

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

**ΘΕΜΑΤΑ ΟΡΑΣΗΣ ΥΠΟΛΟΓΙΣΤΩΝ & ΓΡΑΦΙΚΗΣ**

**ΑΝΑΦΟΡΑ ΒΑΣΙΣΜΕΝΗ ΣΤΗΝ**

**1Η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

ΑΓΓΕΛΟΣ ΝΙΚΟΛΑΟΣ ΠΟΤΑΜΙΑΝΟΣ

Α.Μ. 1084537

up1084537@ac.upatras.gr

Πάτρα, 2024

Προς διευκόλυνση της διαδικασίας της διόρθωσης, μπορείτε να βρείτε τον κώδικα ανεβασμένο στο [GitHub](https://github.com/nickpotamianos/Computer_Vision_Lab/tree/main/CV_1-PYRAMIDS), καθώς και τον κώδικα συμπυκνωμένο σε .ipynb μορφή με τα αποτελέσματα τυπωμένα πάνω στους τίτλους «ΔΙΑΔΙΚΑΣΙΑ Χ.» όταν υπάρχει το αντίστοιχο αρχείο[**ΔΙΑΔΙΚΑΣΙΑ 1.**](https://github.com/nickpotamianos/Computer_Vision_Lab/blob/main/CV_1-PYRAMIDS/exercise1.ipynb)

**Υλοποίηση - Περιγραφή Κώδικα**

Η υλοποίηση της προτεινόμενης μεθόδου πραγματοποιήθηκε σε γλώσσα Python, χρησιμοποιώντας τις βιβλιοθήκες του OpenCV, NumPy και Matplotlib. Ο κώδικας περιλαμβάνει:

1. **Δημιουργία Γκαουσιανού Πυρήνα:** Ένας πυρήνας με διωνυμικούς συντελεστές εφαρμόζεται ως χαμηλοπερατό φίλτρο.
2. **Εφαρμογή Γκαουσιανού Φίλτρου:** Το φίλτρο εφαρμόζεται σε κάθε κανάλι της εικόνας μέσω διαχωριστικού φιλτραρίσματος.
3. **Δημιουργία Γκαουσιανής Πυραμίδας:** Η εικόνα υποδειγματοληπτείται σε κάθε επίπεδο μετά την εφαρμογή του φίλτρου, δημιουργώντας μια σειρά επιπέδων με μειωμένη ανάλυση.
4. **Δημιουργία Λαπλασιανής Πυραμίδας:** Υπολογίζονται οι διαφορές μεταξύ επιπέδων της Γκαουσιανής πυραμίδας για την παραγωγή της Λαπλασιανής πυραμίδας.
5. **Δημιουργία Μάσκας και Πυραμίδας Μάσκας:** Μια δυαδική μάσκα διαιρεί την εικόνα κάθετα, η οποία υποδιαιρείται σε πυραμίδα Γκαουσιανών για ομαλή μεταβατικότητα.
6. **Συρραφή Πυραμίδων:** Οι Λαπλασιανές πυραμίδες των δύο εικόνων συνδυάζονται με βάση την πυραμίδα της μάσκας.
7. **Ανακατασκευή Τελικής Εικόνας:** Η συνδυασμένη πυραμίδα ανασυντίθεται για την παραγωγή της τελικής, συρραφείσας εικόνας.
8. **Οπτικοποίηση Αποτελεσμάτων:** Οι πυραμίδες και η τελική εικόνα εμφανίζονται γραφικά για ανάλυση και αξιολόγηση.

**Αποτελέσματα**

Η εφαρμογή της μεθόδου στις εικόνες apple.jpg και orange.jpg απέδωσε την επιθυμητή συρραφή σε μωσαϊκό. Συγκεκριμένα:

* **Γκαουσιανές Πυραμίδες:** Οι πυραμίδες κάθε εικόνας παρουσίασαν επιμέρους επίπεδα με μειούμενη ανάλυση, διατηρώντας τις χαμηλές συχνότητες και αποφεύγοντας την απώλεια βασικών δομών.
* **Λαπλασιανές Πυραμίδες:** Τα επίπεδα των Λαπλασιανών πυραμίδων ανέδειξαν τις λεπτομέρειες και τα υψηλά συχνότητες των εικόνων, επιτρέποντας την ομαλή συνένωση τους.
* **Μορφή Συρραφής:** Η τελική εικόνα συρραφής (δεξιά), η οποία αποτελεί την ανακατασκευασμένη εικόνα, αποδείχθηκε φυσική και αόρατη στη διάκριση των δύο αρχικών εικόνων.

**exercise1.py:**

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
def create\_gaussian\_kernel():  
 *"""  
 Create the Gaussian kernel h = [1 4 6 4 1]/16   
 """* return np.array([1, 4, 6, 4, 1]) / 16.0  
  
def apply\_gaussian\_filter(image):  
 *"""  
 Apply Gaussian filtering using the specified kernel  
 """* kernel = create\_gaussian\_kernel()  
 if len(image.shape) == 3:  
 filtered = np.zeros\_like(image)  
 for i in range(3):  
 filtered[:,:,i] = cv2.sepFilter2D(image[:,:,i], -1, kernel, kernel)  
 return filtered  
 return cv2.sepFilter2D(image, -1, kernel, kernel)  
  
def generate\_gaussian\_pyramid(image, levels):  
 *"""  
 Generate Gaussian pyramid with L+1 levels  
 """* pyramid = [image]  
 current = image.copy()  
 for \_ in range(levels):  
 filtered = apply\_gaussian\_filter(current)  
 current = filtered[::2, ::2]  
 pyramid.append(current)  
 return pyramid  
  
def generate\_laplacian\_pyramid(image, levels):  
 *"""  
 Generate Laplacian pyramid with L+1 levels  
 """* gaussian\_pyramid = generate\_gaussian\_pyramid(image, levels)  
 pyramid = []  
   
 for i in range(levels):  
 current = gaussian\_pyramid[i]  
 next\_level = gaussian\_pyramid[i + 1]  
 expanded = cv2.resize(next\_level, (current.shape[1], current.shape[0]))  
 laplacian = current - expanded  
 pyramid.append(laplacian)  
   
 pyramid.append(gaussian\_pyramid[-1])  
 return pyramid  
  
def create\_mask(shape):  
 *"""  
 Create a binary mask dividing the image vertically  
 """* mask = np.zeros(shape[:2])  
 mask[:, :shape[1]//2] = 1  
 return mask  
  
def blend\_pyramids(pyr1, pyr2, mask\_pyr):  
 *"""  
 Blend pyramids according to equation (4)   
 """* blended\_pyr = []  
   
 for i in range(len(pyr1)-1):  
 mask = np.expand\_dims(mask\_pyr[i], axis=2) if len(pyr1[i].shape) == 3 else mask\_pyr[i]  
 blended = mask \* pyr1[i] + (1 - mask) \* pyr2[i]  
 blended\_pyr.append(blended)  
   
 mask = np.expand\_dims(mask\_pyr[-1], axis=2) if len(pyr1[-1].shape) == 3 else mask\_pyr[-1]  
 last\_level = mask \* pyr1[-1] + (1 - mask) \* pyr2[-1]  
 blended\_pyr.append(last\_level)  
   
 return blended\_pyr  
  
def reconstruct\_from\_pyramid(pyramid):  
 *"""  
 Reconstruct image from pyramid  
 """* current = pyramid[-1]  
   
 for level in reversed(pyramid[:-1]):  
 expanded = cv2.resize(current, (level.shape[1], level.shape[0]))  
 current = expanded + level  
   
 return current  
  
def display\_pyramid(pyramid, title):  
 *"""  
 Display all levels of a pyramid  
 """* plt.figure(figsize=(20, 4))  
 for i, level in enumerate(pyramid):  
 plt.subplot(1, len(pyramid), i+1)  
 if len(level.shape) == 3:  
 plt.imshow(np.clip(level, 0, 1))  
 else:  
 plt.imshow(level, cmap='gray')  
 plt.title(f'Level {i}')  
 plt.axis('off')  
 plt.suptitle(title)  
 plt.show()  
  
def create\_direct\_blend(img1, img2, mask):  
 *"""  
 Create direct blend without pyramid blending  
 """* mask = np.expand\_dims(mask, axis=2)  
 return img1 \* mask + img2 \* (1 - mask)  
  
  
def main():  
 # Read images  
 try:  
 apple = cv2.imread('photos/apple.jpg')  
 orange = cv2.imread('photos/orange.jpg')  
   
 if apple is None or orange is None:  
 raise FileNotFoundError("Could not read one or both images")  
   
 except Exception as e:  
 print(f"Error loading images: {e}")  
 return  
  
 # Convert from BGR to RGB and normalize  
 apple = cv2.cvtColor(apple, cv2.COLOR\_BGR2RGB).astype(float) / 255  
 orange = cv2.cvtColor(orange, cv2.COLOR\_BGR2RGB).astype(float) / 255  
  
 # Verify images are the same size  
 if apple.shape != orange.shape:  
 print(f"Images must be the same size. Got apple: {apple.shape}, orange: {orange.shape}")  
 return  
  
 # Number of pyramid levels  
 L = 4  
   
 # Create mask and its Gaussian pyramid  
 mask = create\_mask(apple.shape)  
 mask\_pyramid = generate\_gaussian\_pyramid(mask, L)  
   
 # Generate Gaussian pyramids for both images  
 apple\_gaussian\_pyramid = generate\_gaussian\_pyramid(apple, L)  
 orange\_gaussian\_pyramid = generate\_gaussian\_pyramid(orange, L)  
  
 # Display Gaussian pyramids  
 display\_pyramid(apple\_gaussian\_pyramid, 'Apple Gaussian Pyramid')  
 display\_pyramid(orange\_gaussian\_pyramid, 'Orange Gaussian Pyramid')  
  
 # Generate Laplacian pyramids for both images  
 apple\_pyramid = generate\_laplacian\_pyramid(apple, L)  
 orange\_pyramid = generate\_laplacian\_pyramid(orange, L)  
  
 # Display Laplacian pyramids  
 display\_pyramid(apple\_pyramid, 'Apple Laplacian Pyramid')  
 display\_pyramid(orange\_pyramid, 'Orange Laplacian Pyramid')  
 display\_pyramid(mask\_pyramid, 'Mask Gaussian Pyramid')  
  
 # Blend pyramids  
 blended\_pyramid = blend\_pyramids(apple\_pyramid, orange\_pyramid, mask\_pyramid)  
 display\_pyramid(blended\_pyramid, 'Blended Pyramid')  
  
 # Reconstruct the blended image  
 pyramid\_blend = reconstruct\_from\_pyramid(blended\_pyramid)  
  
 # Clip values to [0, 1] range  
 pyramid\_blend = np.clip(pyramid\_blend, 0, 1)  
  
 # Display final results  
 plt.figure(figsize=(12, 5))  
  
 plt.subplot(121)  
 plt.imshow(create\_direct\_blend(apple, orange, mask))  
 plt.title('Without Blending')  
 plt.axis('off')  
  
 plt.subplot(122)  
 plt.imshow(pyramid\_blend)  
 plt.title('With Pyramid Blending')  
 plt.axis('off')  
  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

A close up of an apple

Description automatically generated A close up of an orange

Description automatically generated A screenshot of a computer screen

Description automatically generated A black square with white text

Description automatically generated A black rectangles with text

Description automatically generated A black square with text

Description automatically generated with medium confidence A comparison of an orange and a pencil

Description automatically generated

[**ΔΙΑΔΙΚΑΣΙΑ 2.**](https://github.com/nickpotamianos/Computer_Vision_Lab/blob/main/CV_1-PYRAMIDS/exercise2.ipynb)

**ΥΛΟΠΟΙΗΣΗ:  
Δημιουργία Γκαουσιανού Πυρήνα:**

**def** **create\_gaussian\_kernel**():  
 """Create the Gaussian kernel h = [1 4 6 4 1]/16"""  
 **return** np.array([1, 4, 6, 4, 1]) / 16.0

Ο πυρήνας αυτός χρησιμοποιείται για το φιλτράρισμα της εικόνας σε κάθε επίπεδο της Γκαουσιανής πυραμίδας.

**Εφαρμογή Γκαουσιανού Φίλτρου:**

def apply\_gaussian\_filter(image):  
 """Apply Gaussian filtering using the specified kernel"""  
 kernel = create\_gaussian\_kernel()  
 **if** len(image.shape) == 3:  
 filtered = np.zeros\_like(image)  
 for i **in** range(3):  
 filtered[:,:,i] = cv2.sepFilter2D(image[:,:,i], -1, kernel, kernel)  
 return filtered  
 return cv2.sepFilter2D(image, -1, kernel, kernel)

Η συνάρτηση αυτή εφαρμόζει το Γκαουσιανό φίλτρο σε κάθε κανάλι της εικόνας.

**Δημιουργία Γκαουσιανής Πυραμίδας:**

**def** **generate\_gaussian\_pyramid**(image, levels):  
 """Generate Gaussian pyramid with L+1 levels"""  
 pyramid = [image]  
 current = image.copy()  
 **for** \_ **in** range(levels):  
 filtered = apply\_gaussian\_filter(current)  
 current = filtered[::2, ::2]  
 pyramid.append(current)  
 **return** pyramid

Η συνάρτηση αυτή δημιουργεί μια Γκαουσιανή πυραμίδα με καθορισμένο αριθμό επιπέδων.

**Δημιουργία Λαπλασιανής Πυραμίδας:**

**def** **generate\_laplacian\_pyramid**(image, levels):  
 """Generate Laplacian pyramid with L+1 levels"""  
 gaussian\_pyramid = generate\_gaussian\_pyramid(image, levels)  
 pyramid = []  
   
 **for** i **in** range(levels):  
 current = gaussian\_pyramid[i]  
 next\_level = gaussian\_pyramid[i + 1]  
 expanded = cv2.resize(next\_level, (current.shape[1], current.shape[0]))  
 laplacian = current - expanded  
 pyramid.append(laplacian)  
   
 pyramid.append(gaussian\_pyramid[-1])  
 **return** pyramid

Αυτή η συνάρτηση υπολογίζει τις διαφορές μεταξύ διαδοχικών επιπέδων της Γκαουσιανής πυραμίδας για τη δημιουργία της Λαπλασιανής πυραμίδας.

**Δημιουργία Μάσκας:**

**def** **create\_eye\_mask**(shape, center, radius):  
 """Create a circular mask for the eye region"""  
 mask = np.zeros(shape[:2])  
 y, x = np.ogrid[:shape[0], :shape[1]]  
 dist\_from\_center = np.sqrt((x - center[0])\*\*2 + (y - center[1])\*\*2)  
 mask[dist\_from\_center <= radius] = 1  
   
 # Smooth the mask edges  
 mask = cv2.GaussianBlur(mask, (21, 21), 10)  
 **return** mask

Η μάσκα δημιουργείται ως κυκλική με κέντρο και ακτίνα που καθορίζουν την περιοχή σύνδεσης των εικόνων, με θόλωση στα άκρα για ομαλή μετάβαση.

**Συρραφή Πυραμίδων:**

**def** **blend\_pyramids**(pyr1, pyr2, mask\_pyr):  
 """Blend pyramids according to mask"""  
 blended\_pyr = []  
   
 **for** i **in** range(len(pyr1)-1):  
 mask = np.expand\_dims(mask\_pyr[i], axis=2) **if** len(pyr1[i].shape) == 3 **else** mask\_pyr[i]  
 blended = mask \* pyr1[i] + (1 - mask) \* pyr2[i]  
 blended\_pyr.append(blended)  
   
 mask = np.expand\_dims(mask\_pyr[-1], axis=2) **if** len(pyr1[-1].shape) == 3 **else** mask\_pyr[-1]  
 last\_level = mask \* pyr1[-1] + (1 - mask) \* pyr2[-1]  
 blended\_pyr.append(last\_level)  
   
 **return** blended\_pyr

Η συνάρτηση αυτή συνδυάζει τις Λαπλασιανές πυραμίδες των δύο εικόνων βάσει της πυραμίδας της μάσκας.

**Ανακατασκευή Τελικής Εικόνας:**

**def** **reconstruct\_from\_pyramid**(pyramid):  
 """Reconstruct image from pyramid"""  
 current = pyramid[-1]  
   
 **for** level **in** reversed(pyramid[:-1]):  
 expanded = cv2.resize(current, (level.shape[1], level.shape[0]))  
 current = expanded + level  
   
 **return** current

Η τελική εικόνα ανασυντίθεται προσθέτοντας τα επίπεδα της συνδυασμένης πυραμίδας.

**Οπτικοποίηση Αποτελεσμάτων:**

**def** **display\_pyramid**(pyramid, title):  
 """Display all levels of a pyramid"""  
 plt.figure(figsize=(20, 4))  
 **for** i, level **in** enumerate(pyramid):  
 plt.subplot(1, len(pyramid), i+1)  
 **if** len(level.shape) == 3:  
 plt.imshow(np.clip(level, 0, 1))  
 **else**:  
 plt.imshow(level, cmap='gray')  
 plt.title(f'Level {i}')  
 plt.axis('off')  
 plt.suptitle(title)  
 plt.show()

Η συνάρτηση αυτή επιτρέπει την οπτικοποίηση των επιπέδων των πυραμίδων για ανάλυση και αξιολόγηση.

Εφαρμογή της Μεθόδου στις εικόνες

Για την παρούσα εργασία, χρησιμοποιήθηκαν οι εικόνες woman.png και hand.png. Η διαδικασία ακολούθησε τα εξής βήματα:

Φόρτωση και Προεπεξεργασία των Εικόνων: Οι εικόνες φορτώθηκαν ως εικόνες σε αποχρώσεις του γκρι και κανονικοποιήθηκαν στο εύρος [0, 1]. Διασφαλίστηκε ότι οι εικόνες έχουν το ίδιο μέγεθος μέσω της αναδιαστασιοποίησης της εικόνας woman.png ώστε να ταιριάζει με τις διαστάσεις της hand.png.

Δημιουργία Μάσκας: Χρησιμοποιήθηκε η συνάρτηση create\_eye\_mask για τη δημιουργία μιας κυκλικής μάσκας που διαχωρίζει τις δύο εικόνες. Η μάσκα σχεδιάστηκε με κέντρο στο κέντρο της εικόνας και ακτίνα που αντιστοιχεί στο ένα έκτο του μικρότερου διαστάσεων της εικόνας, εξασφαλίζοντας έτσι ότι η περιοχή σύνδεσης βρίσκεται στο κέντρο της εικόνας.

Δημιουργία Γκαουσιανών και Λαπλασιανών Πυραμιτών: Δημιουργήθηκαν οι Γκαουσιανές πυραμίδες για τις δύο εικόνες καθώς και η μάσκα, χρησιμοποιώντας τη συνάρτηση generate\_gaussian\_pyramid. Στη συνέχεια, παράγονται οι Λαπλασιανές πυραμίδες για τις εικόνες woman.png και hand.png μέσω της συνάρτησης generate\_laplacian\_pyramid.

Συρραφή Πυραμιτών: Οι Λαπλασιανές πυραμίδες των δύο εικόνων συνδυάστηκαν με βάση την πυραμίδα της μάσκας χρησιμοποιώντας τη συνάρτηση blend\_pyramids. Αυτή η διαδικασία εξασφαλίζει ότι σε κάθε κλίμακα, η συμβολή των δύο εικόνων ελέγχεται από τη μάσκα, επιτυγχάνοντας έτσι μια ομαλή μετάβαση.

Ανακατασκευή της Τελικής Εικόνας: Η τελική, συρραφείσα εικόνα ανακατασκευάστηκε από την συνδυασμένη πυραμίδα χρησιμοποιώντας τη συνάρτηση reconstruct\_from\_pyramid. Η ανακατασκευή πραγματοποιήθηκε μέσω της επαναληπτικής επέκτασης και πρόσθεσης των επιπέδων της συνδυασμένης πυραμίδας, αποκαθιστώντας την πολυκλιμακωτή πληροφορία της αρχικής εικόνας.

Οπτικοποίηση Αποτελεσμάτων: Οι πυραμίδες και η τελική εικόνα εμφανίστηκαν γραφικά για ανάλυση και αξιολόγηση. Οι Γκαουσιανές πυραμίδες κάθε εικόνας παρουσίασαν επιμέρους επίπεδα με μειούμενη ανάλυση, ενώ οι Λαπλασιανές πυραμίδες ανέδειξαν τις λεπτομέρειες και τα υψηλά συχνότητες των εικόνων.

Αποτελέσματα

Η εφαρμογή της προτεινόμενης μεθόδου στις εικόνες woman.png και hand.png απέδωσε το επιθυμητό αποτέλεσμα της συρραφής, όπως φαίνεται στο σχήμα. Συγκεκριμένα:

A close-up of a person's face

Description automatically generated

Γκαουσιανές Πυραμίδες: Οι πυραμίδες κάθε εικόνας παρουσίασαν μια σειρά επιπέδων με μειωμένη ανάλυση, διατηρώντας τις χαμηλές συχνότητες και αποφεύγοντας την απώλεια βασικών δομών της εικόνας.

Λαπλασιανές Πυραμίδες: Τα επίπεδα των Λαπλασιανών πυραμίδων ανέδειξαν τις λεπτομέρειες και τις υψηλές συχνότητες των εικόνων, επιτρέποντας την ομαλή συνένωση τους.

Μορφή Συρραφής: Το τελικό αποτέλεσμα της συρραφής, όπως φαίνεται στο σχήμα, αποδεικνύει ότι η μάσκα και η διαδικασία συρραφής έχουν λειτουργήσει αποτελεσματικά, δημιουργώντας μια ομαλή και φυσική ένωση μεταξύ των δύο εικόνων. Η περιορισμένη ορατότητα της ένωσης υποδεικνύει την επιτυχία της μεθόδου στην αποφυγή φωτομετρικών παραμορφώσεων και στη διατήρηση της συνολικής ποιότητας της εικόνας.

**Ανάλυση και Δικαιολόγηση Επιλογών**

**Επιλογή Μάσκας**

Η επιλογή της κυκλικής μάσκας βασίστηκε στην ανάγκη για μια συμμετρική και ομοιόμορφη περιοχή μετάβασης μεταξύ των δύο εικόνων. Η χρήση Gaussian θόλωσης στις άκρες της μάσκας εξασφαλίζει ότι η μετάβαση μεταξύ των εικόνων είναι απαλή, αποφεύγοντας τα οξύτατα σημεία ένωση που θα μπορούσαν να αποκαλύψουν την αρχική διαχωρισμό των εικόνων. Η ακτίνα της μάσκας καθορίστηκε ώστε να καλύπτει σημαντική περιοχή της εικόνας, διασφαλίζοντας την ομαλή σύνδεση των δύο εικόνων στο κέντρο.

**Αριθμός Επιπέδων Πυραμίδας**

Η επιλογή του αριθμού των επιπέδων πυραμίδας (L = 4) καθορίστηκε ώστε να επιτευχθεί μια ισορροπία μεταξύ της πολυπλοκότητας του υπολογισμού και της ποιότητας της συρραφής. Τέσσερα επίπεδα πυραμίδας επιτρέπουν την επαρκή αναπαράσταση τόσο των χαμηλών όσο και των υψηλών συχνοτήτων, διασφαλίζοντας την ομαλή ενσωμάτωση των εικόνων. Αυτός ο αριθμός επιπέδων επιτρέπει την αναπαράσταση των βασικών δομών της εικόνας σε αρκετές κλίμακες, ενώ ταυτόχρονα διατηρεί την υπολογιστική αποδοτικότητα της διαδικασίας.

Χρήση Γκαουσιανών και Λαπλασιανών Πυραμιτών

Η απόφαση για τη χρήση Γκαουσιανών και Λαπλασιανών πυραμιτών απορρέει από την ανάγκη για πολυκλιμακωτή ανάλυση της εικόνας, επιτρέποντας την αποθήκευση και την επεξεργασία διαφορετικών κλιμάκων λεπτομέρειας. Η Γκαουσιανή πυραμίδα μειώνει την ανάλυση της εικόνας σε κάθε επίπεδο, ενώ η Λαπλασιανή πυραμίδα διατηρεί τις διαφορές μεταξύ των κλιμάκων, αναδεικνύοντας τις υψηλές συχνότητες και τις λεπτομέρειες. Αυτό επιτρέπει την αποτελεσματική συνένωση των εικόνων με διατήρηση της ποιότητας και της λεπτομέρειας.

Βελτίωση της Μορφής της Ένωσης

Η εφαρμογή θολώσεως στην μάσκα είναι κρίσιμη για την αποφυγή οξύτητας στα σημεία σύνδεσης των εικόνων. Η Gaussian θόλωση εξασφαλίζει ότι η μετάβαση είναι ομαλή και αόρατη, αποτρέποντας την εμφάνιση φαινομένων όπως οι φωτομετρικές παραμορφώσεις που μπορούν να αποκαλύψουν την αρχική διαχωριστική γραμμή των εικόνων.

**exercise2.py:**

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
def create\_gaussian\_kernel():  
 *"""Create the Gaussian kernel h = [1 4 6 4 1]/16"""* return np.array([1, 4, 6, 4, 1]) / 16.0  
  
def apply\_gaussian\_filter(image):  
 *"""Apply Gaussian filtering using the specified kernel"""* kernel = create\_gaussian\_kernel()  
 if len(image.shape) == 3:  
 filtered = np.zeros\_like(image)  
 for i in range(3):  
 filtered[:,:,i] = cv2.sepFilter2D(image[:,:,i], -1, kernel, kernel)  
 return filtered  
 return cv2.sepFilter2D(image, -1, kernel, kernel)  
  
def generate\_gaussian\_pyramid(image, levels):  
 *"""Generate Gaussian pyramid with L+1 levels"""* pyramid = [image]  
 current = image.copy()  
 for \_ in range(levels):  
 filtered = apply\_gaussian\_filter(current)  
 current = filtered[::2, ::2]  
 pyramid.append(current)  
 return pyramid  
  
def generate\_laplacian\_pyramid(image, levels):  
 *"""Generate Laplacian pyramid with L+1 levels"""* gaussian\_pyramid = generate\_gaussian\_pyramid(image, levels)  
 pyramid = []  
   
 for i in range(levels):  
 current = gaussian\_pyramid[i]  
 next\_level = gaussian\_pyramid[i + 1]  
 expanded = cv2.resize(next\_level, (current.shape[1], current.shape[0]))  
 laplacian = current - expanded  
 pyramid.append(laplacian)  
   
 pyramid.append(gaussian\_pyramid[-1])  
 return pyramid  
  
def create\_eye\_mask(shape, center, radius):  
 *"""Create a circular mask for the eye region"""* mask = np.zeros(shape[:2])  
 y, x = np.ogrid[:shape[0], :shape[1]]  
 dist\_from\_center = np.sqrt((x - center[0])\*\*2 + (y - center[1])\*\*2)  
 mask[dist\_from\_center <= radius] = 1  
   
 # Smooth the mask edges  
 mask = cv2.GaussianBlur(mask, (21, 21), 10)  
 return mask  
  
def blend\_pyramids(pyr1, pyr2, mask\_pyr):  
 *"""Blend pyramids according to mask"""* blended\_pyr = []  
   
 for i in range(len(pyr1)-1):  
 mask = np.expand\_dims(mask\_pyr[i], axis=2) if len(pyr1[i].shape) == 3 else mask\_pyr[i]  
 blended = mask \* pyr1[i] + (1 - mask) \* pyr2[i]  
 blended\_pyr.append(blended)  
   
 mask = np.expand\_dims(mask\_pyr[-1], axis=2) if len(pyr1[-1].shape) == 3 else mask\_pyr[-1]  
 last\_level = mask \* pyr1[-1] + (1 - mask) \* pyr2[-1]  
 blended\_pyr.append(last\_level)  
   
 return blended\_pyr  
  
def reconstruct\_from\_pyramid(pyramid):  
 *"""Reconstruct image from pyramid"""* current = pyramid[-1]  
   
 for level in reversed(pyramid[:-1]):  
 expanded = cv2.resize(current, (level.shape[1], level.shape[0]))  
 current = expanded + level  
   
 return current  
  
def main():  
 # Read images  
 hand = cv2.imread('photos/hand.png', cv2.IMREAD\_GRAYSCALE).astype(float) / 255  
 face = cv2.imread('photos/woman.png', cv2.IMREAD\_GRAYSCALE).astype(float) / 255  
   
 # Ensure images are the same size  
 if hand.shape != face.shape:  
 face = cv2.resize(face, (hand.shape[1], hand.shape[0]))  
   
 # Create eye mask - adjust these coordinates based on your images  
 eye\_center = (hand.shape[1]//2, hand.shape[0]//2) # Center of the image  
 eye\_radius = min(hand.shape) // 6 # Adjust radius as needed  
 mask = create\_eye\_mask(hand.shape, eye\_center, eye\_radius)  
   
 # Number of pyramid levels  
 L = 4  
   
 # Generate pyramids  
 mask\_pyramid = generate\_gaussian\_pyramid(mask, L)  
 hand\_pyramid = generate\_laplacian\_pyramid(hand, L)  
 face\_pyramid = generate\_laplacian\_pyramid(face, L)  
   
 # Blend pyramids  
 blended\_pyramid = blend\_pyramids(face\_pyramid, hand\_pyramid, mask\_pyramid)  
   
 # Reconstruct final image  
 result = reconstruct\_from\_pyramid(blended\_pyramid)  
 result = np.clip(result, 0, 1)  
   
 # Display results  
 plt.figure(figsize=(15, 5))  
   
 plt.subplot(141)  
 plt.imshow(hand, cmap='gray')  
 plt.title('Hand Image')  
 plt.axis('off')  
   
 plt.subplot(142)  
 plt.imshow(face, cmap='gray')  
 plt.title('Face Image')  
 plt.axis('off')  
   
 plt.subplot(143)  
 plt.imshow(mask, cmap='gray')  
 plt.title('Mask')  
 plt.axis('off')  
   
 plt.subplot(144)  
 plt.imshow(result, cmap='gray')  
 plt.title('Blended Result')  
 plt.axis('off')  
   
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

[**ΔΙΑΔΙΚΑΣΙΑ 3.**](https://github.com/nickpotamianos/Computer_Vision_Lab/blob/main/CV_1-PYRAMIDS/exercise3.ipynb)

**1. Ορισμός Μασκών**

Καθορίσαμε έξι μάσκες , k=1,2,…,6 βάσει των περιοχών όπου κάθε εικόνα θα εμφανίζεται στην τελική σύνθεση. Οι μάσκες ορίστηκαν έτσι ώστε:

* **Ιδιότητα 1:** Σε κάθε σημείο n, το άθροισμα των μασκών ισούται με 1:
* **Ιδιότητα 2:** Η μάσκα είναι 1 μόνο στα σημεία όπου η τελική εικόνα I(n) ισούται με την εικόνα :

**def** **create\_masks**(image\_shape, regions):  
 """  
 Create masks following properties 1 and 2:  
 1. Sum of all masks = 1 everywhere  
 2. mk(n) = 1 where image k should be visible  
 """  
 masks = {}  
 total\_mask = np.zeros(image\_shape[:2], dtype=np.float32)  
  
 **for** name, region **in** regions.items():  
 mask = np.zeros(image\_shape[:2], dtype=np.float32)  
 y1, y2, x1, x2 = region  
 mask[y1:y2, x1:x2] = 1  
 # Apply Gaussian blur for smooth transitions  
 mask = cv2.GaussianBlur(mask, (31, 31), 10)  
 masks[name] = mask  
 total\_mask += mask  
  
 # Normalize masks to ensure they sum to 1 (Property 1)  
 total\_mask = np.maximum(total\_mask, 1e-7)  
 **for** name **in** masks:  
 masks[name] = masks[name] / total\_mask  
  
 **return** masks

Η συνάρτηση create\_masks δημιουργεί τις απαραίτητες μάσκες βάσει των καθορισμένων περιοχών, εφαρμόζοντας Gaussian θόλωση στα άκρα για ομαλή μετάβαση και εξασφαλίζοντας ότι οι μάσκες πληρούν τις ιδιότητες 1 και 2.

**2. Δημιουργία Γκαουσιανών Πυραμίδων**

Για κάθε μάσκα , δημιουργήσαμε την Γκαουσιανή πυραμίδα με L+1 επίπεδα, εφαρμόζοντας Gaussian φιλτράρισμα και υποδειγματοληψία κατά παράγοντα 2.

**def** **generate\_gaussian\_pyramid**(img, levels):  
 """  
 Generate Gaussian pyramid with L+1 levels as per equation (1) in PDF  
 """  
 pyramid = [img.astype(np.float32)]  
 kernel = create\_gaussian\_kernel()  
  
 **for** i **in** range(levels):  
 filtered = gaussian\_filter(pyramid[-1], kernel)  
 # Downsample by factor of 2  
 downsampled = filtered[::2, ::2]  
 pyramid.append(downsampled)  
  
 **return** pyramid

Η συνάρτηση generate\_gaussian\_pyramid εφαρμόζει το Gaussian φίλτρο σε κάθε επίπεδο της εικόνας και στη συνέχεια υποδειγματοληπτεί την εικόνα, δημιουργώντας έτσι μια πυραμίδα με μειούμενη ανάλυση σε κάθε επίπεδο.

**3. Δημιουργία Λαπλασιανής Πυραμίδας LIkL\_{I\_k}**

Για κάθε εικόνα , κατασκευάσαμε την Λαπλασιανή πυραμίδα με L+1 επίπεδα, υπολογίζοντας τις διαφορές μεταξύ διαδοχικών επιπέδων της Γκαουσιανής πυραμίδας της εικόνας.

**def** **generate\_laplacian\_pyramid**(img, levels):  
 """  
 Generate Laplacian pyramid with L+1 levels as per equation (2) in PDF  
 """  
 gaussian\_pyramid = generate\_gaussian\_pyramid(img, levels)  
 pyramid = []  
  
 **for** i **in** range(levels):  
 size = (gaussian\_pyramid[i].shape[1], gaussian\_pyramid[i].shape[0])  
 expanded = cv2.resize(gaussian\_pyramid[i + 1], size)  
 laplacian = gaussian\_pyramid[i] - expanded  
 pyramid.append(laplacian)  
  
 # Add the smallest Gaussian level at the end  
 pyramid.append(gaussian\_pyramid[-1])  
  
 **return** pyramid

Η συνάρτηση generate\_laplacian\_pyramid υπολογίζει τις διαφορές μεταξύ διαδοχικών επιπέδων της Γκαουσιανής πυραμίδας, δημιουργώντας έτσι την Λαπλασιανή πυραμίδα που αναδεικνύει τις υψηλές συχνότητες και τις λεπτομέρειες της εικόνας.

**4. Δημιουργία Πυραμίδας Blend**

Η πυραμίδα Blend δημιουργήθηκε συνδυάζοντας τις πυραμίδες των εικόνων με βάση τις μάσκες. Για κάθε επίπεδο, οι Λαπλασιανές πυραμίδες των εικόνων συνδυάστηκαν χρησιμοποιώντας τις αντίστοιχες μάσκες.

**def** **blend\_pyramids**(pyramid1, pyramid2, mask\_pyramid):  
 """  
 Blend pyramids according to equation (4)  
 """  
 blended = []  
 **for** i **in** range(len(pyramid1)):  
 p1 = pyramid1[i]  
 p2 = pyramid2[i]  
 mask = mask\_pyramid[i]  
  
 **if** len(p1.shape) == 3:  
 mask = np.expand\_dims(mask, axis=2)  
  
 blend = mask \* p1 + (1 - mask) \* p2  
 blended.append(blend)  
  
 **return** blended

Η συνάρτηση blend\_pyramids συνδυάζει τις Λαπλασιανές πυραμίδες των δύο εικόνων βάσει της πυραμίδας της μάσκας, εξασφαλίζοντας ότι σε κάθε κλίμακα, η συμβολή των δύο εικόνων ελέγχεται από τη μάσκα, επιτυγχάνοντας έτσι μια ομαλή μετάβαση.

**5. Ανακατασκευή Τελικής Εικόνας**

Η τελική εικόνα ανασυντίθεται προσθέτοντας τα επίπεδα της συνδυασμένης πυραμίδας μέσω της συνάρτησης reconstruct\_from\_pyramid.

**def** **reconstruct\_from\_pyramid**(pyramid):  
 """  
 Reconstruct the final image from a pyramid  
 """  
 current = pyramid[-1]  
  
 **for** level **in** reversed(pyramid[:-1]):  
 size = (level.shape[1], level.shape[0])  
 expanded = cv2.resize(current, size)  
 current = expanded + level  
  
 **return** current

Η συνάρτηση reconstruct\_from\_pyramid ανασυντίθεται την τελική εικόνα προσθέτοντας τα επίπεδα της συνδυασμένης πυραμίδας από το υψηλότερο στο χαμηλότερο, αποκαθιστώντας την πολυκλιμακωτή πληροφορία της αρχικής εικόνας.

**exercise3.py:**

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
import os  
  
  
def create\_gaussian\_kernel():  
 *"""  
 Create the Gaussian kernel h = [1 4 6 4 1]/16 as specified in the PDF  
 """* return np.array([1, 4, 6, 4, 1]) / 16.0  
  
  
def gaussian\_filter(img, kernel):  
 *"""  
 Apply separable Gaussian filtering  
 """* # Handle both grayscale and color images  
 if len(img.shape) == 3:  
 filtered = np.zeros\_like(img, dtype=np.float32)  
 for i in range(3):  
 filtered[:, :, i] = cv2.sepFilter2D(img[:, :, i].astype(np.float32), -1,  
 kernel, kernel)  
 return filtered  
 return cv2.sepFilter2D(img.astype(np.float32), -1, kernel, kernel)  
  
  
def generate\_gaussian\_pyramid(img, levels):  
 *"""  
 Generate Gaussian pyramid with L+1 levels as per equation (1) in PDF  
 """* pyramid = [img.astype(np.float32)]  
 kernel = create\_gaussian\_kernel()  
  
 for i in range(levels):  
 filtered = gaussian\_filter(pyramid[-1], kernel)  
 # Downsample by factor of 2  
 downsampled = filtered[::2, ::2]  
 pyramid.append(downsampled)  
  
 return pyramid  
  
  
def generate\_laplacian\_pyramid(img, levels):  
 *"""  
 Generate Laplacian pyramid with L+1 levels as per equation (2) in PDF  
 """* gaussian\_pyr = generate\_gaussian\_pyramid(img, levels)  
 pyramid = []  
  
 for i in range(levels):  
 size = (gaussian\_pyr[i].shape[1], gaussian\_pyr[i].shape[0])  
 expanded = cv2.resize(gaussian\_pyr[i + 1], size)  
 laplacian = gaussian\_pyr[i] - expanded  
 pyramid.append(laplacian)  
  
 # Add the smallest Gaussian level at the end  
 pyramid.append(gaussian\_pyr[-1])  
  
 return pyramid  
  
  
def create\_masks(image\_shape, regions):  
 *"""  
 Create masks following properties 1 and 2:  
 1. Sum of all masks = 1 everywhere  
 2. mk(n) = 1 where image k should be visible  
 """* masks = {}  
 total\_mask = np.zeros(image\_shape[:2], dtype=np.float32)  
  
 for name, region in regions.items():  
 mask = np.zeros(image\_shape[:2], dtype=np.float32)  
 y1, y2, x1, x2 = region  
 mask[y1:y2, x1:x2] = 1  
 # Apply Gaussian blur for smooth transitions  
 mask = cv2.GaussianBlur(mask, (31, 31), 10)  
 masks[name] = mask  
 total\_mask += mask  
  
 # Normalize masks to ensure they sum to 1 (Property 1)  
 total\_mask = np.maximum(total\_mask, 1e-7)  
 for name in masks:  
 masks[name] = masks[name] / total\_mask  
  
 return masks  
  
  
def blend\_pyramids(pyramid1, pyramid2, mask\_pyramid):  
 *"""  
 Blend pyramids according to equation (4)  
 """* blended = []  
 for i in range(len(pyramid1)):  
 p1 = pyramid1[i]  
 p2 = pyramid2[i]  
 mask = mask\_pyramid[i]  
  
 if len(p1.shape) == 3:  
 mask = np.expand\_dims(mask, axis=2)  
  
 blend = mask \* p1 + (1 - mask) \* p2  
 blended.append(blend)  
  
 return blended  
  
  
def reconstruct\_from\_pyramid(pyramid):  
 *"""  
 Reconstruct the final image from a pyramid  
 """* current = pyramid[-1]  
  
 for level in reversed(pyramid[:-1]):  
 size = (level.shape[1], level.shape[0])  
 expanded = cv2.resize(current, size)  
 current = expanded + level  
  
 return current  
  
  
def main():  
 # Load images and resize to same dimensions  
 target\_size = (800, 1200) # Height, Width  
 image\_paths = {  
 'background': 'photos/P200.jpg',  
 'dog1': 'photos/dog1.jpg',  
 'dog2': 'photos/dog2.jpg',  
 'cat': 'photos/cat.jpg',  
 'bench': 'photos/bench.jpg',  
 'me': 'photos/nick.png'   
 }  
  
 # Load and resize images  
 images = {}  
 for name, path in image\_paths.items():  
 img = cv2.imread(path)  
 if img is None:  
 print(f"Failed to load {path}")  
 continue  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
 img = cv2.resize(img, (target\_size[1], target\_size[0]))  
 images[name] = img.astype(np.float32) / 255.0  
  
 # Define regions for each image (y1, y2, x1, x2)  
 regions = {  
 'dog1': (250, 550, 150, 450),  
 'dog2': (150, 450, 550, 850),  
 'cat': (350, 650, 750, 1050),  
 'bench': (450, 750, 250, 550),  
 'me': (150, 450, 350, 650)  
 }  
  
 # Number of pyramid levels  
 L = 6  
  
 # Create masks  
 masks = create\_masks(target\_size + (3,), regions)  
  
 # Initialize result with background  
 result = images['background'].copy()  
  
 # Process each image  
 for name, img in images.items():  
 if name == 'background':  
 continue  
  
 # Generate pyramids  
 mask\_gaussian = generate\_gaussian\_pyramid(masks[name], L)  
 img\_laplacian = generate\_laplacian\_pyramid(img, L)  
 result\_laplacian = generate\_laplacian\_pyramid(result, L)  
  
 # Blend pyramids  
 blended = blend\_pyramids(img\_laplacian, result\_laplacian, mask\_gaussian)  
  
 # Reconstruct and update result  
 result = reconstruct\_from\_pyramid(blended)  
 result = np.clip(result, 0, 1)  
  
 # Display results  
 plt.figure(figsize=(15, 10))  
  
 # Show original images  
 for i, (name, img) in enumerate(images.items()):  
 plt.subplot(3, 3, i + 1)  
 plt.imshow(img)  
 plt.title(name)  
 plt.axis('off')  
  
 # Show final result  
 plt.subplot(3, 3, 7)  
 plt.imshow(result)  
 plt.title('Final Blended Result')  
 plt.axis('off')  
  
 plt.tight\_layout()  
 plt.show()  
  
 # Save result  
 plt.imsave('photos/blended\_composition.png', result)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

A collage of photos of animals

Description automatically generated

A blurry image of a person's face

Description automatically generated

[**ΔΙΑΔΙΚΑΣΙΑ 4.**](https://github.com/nickpotamianos/Computer_Vision_Lab/blob/main/CV_1-PYRAMIDS/%5BPUBLIC%5D%20CV_1-PYRAMIDS-files/codes_and_models/pspnet_eval.ipynb) **(ή** [**Google Drive**](https://drive.google.com/drive/folders/1XNvzTHxq6UscvCHwUAZvLmGmhNDFavWT?usp=sharing) **με double click στο .ipynb για να ανοίξει στο Google Colab)**

**pspnet\_eval.py:**

import numpy as np  
import torch  
import torch.nn as nn  
from torch.utils.data import DataLoader  
from cityscapes\_dataset import Cityscapes  
from pspnet import PSPNet  
import os  
import cv2  
  
  
  
def calculate\_metrics(pred, target, num\_classes):  
 *"""  
 Calculate mean IoU (Intersection over Union) and pixel accuracy  
 Args:  
 pred: Predicted labels  
 target: Ground truth labels  
 num\_classes: Number of classes  
 Returns:  
 mean\_iou: Mean IoU score  
 pixel\_acc: Pixel accuracy  
 """* # Initialize confusion matrix  
 confusion\_matrix = np.zeros((num\_classes, num\_classes))  
  
 # Populate confusion matrix  
 for t, p in zip(target.flatten(), pred.flatten()):  
 if t < num\_classes: # Ignore index 255  
 confusion\_matrix[t, p] += 1  
  
 # Calculate IoU for each class  
 intersection = np.diag(confusion\_matrix)  
 union = np.sum(confusion\_matrix, axis=1) + np.sum(confusion\_matrix, axis=0) - intersection  
 iou = intersection / (union + 1e-10)  
  
 # Calculate mean IoU  
 mean\_iou = np.mean(iou)  
  
 # Calculate pixel accuracy  
 pixel\_acc = np.sum(intersection) / (np.sum(confusion\_matrix) + 1e-10)  
  
 return mean\_iou, pixel\_acc, iou  
  
  
def evaluate\_model():  
 # Set device  
 device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
  
 # Load validation dataset  
 dataset\_val = Cityscapes(  
 split='val',  
 data\_root='/content/drive/MyDrive/CV\_1-PYRAMIDS-files/cityscapes\_dataset/',  
 data\_list='/content/drive/MyDrive/CV\_1-PYRAMIDS-files/cityscapes\_dataset/list/cityscapes/fine\_val.txt'  
 )  
  
 # Create dataloader  
 val\_dataloader = DataLoader(dataset\_val, batch\_size=1, shuffle=False, num\_workers=4)  
  
 # Initialize model  
 num\_classes = 35 # Number of classes in Cityscapes dataset  
 model = PSPNet(layers=50, bins=(2, 3, 6, 8), dropout=0.1, classes=num\_classes,  
 zoom\_factor=8, use\_ppm=True, pretrained=False).to(device)  
  
 # Load trained weights  
 model.load\_state\_dict(torch.load('train\_epoch\_200\_CPU.pth'))  
 model.eval()  
  
 # Initialize metrics storage  
 all\_ious = []  
 all\_pixel\_accs = []  
 class\_ious = np.zeros(num\_classes)  
 num\_images = len(val\_dataloader)  
  
 with torch.no\_grad():  
 for i, (images, labels) in enumerate(val\_dataloader):  
 # Forward pass  
 images = images.to(device)  
 labels = labels.to(device)  
 outputs = model(images)  
  
 # Get predictions  
 \_, predictions = outputs.max(1)  
  
 # Calculate metrics  
 mean\_iou, pixel\_acc, iou\_per\_class = calculate\_metrics(  
 predictions[0].cpu().numpy(),  
 labels[0].cpu().numpy(),  
 num\_classes  
 )  
  
 # Store metrics  
 all\_ious.append(mean\_iou)  
 all\_pixel\_accs.append(pixel\_acc)  
 class\_ious += iou\_per\_class  
  
 if (i + 1) % 10 == 0:  
 print(f'Processed {i + 1}/{num\_images} images')  
  
 # Calculate final metrics  
 final\_mean\_iou = np.mean(all\_ious)  
 final\_mean\_pixel\_acc = np.mean(all\_pixel\_accs)  
 final\_class\_ious = class\_ious / num\_images  
  
 # Calculate standard deviations  
 iou\_std = np.std(all\_ious)  
 pixel\_acc\_std = np.std(all\_pixel\_accs)  
  
 print("\nFinal Results:")  
 print(f"Mean IoU: {final\_mean\_iou:.4f} ± {iou\_std:.4f}")  
 print(f"Mean Pixel Accuracy: {final\_mean\_pixel\_acc:.4f} ± {pixel\_acc\_std:.4f}")  
  
 # Print per-class IoU  
 print("\nPer-class IoU:")  
 with open('cityscapes\_names.txt', 'r') as f:  
 class\_names = [line.strip() for line in f.readlines()]  
  
 for i, class\_iou in enumerate(final\_class\_ious):  
 if i < len(class\_names):  
 print(f"{class\_names[i]}: {class\_iou:.4f}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 evaluate\_model()

**cityscapes\_dataset.py**

...  
 elif split == 'val':  
 self.transform = tfs.Compose([  
 tfs.Crop([713, 713], crop\_type='center', padding=mean, ignore\_label=255),  
 tfs.ToTensor(),  
 tfs.Normalize(mean=mean, std=std)])  
  
 def \_\_len\_\_(self):  
 return len(self.data\_list)  
...

**Output:**

Processed 10/500 images  
Processed 20/500 images  
Processed 30/500 images  
...  
Processed 470/500 images  
Processed 480/500 images  
Processed 490/500 images  
Processed 500/500 images  
  
Final Results:  
Mean IoU: 0.2231 ± 0.0452  
Mean Pixel Accuracy: 0.9238 ± 0.0910  
  
Per-class IoU:  
unlabeled: 0.0000  
ego vehicle: 0.9329  
rectification border: 0.0000  
out of roi: 0.0000  
static: 0.2085  
dynamic: 0.0542  
ground: 0.0319  
road: 0.9452  
sidewalk: 0.5173  
parking: 0.0295  
rail track: 0.0017  
building: 0.7532  
wall: 0.0570  
fence: 0.0512  
guard rail: 0.0000  
bridge: 0.0058  
tunnel: 0.0000  
pole: 0.3725  
polegroup: 0.0061  
traffic light: 0.2265  
traffic sign: 0.4504  
vegetation: 0.8094  
terrain: 0.1214  
sky: 0.5351  
person: 0.3542  
rider: 0.1604  
car: 0.7562  
truck: 0.0525  
bus: 0.0808  
caravan: 0.0021  
trailer: 0.0000  
train: 0.0106  
motorcycle: 0.0351  
bicycle: 0.2471  
license plate: 0.0000