电子科技大学
University of Electronic Science and Technology of China

WebCV24: Web Toolkit for Edge Detection, OCR, and Image Deblurring

Implementing cutting-edge computer vision techniques into an accessible web toolkit

GODFRED DOE

202424080114

OUTLINE

- Introduction
- Techniques and Methods
- Implementation
- Demonstration and Results
- Conclusion

INTRODUCTION

Objective:

- To create a web-based platform that combines three key CV techniques:
  - Edge detection with canny edge detection
  - Optical Character Recognition (OCR) using Tesseract
  - Image deblurring with MPRNet

Motivation:

- Simplify access to advanced CV tools for technical and non-technical users

Edge Detection

What is Canny Edge Detection?

- A technique to detect edges in images by identifying areas of rapid intensity change

- Steps:
    1. Noise reduction (Gaussian filter)
    2. Gradient calculation
    3. Non-maximum suppression
    4. Edge tracking by hysteresis

Optical Character Recognition

What is OCR?

Converts images of text into machine-readable text

Technology Used:

Google's Tesseract OCR engine

Approach:

- **Preprocessing steps for better OCR results: binarization, resizing, and noise removal**
- Pytesseract **utilized with native python support**

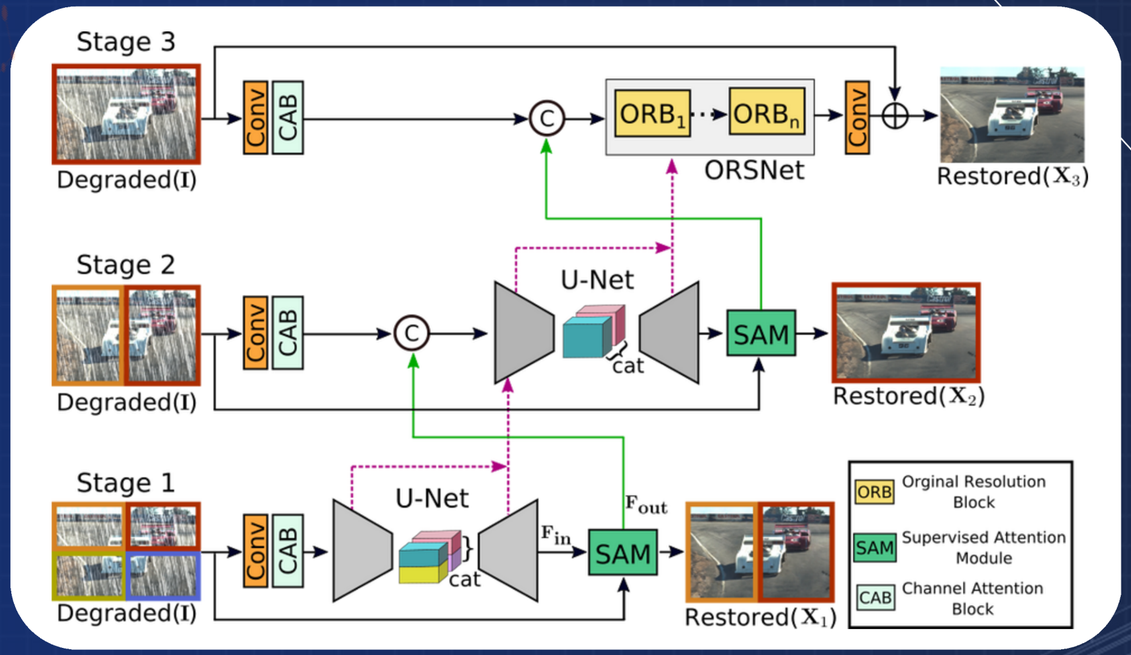
Image Deblurring with MPRNet

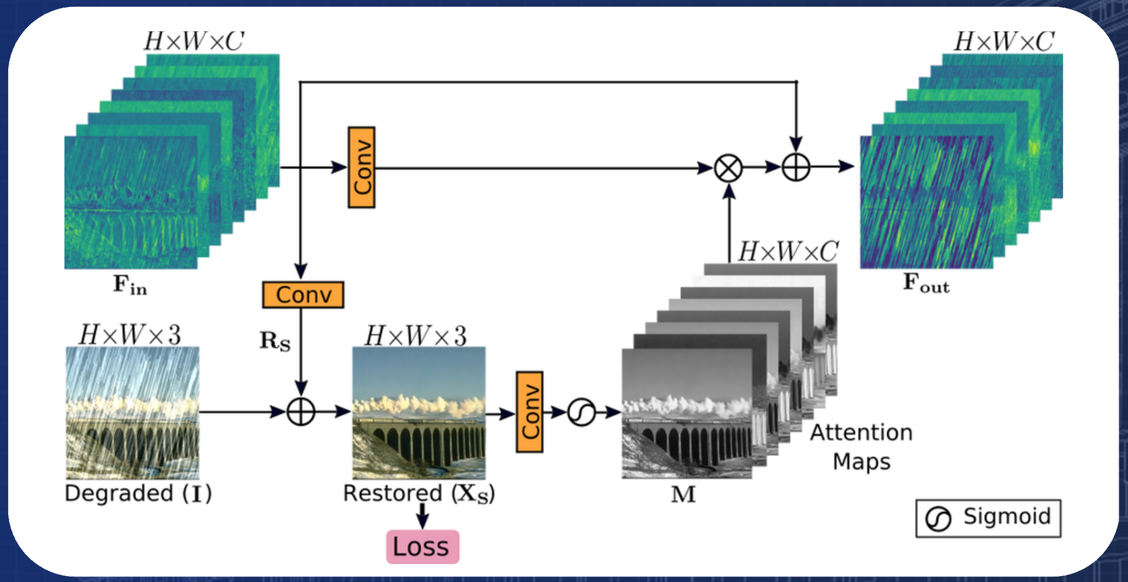
What is MPRNet?

- A state-of-the-art multi-stage network for restoring blurry images

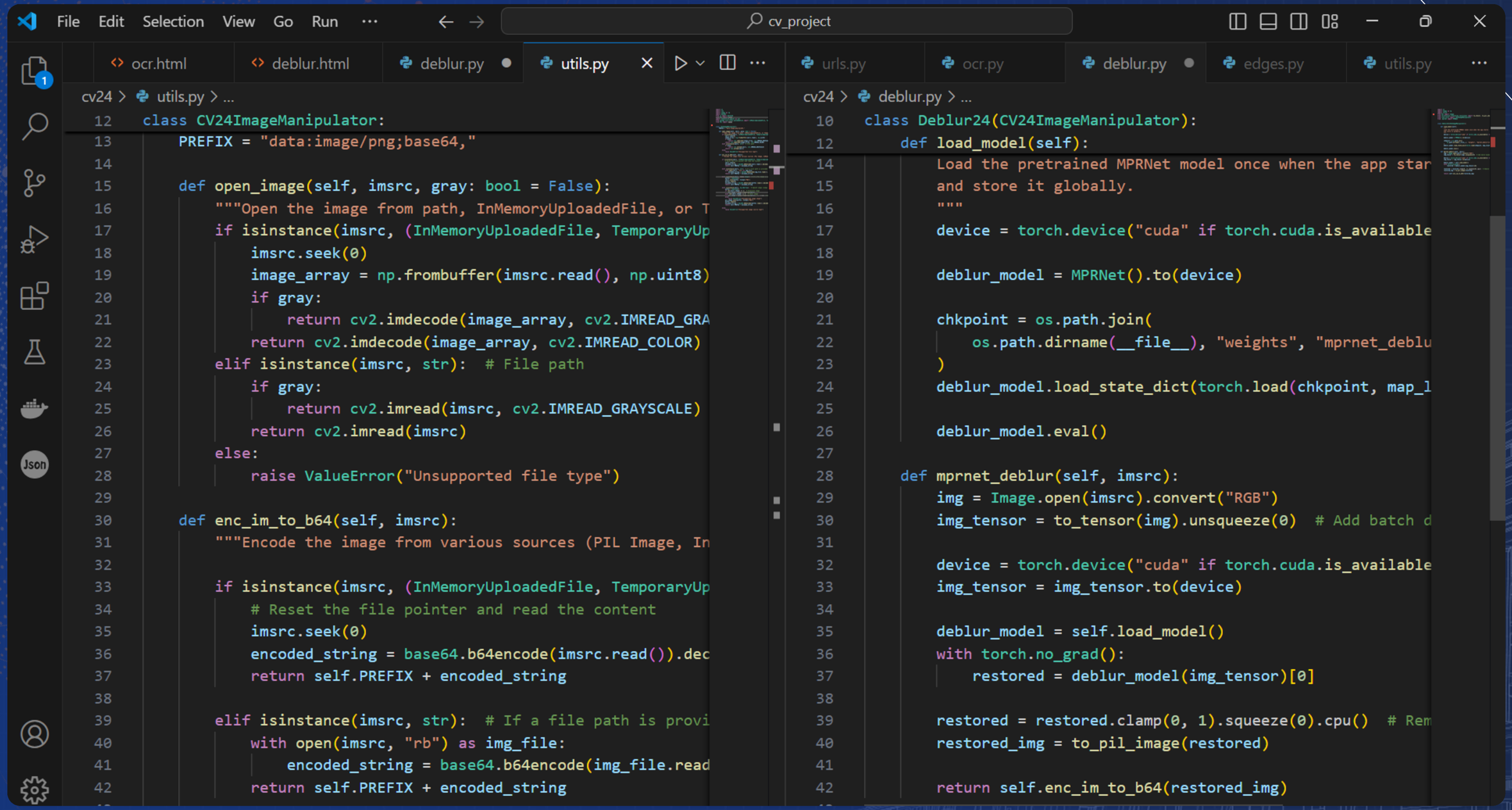
Key Features:

- Encoder-Decoder
- Supervised Attention Model (SAM)
- Cross-Stage Feature Fusion (CSFF)
- Pretrained weights used for efficient inference





WebCV24 Implementation



WebCV24 Implementation

EXPLORER

CV_PROJECT
- cv24
  - __pycache__
  - static
  - templates
  - weights
  - __init__.py
  - admin.py
  - apps.py
  - deblur.py
  - edges.py
  - models.py
  - mprnet.py
  - ocr.py
  - tests.py
  - urls.py
  - utils.py
  - views.py
- cvp
- venv
- .gitignore
- db.sqlite3
- Kupyn_DeblurGAN_Bl...
- manage.py
- requirements.txt
- Zamir_Multi-Stage_Pr...

OUTLINE
TIMELINE

views.py | mprnet.py | apps.py | ocr.html

cv24 > views.py > detect_edges

```python
def detect_edges(request: HttpRequest):
    if request.method == "POST":
        image = request.FILES.get("image")
        detector = EdgeDetector()
        output = detector.apply_canny(image)
        original = detector.enc_im_to_b64(image)
        context = {"original": original, "output": ou
        return render(request, "cv24/edge.html", cont

    context = {}
    return render(request, "cv24/edge.html", context)


def perform_ocr(request: HttpRequest):
    if request.method == "POST":
        image = request.FILES.get("image")
        ocr = OCR()
        output = ocr.apply_ocr(image)
        original = ocr.enc_im_to_b64(image)
        context = {"original": original, "output": ou
        return render(request, "cv24/ocr.html", conte

    context = {}
    return render(request, "cv24/ocr.html", context)


def deblur_image(request: HttpRequest):
    if request.method == "POST":
        image = request.FILES.get("image")
        deblur = Deblur24()
        original = deblur.enc_im_to_b64(image)
```

urls.py | ocr.py | deblur.py | utils.py

cv24 > ocr.py > OCR > preprocess

```python
class OCR(CV24ImageManipulator):
    def preprocess(self, image):
        """
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY
        _, thresh = cv2.threshold(
            gray, 128, 255, cv2.THRESH_BINARY
        )  # Binary thresholding
        return thresh

    def apply_ocr(self, img_src, preprocess=False, la
        """
        Extract text from an image and mark recognize
        """
        try:
            # Read the image for highlighting
            img = self.open_image(img_src)

            # Optionally preprocess the image
            if preprocess:
                img = self.preprocess(img)

            # Perform OCR to get text and bounding bo
            h, w, _ = img.shape
            data = pytesseract.image_to_boxes(img, la

            # Draw bounding boxes
            for box in data.splitlines():
                b = box.split()
                char, x1, y1, x2, y2 = b[0], int(b[1]
                # Tesseract coordinates are in a flip
                y1, y2 = h - y1, h - y2
                cv2.rectangle(img, (x1, y2), (x2, y1)
```

main | 0 0 | 0

Ln 17, Col 1 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.10.10 ('venv': venv) | Prettier

Demonstration and Results

Image Choose File No file chosen Detect Edges

Lower Threshold 100 Upper Threshold 200

Input Image

Output Image

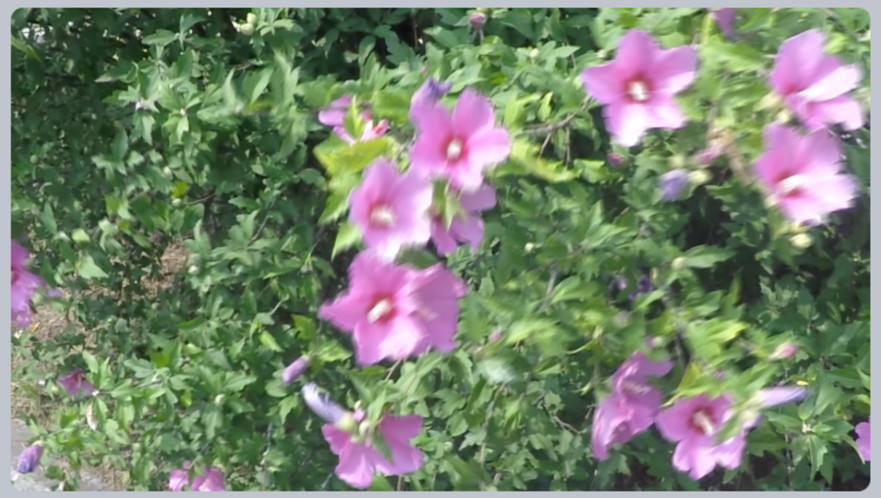
Demonstration and Results

127.0.0.1:8000/deblur/

C V 2 4 Edge Detection Optical Character Recognition Image Deblurring

Image Choose File No file chosen Deblur Image

Input Image



Deblurred Image

Demonstration and Results

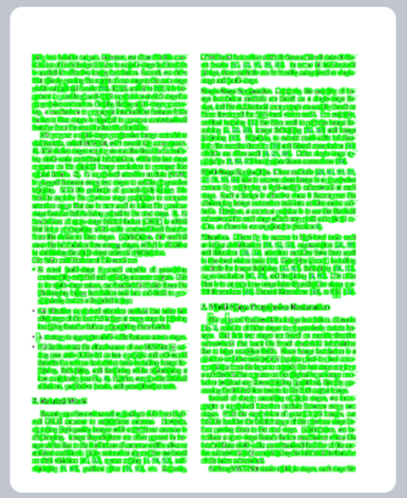
127.0.0.1:8000/ocr/

C V 2 4 | Edge Detection | Optical Character Recognition | Image Deblurring

Image | Choose File | No file chosen | Perform OCR

Input Image

Output Image



Recognized Texts

cally less reliable outputs. However, we show that the combination of both design choices in a multi-stage architecture is needed for effective image restoration. Second, we show that naively passing the output of one stage to the next stage yields suboptimal results [53]. Third, unlike in [88], it is important to provide ground-truth supervision at each stage for

Conclusion

- WebCV24 successfully integrates essential computer vision techniques into an easy-to-use web platform

- Other computer vision techniques can be added in future works and extra parameters can be added for users to experiment with the features

Thank You