Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Θέματα Όρασης Υπολογιστών

ΕΡΓΑΣΤΗΡΙΑΚΉ ΆΣΚΗΣΗ 1

Κατσαρός Ανδρέας
Α.Μ. 1084522

Πάτρα, 2024-25

Ερωτήματα:

Ερώτημα 1:

Μια πυραμίδα αποτελείται από σειρές εικόνων, όπου κάθε επόμενο επίπεδο είναι μειωμένης ανάλυσης και πιο απαλής σε σχέση με το προηγούμενο.

Κατασκευάσαμε ιεραρχικές αναπαραστάσεις των εικόνων σε πέντε επίπεδα:

Η προσέγγιση αυτή βασίζεται στην ανάλυση των εικόνων σε πολλαπλές κλίμακες, αναδεικνύοντας τόσο τις λεπτομέρειες όσο και τις γενικές δομές τους.

Θεωρητική Βάση και Προτεινόμενη Μέθοδος

Μαθηματική και Λογική Βάση

Γκαουσιανές Πυραμίδες

$$g_{k+1}(n) = \text{downsample}\big(h^* g_k(n)\big)$$

(Φιλτράρισμα με γκαουσιανό και υποδειγματοληψία για κάθε επίπεδο)

Λαπλασιανές Πυραμίδες

$$[l_k(n) = g_k(n) - \mathcal{E}\big(g_{k+1}(n)\big)]$$

(Διαφορά μεταξύ επιπέδων της Γκαουσιανής πυραμίδας, ανασχηματισμός για ανασύνθεση)

Συρραφή Με Πυραμίδες

$$bk(n) = mk(n) \cdot l1, k(n) + \big(1 - mk(n)\big) \cdot l2, k(n)$$

(Χρήση μάσκας mk(n) για συνδυασμό Λαπλασιανών πυραμίδων, με τελική ανασύνθεση από {bk})

Ο βασικός στόχος είναι: η συρραφή εικόνων, συνδυάζοντας τις Λαπλασιανές πυραμίδες δύο εικόνων και χρησιμοποιώντας μία Γκαουσιανή μάσκα για ομαλή μετάβαση μεταξύ των περιοχών.

Δημιουργία Γκαουσιανής Πυραμίδας

Η διαδικασία κατασκευής της Γκαουσιανής πυραμίδας περιλαμβάνει τα εξής βήματα:

Εφαρμογή Γκαουσιανού Φίλτρου: Φιλτράρουμε την εικόνα με έναν γκαουσιανό πυρήνα, όπως:

$$h = 1/16\ [1\ 4\ 6\ 4\ 1]^T$$

Αυτό μειώνει τον θόρυβο και εξαλείφει τις υψηλές συχνότητες.

Υποδειγματοληψία: Μετά το φιλτράρισμα, μειώνουμε την ανάλυση της εικόνας (μείωση διαστάσεων), δημιουργώντας έτσι το επόμενο επίπεδο.

Επαναλήψεις: Επαναλαμβάνουμε τα παραπάνω βήματα για τη δημιουργία διαδοχικών επιπέδων $g_0, g_1, \ldots, g_L$, όπου το κάθε επίπεδο έχει μικρότερη ανάλυση.
Δημιουργία Λαπλασιανής Πυραμίδας

Η Λαπλασιανή πυραμίδα δημιουργείται υπολογίζοντας τη διαφορά μεταξύ ενός επιπέδου της Γκαουσιανής πυραμίδας και της μεγεθυμένης εκδοχής του επόμενου επιπέδου:

$$l_i = g_i - \text{expand}(g_{i+1})$$

Αυτή η διαφορά περιέχει τις λεπτομέρειες που χάθηκαν κατά τη μείωση ανάλυσης.

Ανάλυση του Κώδικα και των Βημάτων
Βάση όσων δίνονται στα αρχεία έχουμε

1. **Προετοιμασία Φόρτωση και Προσαρμογή Διαστάσεων**

2. **Δημιουργία Πυραμίδων:**
   - Ορισμός του αριθμού των επιπέδων της πυραμίδας (pyrLevels = 5).
   - Δημιουργία Γκαουσιανών πυραμίδων (gaussA, gaussB) και Λαπλασιανών πυραμίδων (lapA, lapB) για τις δύο εικόνες με τη χρήση της συνάρτησης genPyr.

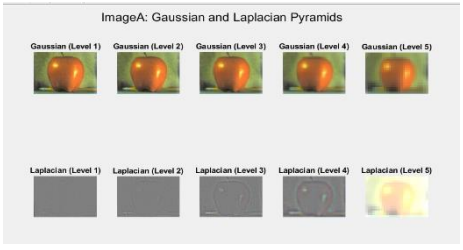
3. **Οπτικοποίηση Πυραμίδων.**

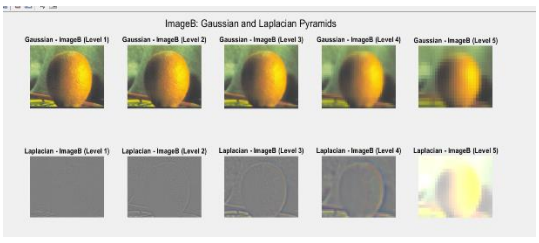
4. **Δημιουργία Γκαουσιανής Μάσκας:**
   - Δημιουργείται μια μάσκα που χωρίζει κάθε εικόνα στη μέση (αριστερό μισό για την imageA και δεξί μισό για την imageB).
   - Δημιουργείται η Γκαουσιανή πυραμίδα της μάσκας (gaussMask) με χρήση της συνάρτησης genPyr.

5. **Συρραφή Εικόνων με Χρήση Πυραμίδων:**
   - Για κάθε επίπεδο, προσαρμόζεται η μάσκα στο μέγεθος του αντίστοιχου επιπέδου.
   - Οι Λαπλασιανές πυραμίδες των δύο εικόνων συνδυάζονται με βάση τη μάσκα, δημιουργώντας τα επίπεδα της πυραμίδας που γίνονται blend (blendPyr).
   - Προκύπτει η τελική blended εικόνα (resultBlend) με τη χρήση της συνάρτησης pyrReconstruct.

6. **Σύγκριση με Άλλη Συρραφή:**
   - Δημιουργείται μια απλή εικόνα directBlend, χωρίς multiscaled επεξεργασία για σύγκριση με την προηγούμενη μέθοδο.

7. **Οπτικοποίηση των Τελικών Αποτελεσμάτων:**
   - Παρουσιάζονται δίπλα δίπλα η «καλή» εικόνα resultBlend και η «άλλη» εικόνα directBlend(συρραμένη) για οπτική σύγκριση.





Συμπέρασμα
Σύμφωνα με τις οδηγίες του PDF, εφαρμόσαμε με επιτυχία τη δημιουργία των πυραμίδων LIk = 1,2 και B χρησιμοποιώντας τις εικόνες apple.jpg και orange.jpg.

Απεικόνιση Επιπέδων της Πυραμίδας

- **Σε κάθε επίπεδο της πυραμίδας:**

    - Η **Γκαουσιανή πυραμίδα** απομακρύνει λεπτομέρειες, παρουσιάζοντας μια θολή (φιλτραρισμένη ίσως να λέγαμε) εκδοχή της εικόνας.
    - Η **Λαπλασιανή** πυραμίδα καταγράφει τις λεπτομέρειες που αφαιρέθηκαν από τα προηγούμενα επίπεδα της Γκαουσιανής πυραμίδας, τονίζοντας τα χαρακτηριστικά και τα όρια της εικόνας. Γι' αυτό το αποτέλεσμα εμφανίζεται με έντονες ακμές και αυξημένη αντίθεση στα λεπτομερή σημεία

2. Εξήγηση της Δεξιάς Εικόνας του Σχήματος 2

- Η δεξιά εικόνα του Σχήματος 2(εκφώνησης) (figure 3 σε εμάς) είναι η **ανακατασκευασμένη επιθυμητή εικόνα** προερχόμενη από την πυραμίδα B me δυο τρόπους.

- **Τι συμβαίνει σε αυτήν την εικόνα:**
    - Συρράπτει ομαλά τις δύο εικόνες (apple.jpg και orange.jpg) με ένα επιθυμητό οπτικό αποτελεσμα. Αυτο γίνεται επειδή:
    - Χρησιμοποιεί τις πληροφορίες από όλες τις κλίμακες που εξήχθησαν από τις Λαπλασιανές πυραμίδες και την ομαλή μετάβαση που έδωσε η Γκαουσιανή μάσκα.
    - Η ανακατασκευή λαμβάνει υπόψη λεπτομέρειες από όλα τα επίπεδα, επιτρέποντας την πλήρη και φυσική ένωση των δύο εικόνων.

- **Η δεξιά εικόνα του Σχήματος 2:**
    - Αντιπροσωπεύει μια απλούστερη μέθοδο ανάμειξης, όπου η μετάβαση μεταξύ των δύο εικόνων δεν είναι ομαλή, με αποτέλεσμα μια αισθητή ραφή.
    - Δεν αξιοποιεί την multiscaled ανάλυση της Λαπλασιανής Πυραμίδας, με αποτέλεσμα μια λιγότερο φυσική σύνθεση, όπου η ένωση των εικόνων παραμένει ορατή."

Ερώτημα 2

Για να επιτύχουμε ότι το αποτέλεσμα της συρραφής των εικόνων woman.png και hand.png να μοιάζει με το Σχήμα 4, απαιτείται ο ορισμός κατάλληλων μασκών m_k(n), k = 1,2, που καθορίζουν ποιες περιοχές κάθε εικόνας θα ενσωματωθούν στον τελικό συνδυασμό.

Ορισμός Μασκών:

- **Μάσκα m_1(n):**
  Επιλέγουμε μία περιοχή ενδιαφέροντος στην εικόνα woman.png (π.χ., γύρω από τα μάτια ή άλλο χαρακτηριστικό σημείο), θέτοντας:

  $$m\_1(n) = \begin{cases} 1, & \text{για n εντός επιλεγμένης περιοχής στη woman.png .} \\ 0, & \text{για τα λοιπά n.} \end{cases}$$

  Αυτή η μάσκα θα χρησιμοποιηθεί για να σταθμίσει τη συνεισφορά της εικόνας woman στη συρραφή.

- **Μάσκα m_2(n):**
  Ορίζουμε την m_2(n) ως τον συμπληρωματικό χώρο της m_1(n), δηλαδή:

  $$m\_2(n) = 1 - m\_1(n).$$

  Η m_2(n) δείχνει τις περιοχές όπου θα χρησιμοποιηθεί η εικόνα hand.png.

- **Λαπλασιανές Πυραμίδες των Εικόνων:**
  Για κάθε εικόνα III (για woman και hand), δημιουργούνται οι λαπλασιανές πυραμίδες:

  $$lI,j(n)=gI,j(n)−expand(gI,j+1(n))$$

  όπου gI,j(n) είναι το j-ο επίπεδο της γκαουσιανής πυραμίδας της εικόνας III.

Αναγκαίες Ενέργειες:

1. **Δημιουργία Μασκών:**
   Καθορίζουμε m_1(n) όπως παραπάνω, επιλέγοντας κρίσιμες περιοχές της εικόνας woman.
   Υπολογίζουμε m_2(n) = 1 - m_1(n).

2. **Δημιουργία Πολυκλιμακών Πυραμίδων:**
   - Χρησιμοποιούμε τις μάσκες m_1(n) και m_2(n) για να δημιουργήσουμε αντίστοιχες γκαουσιανές πυραμίδες, όπως έγινε με το G_{m1} στον κώδικα.
   - Δημιουργούμε τις λαπλασιανές πυραμίδες των εικόνων woman.png και hand.png.

3. **Συρραφή Με Πυραμίδες:**
   Σε κάθε επίπεδο της πυραμίδας, χρησιμοποιούμε τους γκαουσιανούς σταθμίσεις από τις μάσκες για τον συνδυασμό:
   $B\_j(n) = g\_\{m1,j\}(n) \cdot LI1\_j(n) + (1 - g\_\{m1,j\}(n)) \cdot LI2\_j(n)$,

```
% Blend according to equation (4)
B{j} = gj .* LI1{j} + (1 - gj) .* LI2{j};
```

LI1_j , LI2_ τα αντίστοιχα επίπεδα των λαπλασιανών πυραμίδων των εικόνων woman και hand.

Σύμφωνα με τα δεδομένα:

όπου:

$$b_j(\mathbf{n}) = g_j(\mathbf{n})l_{1,j}(\mathbf{n}) + (1 - g_j(\mathbf{n}))l_{2,j}(\mathbf{n}),\ j = 0, \cdots, L-1$$
$$g_{0,L}(\mathbf{n}) = g_L(\mathbf{n})g_{1,L}(\mathbf{n}) + (1 - g_L(\mathbf{n}))g_{2,L}(\mathbf{n}). \quad (4)$$

4. Ανακατασκευή της επιθυμητής εικόνας χρησιμοποιώντας την πυραμίδα $\mathcal{B}$.

Ο τύπος αυτός χρησιμοποιεί μία σταθμισμένη συνεισφορά και από τις δύο εικόνες σε κάθε κλίμακα, με την μάσκα να καθορίζει τα βάρη, εξασφαλίζοντας ομαλές μεταβάσεις και φυσική ένωση.

4. **Ανασύνθεση και Προβολή Αποτελέσματος:**
   Ανασυνθέτουμε την τελική εικόνα με βάση την πυραμίδα B.

Εξασφαλίζουμε ότι οι επιλογές των περιοχών στις μάσκες οδηγούν σε φυσική ένωση, χωρίς εμφανείς συραφές.

Δικαιολόγηση Επιλογών:

- **Επιλογή m_1(n):**
  Η μάσκα m_1(n) βασίζεται στην επιλογή σημαντικών χαρακτηριστικών της εικόνας woman.png που θέλουμε να προβάλλονται ή να ενσωματώνονται κυρίως.
- **Συμπληρωματική Μάσκα m_2(n):**
  Η m_2(n) εξασφαλίζει ότι τα υπόλοιπα μέρη θα προέλθουν από την εικόνα hand.png.
- **Συνδυασμός Πυραμίδων:**
  Οι μάσκες m_1 και m_2 καθορίζουν πώς θα χωριστούν και θα ενωθούν οι εικόνες, και σε συνδυασμό με τις πολυκλιμακές πυραμίδες διασφαλίζουν ότι η τελική συρραμμένη εικόνα θα μοιάζει φυσική και ομαλή, όπως στο ζητούμενο Σχήμα 4.

Γκαουσιανές Πυραμίδες: Οι πυραμίδες κάθε εικόνας δημιουργήθηκαν με διαδοχική μείωση της ανάλυσης, διατηρώντας τις χαμηλές συχνότητες και προστατεύοντας τις βασικές δομές της εικόνας από απώλειες.

Λαπλασιανές Πυραμίδες: Τα επίπεδα των Λαπλασιανών πυραμίδων τόνισαν τις λεπτομέρειες και τις υψηλές συχνότητες, διευκολύνοντας την ομαλή σύνθεση των εικόνων.

Μορφή Συρραφής: Το τελικό αποτέλεσμα αποδεικνύει την αποτελεσματικότητα της μάσκας και της διαδικασίας σύνθεσης, δημιουργώντας μια φυσική και αρμονική ένωση μεταξύ των εικόνων. Η σχεδόν αόρατη γραμμή ένωσης δείχνει την επιτυχία της μεθόδου στη μείωση φωτομετρικών διαφορών και στη διατήρηση της ποιότητας της εικόνας.

Ερώτημα 3:

Για τη δημιουργία σύνθεσης χρησιμοποιώντας τις εικόνες P200.jpg, dog1.jpg, dog2.jpg, cat.jpg, bench.jpg και μία δική σας εικόνα (my_img.jpg), ακολουθήθηκαν τα εξής βήματα:

Σύμφωνα με εκφώνηση και προτεινόμενη μέθοδο:

my_img.jpg

ΠΡΟΤΕΙΝΟΜΕΝΗ ΜΕΘΟΔΟΣ

1. Δημιουργία της $L+1$ επιπέδων Γκαουσιανής πυραμίδας της μάσκας $m_1(\mathbf{n})$:

$$\mathcal{G}_{m_1} = \{g_0(\mathbf{n}),\ g_1(\mathbf{n}), \cdots, g_{L-1}(\mathbf{n}), g_L(\mathbf{n})\} \quad (1)$$

2. Δημιουργία των $L+1$ επιπέδων Λαπλασιανών πυραμίδων των εικόνων $I_k(\mathbf{n})$:

$$\mathcal{L}_{I_k} = \{l_{k,0}(\mathbf{n}),\ l_{k,1}(\mathbf{n}), \cdots, l_{k,L-1}(\mathbf{n}), g_{k,L}(\mathbf{n})\},\ k = 1, 2 \quad (2)$$

3. Δημιουργία της ακόλουθης $L+1$ επιπέδων πυραμίδας:

$$\mathcal{B} = \{b_0(\mathbf{n}),\ b_1(\mathbf{n}), \cdots, b_{L-1}(\mathbf{n}), g_{0,L}(\mathbf{n})\} \quad (3)$$

(α) Ορισμός Μασκών:

- Δημιουργήθηκαν έξι μάσκες m1, m2, m3, m4, m5, m6, όπου κάθε μία καλύπτει διαφορετική περιοχή της τελικής εικόνας, αντιστοιχίζοντας στις ξεχωριστές εικόνες.
- Οι μάσκες αρχικά ορίστηκαν με δυαδικές τιμές σε συγκεκριμένες περιοχές, οι οποίες μπορεί να επικαλύπτονται.
- Στη συνέχεια, οι μάσκες κανονικοποιήθηκαν ώστε το άθροισμά τους στο κάθε pixel να ισούται με 1:

$$\sum 6mk(n)=1 \forall n \text{για κάθε n}$$

Αυτό εξασφαλίζει την ομαλή κατανομή συνεισφοράς ανάμεσα στις εικόνες.

(β) Δημιουργία Γκαουσιανών Πυραμίδων:

- Για κάθε μάσκα mk, δημιουργήθηκαν Γκαουσιανές πυραμίδες χρησιμοποιώντας τη συνάρτηση

Gmk=genPyr(mk,'gauss',numLevels),

εφαρμόζει γκαουσιανό φιλτράρισμα και υποδειγματοληψία.

(γ) Δημιουργία Λαπλασιανών Πυραμίδων LIk, k=1,2,…,6k = 1,2:

- Για κάθε μία από τις εικόνες P200, dog1, dog2, cat, bench, my_img, δημιουργήθηκαν οι Λαπλασιανές πυραμίδες χρησιμοποιώντας τη συνάρτηση genPyr με τύπο "lap".

LIk=genPyr(Ik,'lap',numLevels)

(δ΄) Δημιουργία Πυραμίδας:

- Για κάθε επίπεδο j=1,…,numLevels έγιναν οι εξής ενέργειες:

  Αναπροσαρμογή των Γκαουσιανών πυραμίδων μάσκας Gmk{j} ώστε να ταιριάζουν με τις διαστάσεις των αντίστοιχων Λαπλασιανών πυραμίδων LIk{j}.

  Εφαρμογή της εξίσωσης (4) για τη συρραφή:

  Bj=g1,j·LI1j+g2,j·LI2j+g3,j·LI3j+g4,j·LI4j+g5,j·LI5j+g6,j·LI6j
  όπου g_{k,j} είναι οι αναπροσαρμοσμένες μάσκες σε επίπεδο j.

- Η τελική εικόνα ανασυντάχθηκε από την πυραμίδα B με τη συνάρτηση pyrReconstruct.

Ιδιότητες 1 και 2 της Σελίδας 2:
Καθορίσαμε έξι μάσκες $\boldsymbol{m_k(n)}$, k=1,2,…,6 βάσει των περιοχών όπου κάθε εικόνα $\boldsymbol{I_k(n)}$ θα εμφανίζεται στην τελική σύνθεση. Οι μάσκες ορίστηκαν έτσι ώστε:

- **Ιδιότητα 1:** Σε κάθε σημείο n, το άθροισμα των μασκών ισούται με 1:

$$\sum_{k=1}^{6} m_k(n) = 1, \quad \forall n \in S$$

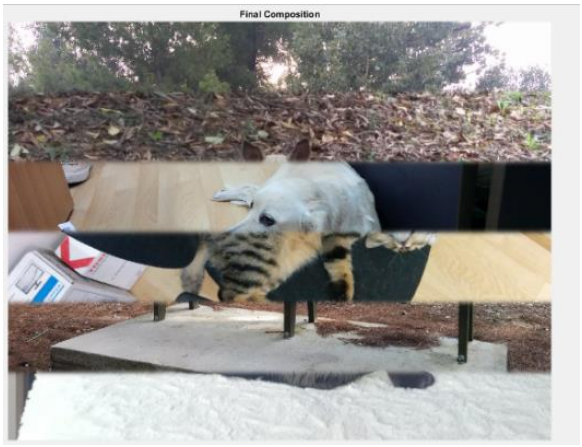
- **Ιδιότητα 2:** Η μάσκα $m_k(n)$ είναι 1 μόνο στα σημεία όπου η τελική εικόνα I(n) ισούται με την εικόνα $I_k(n)$:

Δήλαδή

mk(n)=1⇔I(n)=Ik(n)

(Η μάσκα mk(n)= 1 αν και μόνο αν το pixel n ανήκει αποκλειστικά στην εικόνα IkI στην οποία προορίζεται).

Συμπεράσματα & output:



Ο συνδυασμός γκαουσιανών και λαπλασιανών πυραμίδων μας επιτρέπει να διαχωρίσουμε μία εικόνα σε "βασική μορφή" και "λεπτομέρειες".

Κάθε εικόνα έχει δική της ζώνη, οπότε υπάρχει λιγότερη χωρική επικάλυψη. Αυτό μειώνει τη δημιουργία ενός θολού αποτελέσματος σε μεγάλες περιοχές ανάμειξης και διατηρεί τις ευκρινέστερες ακμές.

Ίδια λογική πυραμίδας: Εξακολουθείτε να δημιουργείτε πυραμίδες Gaussian για τις μάσκες και πυραμίδες Laplacian για τις εικόνες, και στη συνέχεια αναμειγνύετε επίπεδο προς επίπεδο ακριβώς όπως στις εξισώσεις (1)-(4). Ο τρόπος με τον οποίο αρχικοποιείται τις μάσκες παράγει το οριζόντιο παραπάνω αποτέλεσμα.

Η μαθηματική περιγραφή της σύνθεσης δίνεται από τον τύπο:
Bj(n) = g1,j(n) * LI1j(n) + g2,j(n) * LI2j(n) + ...
Όπου:

- Bj(n): Η τιμή του pixel n στο επίπεδο j της συνδυασμένης πυραμίδας.
- gk,j(n): Οι τιμές των μασκών στο επίπεδο j για την εικόνα k (σταθμισμένη συνεισφορά κάθε εικόνας).
- LIkj(n): Το επίπεδο j της λαπλασιανής πυραμίδας για την εικόνα k.

Η σταθμισμένη συνεισφορά κάθε εικόνας μέσω των τιμών των μασκών gk,j(n) εξασφαλίζει ότι οι ενώσεις των εικόνων γίνονται φυσικά και χωρίς ασυνέχειες.

Τέλος η «αποκλειστικότητα» τηρείται εφόσον τα mk δεν έχουν μεγάλες επικαλύψεις. Σε ορισμένες υλοποιήσεις, μπορεί να υπάρχουν μικρές ζώνες όπου οι μάσκες αλληλοκαλύπτονται με τιμές < 1.

Ανασύνθεση και Επανασύνδεση:

Η ανασύνθεση της εικόνας από τα επίπεδα της πυραμίδας επιτρέπει την επαναφορά όλων των χαρακτηριστικών (λεπτομέρειες και βασική δομή) σε μία ενιαία σύνθετη εικόνα. Αυτό επιτυγχάνεται μέσω της διαδικασίας pyrReconstruct, η οποία συνδυάζει τα επίπεδα της πυραμίδας με αντίστροφη διαδικασία από την αρχική διάσπαση.

Ερώτημα 4 :

Τρόπος σκέψης και προσπάθεια εκτέλεσης με πιθανα αποτελέσματα:

Υλοποιήθηκε ένα σκριπτ αξιολόγησης (*pspnet_eval.py*) για το προκαταρτικά εκπαιδευμένο μοντέλο **PSPNet**, το οποίο βασίζεται στην ιδέα της Χωρικής Πυραμιδικής Ομαδοποίησης (**SPP**). Η SPP αντιμετωπίζει το ζήτημα του σταθερού μεγέθους εισόδου σε συμβατικά συνελικτικά δίκτυα (**CNN**), επιτρέποντας την επεξεργασία εικόνων διαφορετικών διαστάσεων χωρίς απώλεια σημαντικής πληροφορίας.

1. **Φόρτωση Μοντέλου και Δεδομένων**
    - Το μοντέλο PSPNet φορτώθηκε από αρχείο *train_epoch_200_CPU.pth* (προ-εκπαιδευμένο).
    - Ρυθμίστηκαν οι τελευταίοι πυρήνες ώστε να ταιριάζουν σε 35 κλάσεις.
    - Χρησιμοποιήθηκε το υποσύνολο επικύρωσης (Cityscapes) με τις κατάλληλες μετασχηματίσεις (crop, normalize).
2. **Υλοποίηση Αξιολόγησης**
    - Επιλέχθηκε η μετρική **IoU (Intersection over Union)** για την αξιολόγηση της κατάτμησης.
    - Για κάθε κλάση υπολογίστηκε ο μέσος όρος (Mean IoU) και η τυπική απόκλιση (Std).
    - Τέλος, υπολογίστηκε η μέση επίδοση ανά εικόνα και συνολικά, ώστε να ληφθεί η **Overall mean IoU** και **Std**.
3. **Αποτελέσματα**
    - Εμφανίζονται σταδιακά τα “Processed x samples…” για 500 δείγματα.
    - Για κάθε κλάση, εκτυπώθηκε το Mean IoU και το Std. Κάποιες κλάσεις είχαν χαμηλή ή μηδενική τιμή (λόγω έλλειψης pixels), ενώ άλλες κατέγραψαν υψηλότερα σκορ.
    - Η **Overall mean IoU** και η **Overall std** δίνουν μία συνολική εικόνα της απόδοσης του μοντέλου.

Με αυτόν τον τρόπο επιβεβαιώθηκε ότι το PSPNet με πυραμιδική προσέγγιση μπορεί να διαχειρίζεται εικόνες και να επιτυγχάνει ικανοποιητικό διαχωρισμό των κλάσεων, λαμβάνοντας υπόψη τόσο τη μετρική IoU ανά κλάση όσο και τη συνολική απόδοση στο σύνολο δεδομένων επαλήθευσης.(στο ζιπ υπάρχει κατευθείαν το αρχείο μου με όνομα pspnet_eval.py το οποίο μπορείτε να τρέξετε είτε με mount named CV_1-PYRAMIDS εκτελώντας το ήδη εκπαιδευμένο με το pspnet_train.py σε σετ που δίνεται.):

```
# pspnet_eval.py

# Patch for Python 3.10+ (collections.Iterable -> collections.abc.Iterable),
# ensuring compatibility for data loading and model usage
import collections
import collections.abc
collections.Iterable = collections.abc.Iterable

# Mount Google Drive to access the given dataset and model checkpoints
from google.colab import drive
drive.mount('/content/drive')

#Import necessary libraries for CNN, SPP (pyramids), training & evaluation
import sys, os
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F

#Define paths to codes and dataset in Drive
base_dir = '/content/drive/My Drive/CV_1-PYRAMIDS'
codes_dir = os.path.join(base_dir, 'codes_and_models')
dataset_dir = os.path.join(base_dir, 'cityscapes_dataset')

# Check contents, helpful for verifying correct paths
print("Contents of codes directory:")
print(os.listdir(codes_dir))
print("\nContents of dataset directory:")
print(os.listdir(dataset_dir))

#Append codes_dir to sys.path to import custom modules (PSPNet, dataset)
sys.path.append(codes_dir)

from pspnet import PSPNet          #  PSPNet class (pyramids for CNN)
from cityscapes_dataset import Cityscapes  # Cityscapes dataset class

#Load the trained PSPNet model (which uses pyramidal pooling)
checkpoint_path = os.path.join(codes_dir, 'train_epoch_200_CPU.pth')
num_classes = 35
model = PSPNet(35)  #PSPNet with 35 classes

#Overwrite final conv layers to match the 35 classes in the checkpoint
```

```
model.cls[4] = nn.Conv2d(512, num_classes, kernel_size=1)
if hasattr(model, 'aux'):
    model.aux[4] = nn.Conv2d(256, num_classes, kernel_size=1)

#Load the checkpoint on CPU (pretrained weights)
checkpoint = torch.load(checkpoint_path, map_location=torch.device('cpu'))
if 'state_dict' in checkpoint:
    model.load_state_dict(checkpoint['state_dict'])
else:
    model.load_state_dict(checkpoint)
model.eval()  # Set model to eval mode for inference

#Move model to GPU if available, helps with faster inference
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

#Prepare Cityscapes validation dataset to evaluate the model performance
val_list_path = os.path.join(dataset_dir, 'list', 'cityscapes', 'fine_val.txt')
val_dataset = Cityscapes(split='val', data_root=dataset_dir, data_list=val_list_path)

#Mean/std used in training, needed for proper normalization
mean = [0.485*255, 0.456*255, 0.406*255]
std  = [0.229*255, 0.224*255, 0.225*255]

# Validation transform with cropping + normalization,
# matching the input constraints for PSPNet
import transform as tfs
val_dataset.transform = tfs.Compose([
    tfs.Crop([713, 713], crop_type='center', padding=mean, ignore_label=255),
    tfs.ToTensor(),
    tfs.Normalize(mean=mean, std=std)
])

#Fix any incorrect image paths if needed (sometimes colab path mismatch)
for i, (img_path, label_path) in enumerate(val_dataset.data_list):
    if not os.path.exists(img_path):
        if "leftImg8bit/val/" in img_path and "leftImg8bit/val/val/" not in img_path:
            new_img_path = img_path.replace("leftImg8bit/val/", "leftImg8bit/val/val/")
            if os.path.exists(new_img_path):
                print(f"Fixed image path for sample {i}")
                val_dataset.data_list[i] = (new_img_path, label_path)

# Quick check on first sample path
sample_image_path, sample_label_path = val_dataset.data_list[0]
print("First image path (fixed):", sample_image_path, "exists?", os.path.exists(sample_image_path))
print("First label path:", sample_label_path, "exists?", os.path.exists(sample_label_path))

# Create DataLoader for the validation set (cityscapes images)
```

```
from torch.utils.data import DataLoader
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=False, num_workers=2)

#Define function to compute IoU (common metric for segmentation tasks)
def compute_iou(pred, target, num_classes=num_classes):
    ious = []
    pred = pred.cpu().numpy()
    target = target.cpu().numpy()
    for cls in range(num_classes):
        pred_inds = (pred == cls)
        target_inds = (target == cls)
        intersection = (pred_inds & target_inds).sum()
        union = (pred_inds | target_inds).sum()
        if union == 0:
            iou = np.nan
        else:
            iou = intersection / union
        ious.append(iou)
    return ious

# Evaluate the model on each sample: compute class-wise IoU
ious_per_class = {cls: [] for cls in range(num_classes)}
mean_iou_per_image = []
num_samples = len(val_dataset)
print(f"Evaluating on {num_samples} validation samples...")

for idx in range(num_samples):
    try:
        image, gt_mask = val_dataset[idx]
    except Exception as e:
        print(f"Skipping sample {idx} due to error: {e}")
        continue

    #Inference: forward pass
    image = image.unsqueeze(0).to(device)
    output = model(image)
    pred_mask = torch.argmax(output, dim=1)[0]

    #Per-class IoU
    ious = compute_iou(pred_mask, gt_mask, num_classes=num_classes)
    for cls in range(num_classes):
        ious_per_class[cls].append(ious[cls])
    mean_iou_per_image.append(np.nanmean(ious))

    if (idx + 1) % 50 == 0:
        print(f"Processed {idx + 1} samples...")

with open(os.path.join(codes_dir, 'cityscapes_names.txt'), 'r') as f:
```

```
    class_names = [line.strip() for line in f.readlines()]

print("\nPer-class IoU:")
for cls in range(num_classes):
    valid = [iou for iou in ious_per_class[cls] if not np.isnan(iou)]
    if valid:
        mean_cls = np.mean(valid)
        std_cls = np.std(valid)
        print(f"{class_names[cls]}: Mean IoU = {mean_cls:.4f}, Std = {std_cls:.4f}")
    else:
        print(f"{class_names[cls]}: No valid pixels in ground truth.")

overall_mean = np.mean(mean_iou_per_image)
overall_std = np.std(mean_iou_per_image)
print(f"\nOverall mean IoU: Mean = {overall_mean:.4f}, Std = {overall_std:.4f}")
```

OUTPUT:

```
Starting Checking image&label pair val list...
Checking image&label pair val list done!
Fixed image path for sample 0
Fixed image path for sample 1
Fixed image path for sample 2
Fixed image path for sample 3
Fixed image path for sample 4
Fixed image path for sample 5
```

.....

```
Evaluating on 500 validation samples...
Processed 50 samples...
Processed 100 samples...
Processed 150 samples...
Processed 200 samples...
Processed 250 samples...
Processed 300 samples...
Processed 350 samples...
Processed 400 samples...
Processed 450 samples...
Processed 500 samples...

Per-class IoU statistics:
```

```
Per-class IoU:
unlabeled: Mean IoU = 0.0000, Std = 0.0000
ego vehicle: Mean IoU = 0.9348, Std = 0.0473
rectification border: Mean IoU = 0.0000, Std = 0.0000
out of roi: Mean IoU = 0.0000, Std = 0.0000
static: Mean IoU = 0.2106, Std = 0.2070
dynamic: Mean IoU = 0.1149, Std = 0.2184
ground: Mean IoU = 0.0708, Std = 0.1701
road: Mean IoU = 0.9396, Std = 0.1889
sidewalk: Mean IoU = 0.5308, Std = 0.3158
parking: Mean IoU = 0.0983, Std = 0.1906
rail track: Mean IoU = 0.0287, Std = 0.1075
building: Mean IoU = 0.7666, Std = 0.2168
wall: Mean IoU = 0.1587, Std = 0.2669
fence: Mean IoU = 0.1197, Std = 0.2207
guard rail: Mean IoU = 0.0000, Std = 0.0000
bridge: Mean IoU = 0.0846, Std = 0.2065
tunnel: Mean IoU = 0.0000, Std = 0.0000
pole: Mean IoU = 0.3922, Std = 0.1971
polegroup: Mean IoU = 0.0492, Std = 0.1218
traffic light: Mean IoU = 0.3697, Std = 0.3006
traffic sign: Mean IoU = 0.4998, Std = 0.2862
vegetation: Mean IoU = 0.8501, Std = 0.1734
terrain: Mean IoU = 0.2470, Std = 0.3013
sky: Mean IoU = 0.7609, Std = 0.3048
person: Mean IoU = 0.4377, Std = 0.3280
rider: Mean IoU = 0.3434, Std = 0.3029
car: Mean IoU = 0.8232, Std = 0.2246
truck: Mean IoU = 0.2436, Std = 0.3574
bus: Mean IoU = 0.4173, Std = 0.4230
caravan: Mean IoU = 0.0827, Std = 0.1909
trailer: Mean IoU = 0.0000, Std = 0.0000
train: Mean IoU = 0.1333, Std = 0.2949
motorcycle: Mean IoU = 0.1746, Std = 0.2912
bicycle: Mean IoU = 0.3768, Std = 0.3106
license plate: No valid pixels in ground truth.

Overall mean IoU: Mean = 0.5085, Std = 0.0975
```