Deep Document Unwarping (Final Project)

Goal: To design and train a deep learning pipeline that reconstructs a flat, readable document from an image of a crumpled or folded page, demonstrating mastery of **Geometric Deep Learning** and **Transformer Architectures**.

Material Covered: This project draws primarily from **Neural Networks**, **Vision Transformers (ViTs)**, **Synthetic Dataset Processing**, and **Differentiable Warping**.

Project Overview

Unlike the Midterm Project, where you computed a global homography for planar documents, this project deals with **non-rigid deformations** (crumples, folds, and curves). A simple 3 x 3 matrix cannot solve this; you need a model that predicts pixel-wise displacement.

You will build a system that:

1. Takes a **crumpled document image** as input.
2. Uses a **pretrained** backbone to understand the 3D geometry.
3. Predicts a **UV Map** (a dense coordinate lookup table).
4. Resamples the pixels to produce a (hopefully) flat, rectified output.

Key Technical Steps

Your system must implement the following pipeline using **PyTorch**:

1. The Dataset

The dataset is located in `renders/synthetic_data_pitch_sweep/` and is structured to separate geometry from appearance.

- **`rgb/`**: The input warped document images (JPEG).
- **`ground_truth/`**: The target flat paper textures (PNG).
- **`uv/`**: (Optional) The UV coordinate maps.
- **`border/`**: (Critical) Document boundary masks to distinguish paper from the background.

Dataset Task: You are provided with an elaborate starter script `dataset_loader.py`. You must use this to load the data, but you may want to expand upon it (e.g., adding augmentations). The loader script also contains boilerplate / starter code for implementing

the model and running the training loops - take a good look there!

3. Key Technical Steps

Your system should implement the following rough pipeline using **PyTorch**:

A. The Architecture (Encoder-Decoder)

You should implement a network that predicts a **deformation field** (UV map or Flow field).

- **Backbone (Encoder):** Do not train from scratch. Use a pre-trained backbone via timm or transformers.
    - *Options:* ResNet-50, EfficientNet, or any of the Vision Transformers (ViT/Swin).
- **Decoder:** Use a decoder (consider U-Net with **skip connections**) to preserve spatial details from the encoder.
- **Head:** A final convolution layer producing **2 channels** (representing x, y displacement or u, v coordinates).

B. Differentiable Unwarping (grid_sample)

The network does not output the flat image directly. It predicts the *geometric transformation*. You must use torch.nn.functional.grid_sample to perform differentiable warping.

- **Logic:** Output = GridSample(Input, Predicted_Flow)
- **Note:** grid_sample expects coordinates in the range [-1, 1]. If your model predicts UVs in [0, 1], you must normalize them before sampling.

C. Loss Functions (The "Geometry First" Approach)

Because the ground truth has different lighting than the input, standard pixel losses (MSE) may fail. You must experiment with losses that prioritize structure:

1. **Structural Similarity (SSIM):** *Highly Recommended.* Measures structural alignment rather than pixel intensity.
2. **Masked Loss:** Use the masks in the border/ folder to compute loss *only* on the document pixels. This prevents the model from wasting capacity on the background.
3. **Perceptual Loss (Optional):** Uses VGG features to compare high-level content rather than raw pixels.

Here are SotA papers that you may find useful for picking an architecture for your model:

- DocTr: Document Image Transformer for Geometric Unwarping and Illumination Correction
- DewarpNet: Single-Image Document Unwarping With Stacked 3D and 2D Regression Networks 😊
- Geometric Representation Learning for Document Image Rectification
- UVDoc: Neural Grid-based Document Unwarping
- DocUNet: Document Image Unwarping via a Stacked U-Net

It would be advisable to **only implement the geometry reconstruction part** of these papers instead of the entire method, which may be too involved and go into things like re-lighting methods which we do not focus on in this exercise.

Consult the `dataset_loader.py` script for getting started! Also provided is a `README.md` file with *a lot* of further technical details, code samples and explanations - read through it.

4. Milestones

Suggested milestones to guide your work:

- **Milestone 1:** Implement a simple Encoder-Decoder (baseline) and train with MSE loss. Visualize the output.
- **Milestone 2:** Integrate a pre-trained backbone (e.g., ResNet from `timm`) and implement SSIM or Perceptual Loss.
- **Milestone 3 (Advanced):** Implement masking to ignore background pixels and experiment with direct UV supervision or Flow Smoothness.

Feel free to implement your method in whatever way you wish, but working in a stratified, incremental way would definitely help your progress. Create a simple solution first that works, and then try a fancier method.

System Requirements

Requirement	Details
Framework	**PyTorch** (Python 3)
Compute	Requires **GPU** acceleration (Google Colab T4/A100 or local CUDA).
Architecture	Must use a backbone (Transformer / CNN).
Automation	Similar to the midterm, the inference script must process a folder of images automatically without manual intervention.
Input/Output	For every input crumpled_N.png, save rectified_N.png and predicted_uv_N.png (visualization).

6. Deliverables

Submit a compressed file containing:

1. **model.py**: Your PyTorch model definition (Encoder, Decoder, and Unwarping logic).
2. **train.py / train.ipynb**: The code used to train the model, including the training loop and loss curves.
3. **evaluate.py / inference.py**: A script that loads weights and calculates SSIM on the validation set.
4. **best_model.pth**: Your trained model weights.
5. **README_REPORT.md (or PDF)**: A brief 1-2 page report detailing:
    - **Architecture Diagram:** A representation of your flow (visuals encouraged).
    - **Loss Function Choice:** Justification for using SSIM, Perceptual, or Masked losses.
    - **Results:** A comparison figure showing: [Input] → [Predicted Flow/UV] → [Rectified Result] → [Ground Truth].
    - **Metrics:** Your final SSIM scores.

Academic Integrity & Grading

- **AI Tools:** As with the midterm, ***you are encouraged*** to use AI tools (Claude, Gemini, Codex, Cursor, etc.) to build and debug PyTorch code, errors or understand the APIs. However, the architecture design and the training logic must be your own (perhaps, AI-collaborative) work.

- **Grading Philosophy:**
    - We understand that training Transformers on limited compute is difficult. You will **not** be penalized if the final rectification is slightly blurry or has artifacts.

    - **Grading Focus:**
        1. Correct implementation of the **Encoder-Decoder** shapes (handling the tokens-to-grid reshape correctly).
        2. Correct implementation of the **Differentiable Warping** / **UV Mapping**.
        3. Evidence of **Loss Convergence** (the loss went down over time).
        4. Clarity of the **Code and Report**.

    - **Note**: Since this is an advanced task in computer vision (e.g. for which a very good method may actually award you an article publication in a reputable venue) we will not be grading strictly based on your *results*, rather we will look at the method and your choices. By all means if you wish to submit your method as a paper in any relevant venue (CVPR, ICCV, ECCV, ACCV, BMVC, WACV, DAS, ICDAR, IJDAR, DocEng, ACM MM, etc.) - that is encouraged!

Good luck, and enjoy working with the state-of-the-art!