Project Overview

This project involves building an end-to-end document classification system using a subset of the **RVL-CDIP dataset (small version)**. The system preprocesses grayscale document images, trains a deep learning model (with EfficientNet as backbone), and deploys the trained model via a Flask API to classify document types such as:

- **Advertisement**
- **Form**
- **Handwritten**
- **Invoice**
- **Letter**

Dataset Used

Dataset: RVL-CDIP (Ryerson Vision Lab Complex Document Information Processing) – *Small Subset*

- **Classes used (5 out of 16 original):**
    - Letter
    - Form
    - Handwritten
    - Advertisement
    - Invoice
- **Sample Size (per class):**
    - Train: 50 images
    - Validation: 15 images
- **Total images used:**
  5 classes × (50 train + 15 val) = 325 images

Project Pipeline

Dataset Organization

```
/data
├── train/
│   ├── Advertisement/
│   ├── Form/
│   ├── Handwritten/
│   ├── Invoice/
│   └── Letter/
└── val/
    └── (same structure)
```

- Used shutil and random to create stratified train/val splits.
- Capitalized class names for consistency in labeling.

Image Preprocessing

- **Resized** to 128×128
- **Grayscale conversion**
- **Otsu thresholding** for binarization
- **Normalized** pixel values to [0, 1]
- Saved as .npy NumPy arrays for fast loading.

Data Augmentation

To improve generalization, the following real-time augmentations were applied using tf.py_function:

- Random **rotation** (±15°)
- Random **zoom** (90%–110%)
- **Brightness** adjustment (0.8× to 1.2×)
- **Gaussian noise** addition
- **Gaussian blur** (3×3 kernel)

All augmentations preserve image shape and normalize intensity afterward.

Model Architecture

- Input: (128, 128, 1)
- 2× Conv2D → MaxPooling
- Projected to 3 channels via Conv2D(3×1x1)
- Passed through **EfficientNetB0** (pretrained on ImageNet)
- Dense layer (256 ReLU) + Dropout
- Softmax output (5 classes)

Evaluation

- Trained for **50 epochs**
- Monitored train/validation accuracy and loss
- Generated:

- **Training curves**
- **Classification Report**
- **Confusion Matrix**
- **Per-class Accuracy**
- **Sample predictions visualization**

Deployment

- Deployed using **Flask REST API**

Challenges Faced

1. HDF5 Loading Bug in Keras 3

- Saving the model as .h5 caused deserialization errors due to InputLayer and batch_shape.
- Solution: Switched to **Keras v3 format (.keras)** or TensorFlow's **SavedModel** directory format.

2. Augmentation Latency

- tf.py_function for augmentation introduced overhead.
- Used .prefetch() and .AUTOTUNE to reduce bottlenecks.

3. **Dataset Size vs. Performance**

- Small subset accelerated iteration but risked underfitting real-world variance.

Future Improvements

Model & Accuracy

- **Switch to MobileNetV3 or EfficientNet-Lite** for faster inference and better suitability on edge devices.
- **Use transfer learning properly** by fine-tuning more layers rather than only using include_top=False.
- **Train with more classes** from the full RVL-CDIP dataset to enhance generalizability.

Augmentation & Data

- Implement advanced augmentation techniques such as:
    - **Elastic distortions**
    - **CutMix** or **Mixup**
    - **Grid distortions or random erasing**

- Use **synthetic data generation** for low-resource classes (e.g., handwritten or invoices).
- Consider **semi-supervised learning** if more unlabeled data is available.

Monitoring & Evaluation

- Implement **model monitoring** tools (like Prometheus + Grafana or ELK stack) to detect prediction drift.
- Track **per-class confidence**, **inference time**, and **failure cases** in production.
- Incorporate **uncertainty estimation** using MC Dropout or deep ensembles to flag low-confidence predictions.