Task 4: Unsharp Masking [illegible] High-boost Filtering

Image: datasets/mega_space_molly.jpg

Generated: 2025-11-02 00:50:15

[illegible] UNSHARP MASKING:

Unsharp masking [illegible]
[illegible]

[illegible]:
1. [illegible]: blur = Gaussian_blur(original)
2. [illegible] mask: mask = original - blur
3. [illegible] mask [illegible]: sharpened = original + mask

[illegible]:

$$g(x,y) = f(x,y) + [f(x,y) - \bar{f}(x,y)]$$

[illegible]:

$f(x,y)$ = [illegible]

$\bar{f}(x,y)$ = [illegible]

$[f(x,y) - \bar{f}(x,y)]$ = unsharp mask

[illegible] HIGH-BOOST FILTERING:

High-boost filtering [illegible] Unsharp Masking [illegible]
[illegible] k (amplification factor)

[illegible]:

$$g(x,y) = k \cdot f(x,y) - \bar{f}(x,y)$$

[illegible]:

$$g(x,y) = (k-1) \cdot f(x,y) + [f(x,y) - \bar{f}(x,y)]$$

[illegible]:

$k = 1.0$: [illegible] unsharp masking [illegible]

$k > 1.0$: [illegible] (high-boost)

$k < 1.0$: [illegible]

PARAMETERS [illegible]:

- Kernel size: 5×5
- Sigma (σ): 1.0
- k values: 1.0, 1.5, 2.0, 2.5

Unsharp Masking Process

Original Image

Blurred (Gaussian 5x5, σ=1.0)

Unsharp Mask (Original - Blurred)

Sharpened (Unsharp Masking)

High-boost Filtering - Effect of k Parameter

Original Image

Unsharp Masking
(k = 1.0)

High-boost Filtering
(k = 1.5)

High-boost Filtering
(k = 2.0)

High-boost Filtering
(k = 2.5)

```
PARAMETERS:
• Kernel size: 5x5
• Sigma (σ): 1.0

OBSERVATION:
• k = 1.0: Standard unsharp masking
• k = 1.5: Moderate sharpening
• k = 2.0: Strong sharpening
• k = 2.5: Very strong sharpening

Note: Higher k values amplify
edges and details more, but may
also amplify noise.
```

Detailed Comparison: Original vs Enhanced

Original

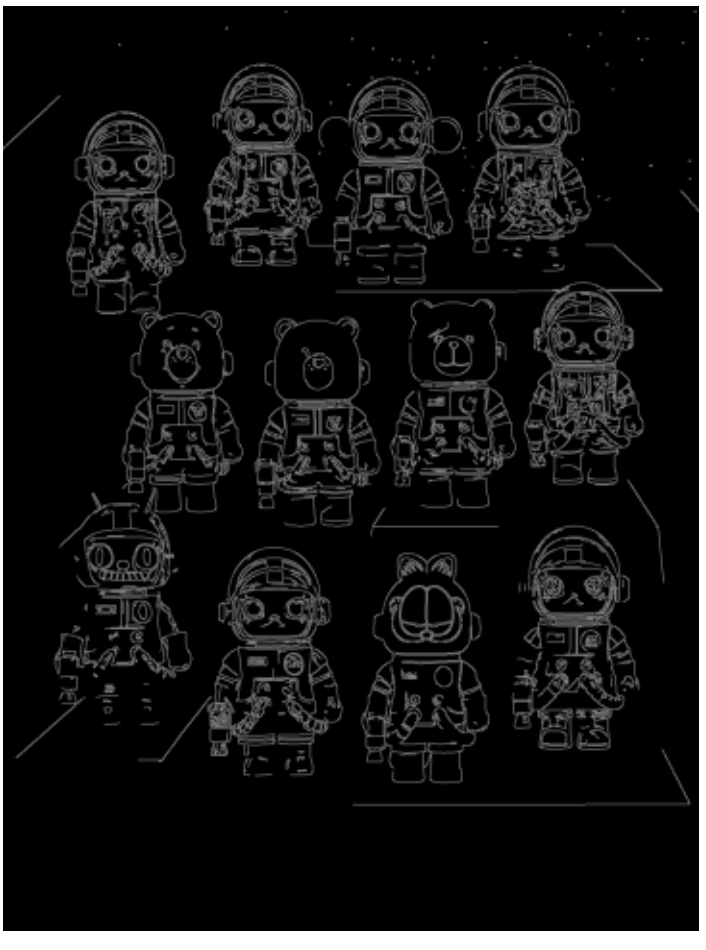
Unsharp Masking (k=1.0)

High-boost (k=1.5)

High-boost (k=2.0)

High-boost (k=2.5)

Edge Detection (Original)



Source Code (Page 5)

```
#!/usr/bin/env python3
"""
Task 4: Unsharp Masking and High-boost Filtering
Image: datasets/mega_space_molly.jpg
"""

import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from datetime import datetime

# ========================================
# □□□□□□ Unsharp Masking □□□ High-boost Filtering
# ========================================
"""
UNSHARP MASKING:
----------------
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□:
1. □□□□□□□□□□□□□□□□□□□□□□: blur = Gaussian_blur(original)
2. □□□□□□□ mask (unsharp mask): mask = original - blur
3. □□□□□□□ mask □□□□□□□□□□□□□□□□□□□□□□□□: sharpened = original + mask

□□□□□:
    g(x,y) = f(x,y) + [f(x,y) - f̄(x,y)]

    □□□□:
    - f(x,y) = □□□□□□□□□□□□□□□□
    - f̄(x,y) = □□□□□□□□□□□□□□□□
    - [f(x,y) - f̄(x,y)] = unsharp mask

HIGH-BOOST FILTERING:
---------------------
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Unsharp Masking □□□□□□□□□□□□□□□□□□□□□ k (amplification factor)

□□□□□:
    g(x,y) = k·f(x,y) - f̄(x,y)

    □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□:
    g(x,y) = (k-1)·f(x,y) + [f(x,y) - f̄(x,y)]

    □□□□:
    - k = 1: □□□□□□□□□□□ unsharp masking □□□□□
    - k > 1: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ (high-boost)
    - k < 1: □□□□□□□□□□□□□□□□□□
"""

print("="*80)
print("TASK 4: UNSHARP MASKING □□□ HIGH-BOOST FILTERING")
print("="*80)

# □□□□□□□□□□
image_path = 'datasets/mega_space_molly.jpg'
print(f"\n□ □□□□□□□□□□□□□□□□□□□□: {image_path}")

img = cv2.imread(image_path)
if img is None:
    print(f"□ Error: □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ {image_path}")
    exit(1)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
print(f"✓ □□□□□□□□□□□□□□□□□□□□□□□: {img.shape}")

# ========================================
# FUNCTION DEFINITIONS
# ========================================

def apply_unsharp_masking(image, kernel_size=5, sigma=1.0, amount=1.0):
    """
    Unsharp Masking

    Parameters:
    - image: □□□□□□□□□□□□□□□□ (RGB)
```

Source Code (Page 6)

```
    - kernel_size: □□□□ Gaussian kernel (□□□□□□□□□□□□□□□□)
    - sigma: □□□ standard deviation □□□ Gaussian
    - amount: □□□□□□□□□□□□□□□ sharpen (□□□□□□□□□□ = 1.0 □□□□□□□ unsharp masking)

    Returns:
    - sharpened: □□□□□□□□ unsharp masking □□□□
    - mask: unsharp mask
    - blurred: □□□□□□□□□□□
    """
    # □□□□□□□□ float □□□□□□□□□□□□□□
    image_float = image.astype(np.float32)

    # Step 1: □□□□□□□□□□□ Gaussian blur
    blurred = cv2.GaussianBlur(image_float, (kernel_size, kernel_size), sigma)

    # Step 2: □□□□□ unsharp mask = original - blurred
    mask = image_float - blurred

    # Step 3: □□□□□ mask □□□□□□□□□□□□□□□
    # sharpened = original + amount * mask
    sharpened = image_float + amount * mask

    # Clip □□□□□□□□□□□□□□□□□ [0, 255]
    sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

    return sharpened, mask, blurred.astype(np.uint8)


def apply_highboost_filtering(image, kernel_size=5, sigma=1.0, k=1.5):
    """
    High-boost Filtering

    Parameters:
    - image: □□□□□□□□□□ (RGB)
    - kernel_size: □□□□ Gaussian kernel
    - sigma: □□□ standard deviation □□□ Gaussian
    - k: amplification factor (k > 1 □□□□□□ high-boost)

    Returns:
    - enhanced: □□□□□□□□ high-boost filtering □□□□

    □□□□: g(x,y) = k·f(x,y) - f̄(x,y)
    """
    # □□□□□□□□ float
    image_float = image.astype(np.float32)

    # □□□□□□□□
    blurred = cv2.GaussianBlur(image_float, (kernel_size, kernel_size), sigma)

    # High-boost filtering: k*original - blurred
    enhanced = k * image_float - blurred

    # Clip □□□□□□□□□□□□□□□□□ [0, 255]
    enhanced = np.clip(enhanced, 0, 255).astype(np.uint8)

    return enhanced


# ========================================
# PROCESSING
# ========================================

print("\n" + "="*80)
print("□□□□□□□□□□□□□□□...")
print("="*80)

# Parameters
kernel_size = 5
sigma = 1.0

# 1. Unsharp Masking (amount = 1.0)
print("\n1. Unsharp Masking (amount = 1.0)...")
unsharp_result, unsharp_mask, blurred_img = apply_unsharp_masking(
    img_rgb, kernel_size=kernel_size, sigma=sigma, amount=1.0
)
```

Source Code (Page 7)

```
# 2. High-boost Filtering with different k values
k_values = [1.0, 1.5, 2.0, 2.5]
highboost_results = {}

print("\n2. High-boost Filtering:")
for k in k_values:
    print(f"   - k = {k}")
    result = apply_highboost_filtering(img_rgb, kernel_size=kernel_size, sigma=sigma, k=k)
    highboost_results[k] = result

print("\n✓ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□!")

# ========================================
# VISUALIZATION
# ========================================

print("\n" + "="*80)
print("□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□...")
print("="*80)

# Figure 1: Unsharp Masking Process
fig1, axes1 = plt.subplots(2, 2, figsize=(14, 12))
fig1.suptitle('Unsharp Masking Process', fontsize=16, fontweight='bold')

# Original
axes1[0, 0].imshow(img_rgb)
axes1[0, 0].set_title('Original Image', fontsize=12, fontweight='bold')
axes1[0, 0].axis('off')

# Blurred
axes1[0, 1].imshow(blurred_img)
axes1[0, 1].set_title(f'Blurred (Gaussian {kernel_size}x{kernel_size}, σ={sigma})', fontsize=12)
axes1[0, 1].axis('off')

# Unsharp Mask (□□□□□□□□ grayscale □□□□□□□ contrast)
mask_display = unsharp_mask.astype(np.float32)
mask_display = (mask_display - mask_display.min()) / (mask_display.max() - mask_display.min())
axes1[1, 0].imshow(mask_display)
axes1[1, 0].set_title('Unsharp Mask (Original - Blurred)', fontsize=12)
axes1[1, 0].axis('off')

# Sharpened Result
axes1[1, 1].imshow(unsharp_result)
axes1[1, 1].set_title('Sharpened (Unsharp Masking)', fontsize=12, fontweight='bold')
axes1[1, 1].axis('off')

plt.tight_layout()

# Figure 2: High-boost Filtering Comparison
fig2, axes2 = plt.subplots(2, 3, figsize=(16, 11))
fig2.suptitle('High-boost Filtering - Effect of k Parameter', fontsize=16, fontweight='bold')

# Original
axes2[0, 0].imshow(img_rgb)
axes2[0, 0].set_title('Original Image', fontsize=12, fontweight='bold')
axes2[0, 0].axis('off')

# Unsharp Masking (k=1.0 equivalent)
axes2[0, 1].imshow(unsharp_result)
axes2[0, 1].set_title('Unsharp Masking\n(k = 1.0)', fontsize=12)
axes2[0, 1].axis('off')

# High-boost k=1.5
axes2[0, 2].imshow(highboost_results[1.5])
axes2[0, 2].set_title('High-boost Filtering\n(k = 1.5)', fontsize=12, fontweight='bold')
axes2[0, 2].axis('off')

# High-boost k=2.0
axes2[1, 0].imshow(highboost_results[2.0])
axes2[1, 0].set_title('High-boost Filtering\n(k = 2.0)', fontsize=12, fontweight='bold')
axes2[1, 0].axis('off')

# High-boost k=2.5
axes2[1, 1].imshow(highboost_results[2.5])
```

Source Code (Page 8)

```
axes2[1, 1].set_title('High-boost Filtering\n(k = 2.5)', fontsize=12, fontweight='bold')
axes2[1, 1].axis('off')

# Comparison text
comparison_text = f"""
PARAMETERS:
• Kernel size: {kernel_size}x{kernel_size}
• Sigma (σ): {sigma}

OBSERVATION:
• k = 1.0: Standard unsharp masking
• k = 1.5: Moderate sharpening
• k = 2.0: Strong sharpening
• k = 2.5: Very strong sharpening

Note: Higher k values amplify
edges and details more, but may
also amplify noise.
"""
axes2[1, 2].text(0.1, 0.5, comparison_text, fontsize=10,
                 verticalalignment='center', family='monospace',
                 bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))
axes2[1, 2].axis('off')

plt.tight_layout()

# Save figures
fig1.savefig('output/task4_unsharp_masking_process.png', dpi=150, bbox_inches='tight')
fig2.savefig('output/task4_highboost_comparison.png', dpi=150, bbox_inches='tight')

print("✓ □□□□□□□□□□:")
print("  - output/task4_unsharp_masking_process.png")
print("  - output/task4_highboost_comparison.png")

# ========================================
# DETAILED COMPARISON
# ========================================

fig3, axes3 = plt.subplots(3, 2, figsize=(12, 16))
fig3.suptitle('Detailed Comparison: Original vs Enhanced', fontsize=16, fontweight='bold')

images_to_compare = [
    ("Original", img_rgb),
    ("Unsharp Masking (k=1.0)", unsharp_result),
    ("High-boost (k=1.5)", highboost_results[1.5]),
    ("High-boost (k=2.0)", highboost_results[2.0]),
    ("High-boost (k=2.5)", highboost_results[2.5]),
]

# Add edge detection comparison
edges_original = cv2.Canny(cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY), 100, 200)
images_to_compare.append(("Edge Detection (Original)", edges_original))

for idx, (title, image) in enumerate(images_to_compare):
    row = idx // 2
    col = idx % 2
    if len(image.shape) == 2:  # Grayscale
        axes3[row, col].imshow(image, cmap='gray')
    else:
        axes3[row, col].imshow(image)
    axes3[row, col].set_title(title, fontsize=11, fontweight='bold')
    axes3[row, col].axis('off')

plt.tight_layout()
fig3.savefig('output/task4_detailed_comparison.png', dpi=150, bbox_inches='tight')

print("  - output/task4_detailed_comparison.png")

# ========================================
# CREATE PDF REPORT
# ========================================

print("\n" + "="*80)
print("□□□□□□□□□□ PDF Report...")
print("="*80)
```

Source Code (Page 9)

```
pdf_filename = 'output/Task4_Unsharp_Highboost_Report.pdf'

with PdfPages(pdf_filename) as pdf:

    # Page 1: Title and Theory
    fig_title = plt.figure(figsize=(8.5, 11))
    fig_title.text(0.5, 0.9, 'Task 4: Unsharp Masking □□□ High-boost Filtering',
                   ha='center', fontsize=18, fontweight='bold')
    fig_title.text(0.5, 0.85, f'Image: {image_path}',
                   ha='center', fontsize=12)
    fig_title.text(0.5, 0.82, f'Generated: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}',
                   ha='center', fontsize=10, style='italic')

    theory_text = """
    □□□□□□ UNSHARP MASKING:

    Unsharp masking □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□

    □□□□□□□:
    1. □□□□□□□□□□□□□□□□: blur = Gaussian_blur(original)
    2. □□□□□□ mask: mask = original - blur
    3. □□□□□□ mask □□□□□□□□□□□□□□□□□□□□: sharpened = original + mask

    □□□□:
        g(x,y) = f(x,y) + [f(x,y) - f¯(x,y)]

    □□□:
        f(x,y) = □□□□□□□□□□
        f¯(x,y) = □□□□□□□□□□
        [f(x,y) - f¯(x,y)] = unsharp mask

    ─────────────────────────────────────────────────────────────────

    □□□□□□ HIGH-BOOST FILTERING:

    High-boost filtering □□□□□□□□□□□□□□□□□□□□ Unsharp Masking □□□□□□□□□□□□□□□
    □□□ k (amplification factor)

    □□□□:
        g(x,y) = k·f(x,y) - f¯(x,y)

    □□□□□□□□□□□□□□□□□□:
        g(x,y) = (k-1)·f(x,y) + [f(x,y) - f¯(x,y)]

    □□□:
        k = 1.0: □□□□□□□ unsharp masking □□□□
        k > 1.0: □□□□□□□□□□□□□□□□□□□□□□□□ (high-boost)
        k < 1.0: □□□□□□□□□□□

    ─────────────────────────────────────────────────────────────────

    PARAMETERS □□□□□□□□□□:
        • Kernel size: 5×5
        • Sigma (σ): 1.0
        • k values: 1.0, 1.5, 2.0, 2.5
    """

    fig_title.text(0.1, 0.7, theory_text, fontsize=9, family='monospace',
                   verticalalignment='top',
                   bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.3))

    fig_title.text(0.5, 0.05, 'Page 1', ha='center', fontsize=8)
    plt.axis('off')
    pdf.savefig(fig_title, bbox_inches='tight')
    plt.close()

    # Page 2: Unsharp Masking Process
    pdf.savefig(fig1, bbox_inches='tight')
    plt.close(fig1)

    # Page 3: High-boost Comparison
    pdf.savefig(fig2, bbox_inches='tight')
    plt.close(fig2)
```

Source Code (Page 10)

```
    # Page 4: Detailed Comparison
    pdf.savefig(fig3, bbox_inches='tight')
    plt.close(fig3)

    # Page 5-6: Source Code
    fig_code = plt.figure(figsize=(8.5, 11))

    # Read this file's source code
    with open(__file__, 'r', encoding='utf-8') as f:
        source_code = f.read()

    # Split code into pages if too long
    lines_per_page = 75
    code_lines = source_code.split('\n')

    page_num = 5
    for i in range(0, len(code_lines), lines_per_page):
        fig_code_page = plt.figure(figsize=(8.5, 11))
        code_chunk = '\n'.join(code_lines[i:i+lines_per_page])

        fig_code_page.text(0.5, 0.98, f'Source Code (Page {page_num})',
                           ha='center', fontsize=14, fontweight='bold')
        fig_code_page.text(0.05, 0.95, code_chunk, fontsize=6, family='monospace',
                           verticalalignment='top', wrap=True)
        fig_code_page.text(0.5, 0.02, f'Page {page_num}', ha='center', fontsize=8)
        plt.axis('off')
        pdf.savefig(fig_code_page, bbox_inches='tight')
        plt.close(fig_code_page)
        page_num += 1

    # PDF metadata
    d = pdf.infodict()
    d['Title'] = 'Task 4: Unsharp Masking and High-boost Filtering'
    d['Author'] = 'Image Processing Course'
    d['Subject'] = 'Unsharp Masking, High-boost Filtering'
    d['Keywords'] = 'Image Enhancement, Sharpening, Unsharp Masking'
    d['CreationDate'] = datetime.now()

print(f"\n✓ □□□□□□ PDF □□□□□□□: {pdf_filename}")

# ========================================
# SUMMARY
# ========================================

print("\n" + "="*80)
print("□□□□□□□□□□□□□□□□□□")
print("="*80)

print(f"""
□□□□□□□□□□□: {image_path}
□□□□□□□□: {img.shape[1]} × {img.shape[0]} pixels

□□□□□□□□□□□□□□□:
1. Unsharp Masking (amount = 1.0)
2. High-boost Filtering (k = 1.0, 1.5, 2.0, 2.5)

Parameters:
- Gaussian kernel size: {kernel_size}×{kernel_size}
- Sigma (σ): {sigma}

□□□□□□□□:
✓ Unsharp masking □□□□□□□□□ sharpen □□□□□□□□□□□□
✓ High-boost filtering (k > 1) □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ k
✓ □□□ k □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ artifacts □□□ amplify noise

□□□□□□□□□□□□□□□:
□ {pdf_filename}
□ output/task4_unsharp_masking_process.png
□ output/task4_highboost_comparison.png
□ output/task4_detailed_comparison.png
""")

print("="*80)
print("□ Task 4 □□□□□□□□□□□□□□□!")
```

Source Code (Page 11)

```
print("="*80)

# Show plots
plt.show()
```