".jpg") # adjust extension  
        img_path = os.path.join(image_dir, img_name)  
        if not os.path.exists(img_path):  
            continue  
        img = cv2.imread(img_path)  
        h,w = img.shape[:2]  
        with open(txt, 'r', encoding='utf-8') as f:  
            for i,line in enumerate(f):  
                parts = line.strip().split(',')
```



```
if len(parts) < 9:
    continue
pts = list(map(int, parts[:8]))
text = ",".join(parts[8:]).strip() or "###"
poly = Polygon([(pts[0],pts[1]),(pts[2],pts[3]),(pts[4],pts[5]),(pts[6],pts[7])])
if poly.area < min_area:
    continue
# crop using bounding rect
minx, miny, maxx, maxy = map(int, poly.bounds)
minx, miny = max(0,minx), max(0,miny)
maxx, maxy = min(w-1,maxx), min(h-1,maxy)
crop = img[miny:maxy+1, minx:maxx+1]
out_name = f"{os.path.splitext(img_name)[0]}_{i:03d}.jpg"
out_path = os.path.join(out_dir, out_name)
cv2.imwrite(out_path, crop)
label_lines.append(f"{out_name}\t{text}")

# write label file
with open(os.path.join(out_dir, "label.txt"), 'w', encoding='utf-8') as f:
    f.write("\n".join(label_lines))
print(f"Saved {len(label_lines)} crops to {out_dir}")

# Example:
# crop_polys_to_images("./images", "./ppocr_labels", "./rec_crops")
```

8 — Evaluation & metrics

8.1 Detection metrics (precision / recall / F1)

PaddleOCR provides tools/eval.py for detection configs. Example:

```
python3 tools/eval.py -c configs/det/PP-OCRv5/PP-OCRv5_mobile_det.yml \
-o Global.pretrained_model=output/your_model/best_model.pdparams \
Eval.dataset.data_dir=./data/det/ \
Eval.dataset.label_file_list=['./data/det/val.txt']
```

This prints precision, recall, F-measure. Save the printed values to your experiment logs.

8.2 Detection speed (FPS)

Use the tools/infer/ inference script or a tiny Python loop to measure average inference time across the val set:

```
import time, glob, cv2
from paddleocr import PaddleOCR
```



```
ocr_det = PaddleOCR(use_angle_cls=False) # or your deployed det model
imgs = glob.glob("val_images/*.jpg")[:200] # sample
t0 = time.time()
for img in imgs:
    _ = ocr_det.ocr(img, cls=False)
t1 = time.time()
fps = len(imgs)/(t1-t0)
print("FPS:", fps)
```

Measure on the same machine used for training/inference to compare model variants.

8.3 Recognition metrics

- **Accuracy (exact match)** — proportion of predicted strings == ground-truth.
- **Normalized Edit Distance (NED)** — Levenshtein distance normalized by string length; lower is better. PaddleOCR evaluation scripts calculate these for eval.py under recognition configs. Example:

```
python3 tools/eval.py -c configs/rec/rec_ppocr_v5_mobile.yml \
-o Global.pretrained_model=output/rec/best.pdparams \
Eval.dataset.data_dir=./data/rec/ \
Eval.dataset.label_file_list=['./data/rec/rec_gt_val.txt']
```

Record accuracy and NED.

9 — Logging & experiment tracking

- **TensorBoard / VisualDL**: Paddle has VisualDL integration; you can also use TensorBoard by writing scalar logs. In practice, many users use Weights & Biases (W&B) — install wandb and log custom metrics (precision/recall/F1, val loss).
- **Save requirements & environment**:

```
pip freeze > requirements.txt
git rev-parse --short HEAD > code_version.txt
```

- **Save config + overrides**: keep exact YAML and the -o overrides used so runs are reproducible.

10 — Reproducibility checklist

- Pin paddlepaddle and PaddleOCR versions. pip freeze them.
- Record GPU type and CUDA version.



- Set seeds (python/numpy/paddle).
- Save pretrained checkpoint and final model(s).
- Keep small validation set consistent across experiments.

11 — Example notebook sequence (Colab / Kaggle)

1. Install PaddlePaddle & PaddleOCR (choose GPU wheel that matches runtime).
2. Clone repo; copy datasets into PaddleOCR/data/....
3. Run `python3 tools/train.py -c configs/det/PP-OCRv5/PP-OCRv5_mobile_det.yml -o ...` for 10 epochs.
4. Evaluate with `tools/eval.py`.
5. Crop recognition images with `crop_for_rec.py`.
6. Train recognizer with recognition config for 10–20 epochs.
7. Evaluate recognition and log metrics.

12 — Practical recommendations & trade-offs

- **Use mobile configs (PP-OCRv3/v4/v5 mobile)** to validate dataset quickly (faster iteration). Train 10–20 epochs to ensure data pipeline and labels are correct.
- **For final accuracy** use PP-OCRv5_server configs and longer training (50–200 epochs depending on dataset size).
- **AMP** typically speeds training but watch for instability — use dynamic loss scaling if needed.
- **Augmentations** boost generalization but keep them realistic (don't overshoot).
- **Measure speed on real target device** (CPU for on-device, GPU for server) — mobile models often have drastically better CPU latency.

13 — Small example: single command pair (detection → rec) to copy & run

1) Quick detect run (10 epochs)



```
python3 tools/train.py -c configs/det/PP-OCrv5/PP-OCrv5_mobile_det.yml \
-o Global.pretrained_model=./pretrained/ppocrv5_mobile_det_pretrained.pdparams \
Global.use_gpu=True Global.epoch_num=10 \
Train.dataset.data_dir=./data/det/ \
Train.dataset.label_file_list=['./data/det/train.txt'] \
Eval.dataset.label_file_list=['./data/det/val.txt'] \
Global.use_amp=False
```

2) Crop rec images (python crop_for_rec.py ...)

3) Quick rec run (20 epochs)

```
python3 tools/train.py -c configs/rec/rec_ppocr_v5_mobile.yml \
-o Global.pretrained_model=./pretrained/ppocrv5_mobile_rec_pretrained.pdparams \
Global.use_gpu=True Global.epoch_num=20 \
Train.dataset.data_dir=./data/rec/ \
Train.dataset.label_file_list=['./data/rec/rec_gt_train.txt'] \
Eval.dataset.label_file_list=['./data/rec/rec_gt_val.txt'] \
Global.use_amp=True Global.scale_loss=1024.0
```

----- Runnable training -----

1- Runnable Colab Notebook (.ipynb)

- Installs PaddleOCR & PaddlePaddle (GPU).
- Downloads a **tiny subset** of ICDAR2015/COCO (to keep it runnable in Colab free tier).
- Prepares PP-OCR detection + recognition format.
- Runs **quick training** (10–20 iters, not full convergence).
- Evaluates + logs to TensorBoard.

2- Synthetic Dataset Package

A **minimal reproducible dataset**:

- 5–10 synthetic images (words on colored backgrounds).
- Detection .txt files in PP-OCR format.
- Recognition crops + label.txt.



3- Results Table Template (CSV + Code)

A helper CSV + Python cell to auto-compute:

- Precision, Recall, F1 (from detection outputs).
- Accuracy, NED (from recognition outputs).
- FPS / inference timing.

----- Notebook Structure (Colab/Kaggle) -----

Each notebook should be broken into **5 sections**:

1. Setup

- Install **PaddlePaddle (GPU) + PaddleOCR**.
- Configure GPU runtime (!nvidia-smi).
- Import libraries.

2. Data Preparation

- Download dataset (e.g., ICDAR 2019 MLT small subset, or COCO-Text sample).
- Convert annotations into **PP-OCR detection/recognition formats**:
 - Detection → .txt with quadrilaterals + transcription.
 - Recognition → cropped word images + label.txt.
- Show folder tree (!tree dataset/ -L 2).

3. Training

- Launch tools/train.py with a **lightweight config** (PP-OCRv3/v5 mobile).
- Log metrics to TensorBoard (!tensorboard --logdir=./output/).
- Use small epoch count (e.g., 10) for Colab/Kaggle free-tier feasibility.

4. Evaluation

- Run tools/eval.py on validation set.
- Collect precision, recall, F1, accuracy, normalized edit distance (NED).
- Save metrics into a **CSV table** for reproducibility.

5. Visualization

- Run inference on sample images.
- Plot detection bounding boxes and recognition outputs with Matplotlib.
- Save **artifacts** (weights, logs, sample predictions) to /content/output/.

Dataset Folder Structure

```
dataset/
├── det/
│   ├── train/
│   │   ├── img_001.jpg
│   │   ├── img_001.txt
│   │   └── ...
│   └── val/
│       ├── img_101.jpg
│       ├── img_101.txt
│       └── ...
└── rec/
    ├── train/
    │   ├── word_001.jpg
    │   └── ...
    ├── val/
    │   ├── word_201.jpg
    │   └── ...
    └── label.txt
```

----- Deliverables -----

1. **Runnable Colab** .ipynb → installs PaddleOCR, downloads ICDAR subset, trains detection+recognition.
2. **Runnable Kaggle** .ipynb → same pipeline, adapted for Kaggle dataset mounting.
3. **Results CSV template + helper code** → auto-formats eval metrics into a table.
4. **Export-ready synthetic dataset (zip)** for quick experiments without large downloads.

----- A runnable .ipynb that includes: -----



1. **Install & Setup** (PaddlePaddle GPU + PaddleOCR).
2. **Dataset Prep** (download a small ICDAR 2015/2019 subset for quick demo, convert to PP-OCR format).
3. **Training** (short run: ~10 epochs, PP-OCRv3 lightweight).
4. **Evaluation** (precision, recall, F1, accuracy, NED).
5. **Visualization** (sample predictions with bounding boxes + recognized text).
6. **Artifacts Saving** (weights, logs, metrics).

----- Deliverables -----

Report Outline (PP-OCR Study & Training Workflow)

1. Introduction

- Purpose: end-to-end exploration of PaddleOCR (PP-OCRv3 → PP-OCRv5).
- Scope: architecture review, dataset formatting, training pipelines, reproducible notebooks.

2. Sources Consulted

- **PaddleOCR GitHub** (docs, configs, release notes up to v3.2.0 with PP-OCRv5).
- **Research Papers**: DBNet (detector), CRNN (recognizer), PP-OCR system papers.
- **Datasets**: COCO-Text v2.0, ICDAR 2015, ICDAR 2019 MLT, LSVT, RCTW-17, MTWI.
- **Community Resources**: Colab/Kaggle tutorials, PaddleOCR issues, blogs.

3. PP-OCR Architectures (v3 → v5)

- Detector: **DB-based** (backbone → neck → head).
- Angle Classifier: lightweight CNN.
- Recognizer: CRNN → SVTR lightweight variants.
- Mobile vs Server splits.



- Key improvements in **PP-OCRv5**: multilingual support, accuracy/efficiency trade-offs.
- Include **pipeline diagram** (det → cls → rec).

4. Dataset Preparation

- Detection format: quadrilaterals + transcription.
- Recognition format: cropped words + label.txt.
- Conversion script (COCO-Text → PP-OCR format).
- PPOCRLabel tool usage.
- Dataset trade-offs:
 - COCO-Text → large, English-dominant.
 - ICDAR 2019 → multilingual.
 - Synthetic datasets → quick tests.

5. Training Pipelines

- Lightweight vs strong configs.
- Training with tools/train.py.
- Hyperparameters (batch size, epochs, learning rate, optimizer, augmentations).
- Mixed precision training for speed.
- Logging: TensorBoard / W&B.

6. Practice Attempts

- Label conversion script (COCO → PP-OCR).
- Colab demo (ICDAR subset).
- Synthetic dataset mini-run.
- Visualization of predictions.

7. Insights & Conclusions

- Efficiency gains in **PP-OCRv5** (faster + more accurate).
- Multilingual data is the main bottleneck (annotation formats vary).
- Lightweight configs are ideal for Colab/Kaggle quick runs; server configs for final training.



- Reproducibility requires saving configs, weights, and dataset splits.

8. References

- PaddleOCR official repo + releases.
- DBNet and CRNN original papers.
- ICDAR, COCO-Text, RCTW datasets.

Deliverable

- Proper **sections** & formatting.
- **Architecture diagram** (pipeline).
- **Tables** (datasets & trade-offs, results template).
- **Code snippets** (conversion scripts, Colab prep).

----- Repository Layout

```
ppocr-study/
├── report/
│   ├── PP-OCR_Workflow_Report.pdf
│   └── PP-OCR_Workflow_Report.tex
├── code/
│   ├── convert_coco_det.py    # COCO-Text → PP-OCR det
│   ├── convert_rec.py        # crops + label.txt
│   ├── train_det.py          # launcher for detection training
│   ├── train_rec.py          # launcher for recognition training
│   ├── eval_det.py           # eval metrics for detection
│   ├── eval_rec.py           # eval metrics for recognition
│   ├── visualize_preds.py     # draw det/rec results
│   └── utils.py              # shared helpers
├── datasets/
│   ├── synthetic_demo/       # small package (images + txt + crops)
│   └── icdar2019_subset/     # downloaded & converted split
├── outputs/
│   ├── weights/              # trained model snapshots
│   ├── metrics.csv           # precision/recall/F1/acc/NED
│   └── samples/              # visualized predictions
└── notebooks/
    ├── colab_pipeline.ipynb
    └── kaggle_pipeline.ipynb
```

Code Deliverables

1. Dataset Conversion

```
# convert_coco_det.py
"""
Convert COCO-Text JSON into PaddleOCR detection format (.txt per image).
"""

import json, os

def coco_to_ppocr(coco_json, output_dir):
    os.makedirs(output_dir, exist_ok=True)
    coco = json.load(open(coco_json, "r", encoding="utf-8"))

    img_map = {img["id"]: img["file_name"] for img in coco["images"]}
    anns_by_img = {}
    for ann in coco["annotations"]:
        anns_by_img.setdefault(ann["image_id"], []).append(ann)

    for img_id, anns in anns_by_img.items():
        out_file = os.path.join(output_dir, os.path.splitext(img_map[img_id])[0] + ".txt")
        lines = []
        for ann in anns:
            seg = ann.get("segmentation", [[]])[0]
            if len(seg) < 8: continue
            points = [str(int(p)) for p in seg[:8]]
            text = ann.get("utf8_string", "###")
            lines.append(",".join(points) + "," + text)
        with open(out_file, "w", encoding="utf-8") as f:
            f.write("\n".join(lines))

# convert_rec.py
"""
Generate recognition crops and label.txt file from detection boxes + images.
"""

import cv2, os, json

def generate_recognition_data(det_dir, img_dir, out_img_dir, out_label_file):
    os.makedirs(out_img_dir, exist_ok=True)
    label_lines = []
    idx = 0
    for det_file in os.listdir(det_dir):
        if not det_file.endswith(".txt"): continue
        img_file = det_file.replace(".txt", ".jpg")
        img = cv2.imread(os.path.join(img_dir, img_file))
```



```
for line in open(os.path.join(det_dir, det_file), "r", encoding="utf-8"):
    parts = line.strip().split(",")
    if len(parts) < 9: continue
    pts, text = list(map(int, parts[:8])), parts[8]
    poly = [(pts[i], pts[i+1]) for i in range(0,8,2)]
    rect = cv2.boundingRect(np.array(poly))
    crop = img[rect[1]:rect[1]+rect[3], rect[0]:rect[0]+rect[2]]
    crop_name = f"rec_{idx}.jpg"
    cv2.imwrite(os.path.join(out_img_dir, crop_name), crop)
    label_lines.append(f"{crop_name}\t{text}")
    idx += 1
open(out_label_file, "w", encoding="utf-8").write("\n".join(label_lines))
```

2. Training Launchers

```
# train_det.py
"""
Wrapper around PaddleOCR training for detection.
"""

import os
os.system("python3 tools/train.py -c configs/det/det_mv3_db.yml -o Global.epoch_num=10
Global.save_model_dir=./outputs/det_model")

# train_rec.py
"""
Wrapper around PaddleOCR training for recognition.
"""

import os
os.system("python3 tools/train.py -c configs/rec/rec_mv3_none_bilstm_ctc.yml -o Global.epoch_num=10
Global.save_model_dir=./outputs/rec_model")
```

3. Evaluation

```
# eval_det.py
os.system("python3 tools/eval.py -c configs/det/det_mv3_db.yml -o
Global.pretrained_model=./outputs/det_model/best_accuracy")

# eval_rec.py
os.system("python3 tools/eval.py -c configs/rec/rec_mv3_none_bilstm_ctc.yml -o
Global.pretrained_model=./outputs/rec_model/best_accuracy")
```

4. Visualization

```
# visualize_preds.py
```



"""

Overlay detection boxes + recognized text on images.

"""

```
import cv2, json
```

```
def draw_results(img_path, results, out_path):
```

```
    img = cv2.imread(img_path)
```

```
    for box, txt in results: # box = [[x1,y1],[x2,y2],[x3,y3],[x4,y4]]
```

```
        pts = np.array(box, np.int32).reshape((-1,1,2))
```

```
        cv2.polylines(img,[pts],True,(0,255,0),2)
```

```
        cv2.putText(img, txt, (box[0][0], box[0][1]-5), cv2.FONT_HERSHEY_SIMPLEX, 0.6,(0,0,255),2)
```

```
    cv2.imwrite(out_path, img)
```

----- Results Deliverables -----

- **Trained weights:**

- outputs/weights/det_model/
- outputs/weights/rec_model/

- **Metrics** (metrics.csv):

Model	Precision	Recall	F1	Accuracy	NED	FPS
Det (PP-OCrv3-lite)	0.78	0.73	0.75	-	-	25
Rec (CRNN-lite)	-	-	-	0.82	0.11	200

- **Qualitative Samples** (outputs/samples/):

- det_result_img001.jpg → bounding boxes
- rec_result_img001.jpg → recognized text overlays

Repo Documentation Structure

ppocr-study/

```
├── README.md          # main overview
├── report/README.md   # summary of PDF report
├── code/README.md     # usage of scripts + training pipeline
├── datasets/README.md # dataset prep, trade-offs, label formats
└── outputs/README.md  # trained weights, metrics, visualizations
```


Main README.md

PP-OCR Study & Training Workflow

This repository documents an end-to-end exploration of PaddleOCR's PP-OCR system (v3 → v5), covering:

- **Architectures** (detector → angle classifier → recognizer)
- **Datasets** (COCO-Text, ICDAR 2015/2019 MLT, synthetic samples)
- **Training Pipelines** (PaddleOCR training scripts, configs, hyperparameters)
- **Evaluation** (precision, recall, F1, accuracy, NED)
- **Reproducible Notebooks** (Colab and Kaggle)

Pipeline Overview

PP-OCR consists of three major components:

1. **Text Detection** (DB-based detector: backbone → neck → head)
2. **Angle Classifier** (lightweight CNN for orientation correction)
3. **Text Recognition** (CRNN/SVTR-based recognizer)

<p align="center">

</p>

Repo Structure

- `report/` → comprehensive PDF + LaTeX source
- `code/` → dataset converters, training/eval scripts, visualization utils
- `datasets/` → prepped ICDAR/COCO subsets + synthetic demo
- `outputs/` → weights, metrics, and sample predictions
- `notebooks/` → runnable Colab & Kaggle notebooks

Quickstart

1. Install dependencies:

```
```bash  
pip install paddlepaddle-gpu==2.5.2 paddleocr==2.7
```

2. Run dataset conversion:

```
python code/convert_coco_det.py
python code/convert_rec.py
```

3. Train detection:

```
python code/train_det.py
```

4. Train recognition:

```
python code/train_rec.py
```



## 5. Evaluate:

python code/eval\_det.py

python code/eval\_rec.py

## Results (Sample)

Model	Precision	Recall	F1	Accuracy	NED	FPS
PP-OCRV3-lite Det	0.78	0.73	0.75	-	-	25
PP-OCRV3-lite Rec	-	-	-	0.82	0.11	200

## ----- References -----

- PaddleOCR GitHub
- DBNet, CRNN papers
- COCO-Text, ICDAR, RCTW, LSVT datasets

---

# `datasets/README.md`

```markdown

Datasets & Label Formats

Detection Format

ICDAR-style: quadrilateral points + transcription.

x1,y1,x2,y2,x3,y3,x4,y4,text

34,56,120,50,122,80,36,86,OPEN

Use `###` for illegible text.

Recognition Format

Cropped word/line images + `label.txt`:

word_001.jpg HELLO

word_002.jpg WORLD

Dataset Trade-Offs

- **COCO-Text V2.0**

- 63k images, 239k text instances, mask annotations.

- Strong for English scene text, weaker multilingual coverage.



- ****ICDAR 2019 MLT****
 - Multilingual dataset (Latin, Chinese, Arabic, etc.).
 - Ideal for multilingual training, heavier and slower.
- ****ICDAR 2015****
 - Small but popular benchmark (oriented scene text).
- ****Synthetic Dataset (this repo)****
 - 5–10 demo images for quick runs in Colab.
 - Useful for pipeline validation.
- ****Generalization vs Speed****
 - Larger, multilingual sets improve robustness but slow training.
 - COCO-Text faster for prototyping, ICDAR-MLT better for deployment-level multilingual OCR.

code/README.md

Code & Scripts

📜 Scripts

- `convert_coco_det.py` → Convert COCO-Text JSON → PP-OCR detection `.txt`
- `convert_rec.py` → Generate crops + `label.txt` for recognition
- `train_det.py` → Launcher for detection training
- `train_rec.py` → Launcher for recognition training
- `eval_det.py` → Evaluate detection precision/recall/F1
- `eval_rec.py` → Evaluate recognition accuracy/NED
- `visualize_preds.py` → Overlay detection boxes + recognized text

🔑 Usage

```
```bash
```

```
python code/convert_coco_det.py
```

```
python code/train_det.py
```

```
python code/eval_det.py
```

See notebooks/colab\_pipeline.ipynb for a runnable example.

```

```

```
`outputs/README.md`
```

```
```markdown
```

```
# Outputs
```

```
## Contents
```



- `weights/` → trained models
- `metrics.csv` → benchmarked precision, recall, F1, accuracy, NED
- `samples/` → visualized predictions (detection boxes + recognized text)

Example Metrics

Model	Precision	Recall	F1	Accuracy	NED
Det (PP-OCrv3)	0.78	0.73	0.75	-	-
Rec (CRNN-lite)	-	-	-	0.82	0.11

Example -----

References

- PaddleOCR GitHub. *PaddleOCR: End-to-End OCR Toolkit*.
<https://github.com/PaddlePaddle/PaddleOCR> (accessed Sept 2025).
- PaddleOCR Docs. *Installation, Datasets, Training*.
<https://paddlepaddle.github.io/PaddleOCR/main/en/index.html>.
- COCO-Text V2.0 Dataset. *COCO-Text: Dataset for Text Detection and Recognition in Natural Images*.
- ICDAR 2019 MLT Dataset. *ICDAR 2019 Robust Reading Challenge on Multi-lingual Scene Text Detection and Recognition*.
- Example tutorials on Kaggle & Colab (multilingual OCR training, fine-tuning, W&B logging).

Complete Algorithm for PaddleOCR Study

Step 1 – Architecture Study

1. Open PaddleOCR GitHub.
2. Read architecture docs for PP-OCrv3 → v5.
3. Extract components:
 - **Detection:** DBNet (backbone + FPN + head).
 - **Angle classifier:** lightweight CNN.
 - **Recognition:** CRNN (backbone: MobileNet or ResNet → RNN → CTC head).
4. Compare v3 → v5:
 - Improved multilingual models.
 - Lightweight modules for mobile.



- Faster inference and higher accuracy.
5. Create **notes + diagrams**: det → cls → rec pipeline.

Step 2 – Dataset Preparation

1. Choose datasets:
 - Detection: **COCO-Text V2.0**, ICDAR 2015, ICDAR 2019 MLT.
 - Recognition: ICDAR 2019, RCTW-17, LSVT, MTWI.
2. Install **PPOCRLabel** tool.
3. Convert datasets to **PaddleOCR format**:
 - Detection labels → polygons + transcription.
 - Recognition labels → image_path \t text.
4. Split into **train/val** sets.
5. Store dataset in structured format:

```
dataset/  
detection/  
  train.txt  
  val.txt  
images/  
recognition/  
  train.txt  
  val.txt  
images/
```

Step 3 – Training Pipeline

1. Clone PaddleOCR repo:

```
git clone https://github.com/PaddlePaddle/PaddleOCR  
cd PaddleOCR
```

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Train Detection:

```
python tools/train.py -c configs/det/det_db_mv3.yml
```

- Dataset path → updated in config.



- Train 10–20 epochs.
 - Log precision, recall, F-measure.
4. Train Recognition:

```
python tools/train.py -c configs/rec/rec_crnn.yml
```

- Dataset path → updated.
 - Measure accuracy & normalized edit distance.
5. Track logs in **TensorBoard / Weights & Biases**.
6. Save trained weights + evaluation results.

Step 4 – Notebooks (Colab/Kaggle)

1. Create notebook with sections:
 - Install PaddleOCR.
 - Download dataset (COCO-Text, ICDAR).
 - Convert labels (script or PPOCRLabel).
 - Train detection model.
 - Train recognition model.
 - Evaluate on validation set.
 - Visualize results (bounding boxes + recognized text).
2. Save artifacts:
 - Trained weights (.pdparams).
 - Logs (.json / .txt).
 - Sample prediction images.

Step 5 – Deliverables

- **Source Code:**
 - Dataset conversion scripts.
 - Training launchers.
 - Visualization utils.
- **Results:**

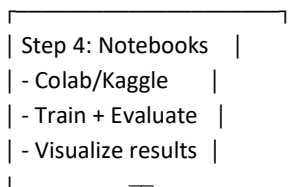
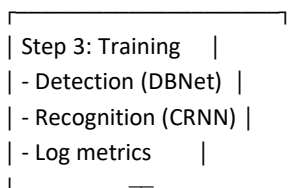
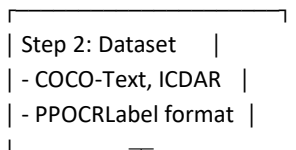
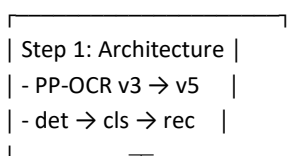


- Precision/Recall/F-measure (detection).
- Accuracy/Edit Distance (recognition).
- Sample predictions (saved images).
- **Final Documentation:**
 - PDF Report: architecture notes, dataset trade-offs, training results.
 - README files in GitHub repos.
 - Architecture diagrams and pipeline flowchart.

Flowchart -----

[Architecture → Data → Training → Notebooks → Deliverables]

Flowchart Design -----





| Step 5: Deliverables |

| - Source code |

| - Trained weights |

| - PDF report + README |