КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
МЕХАНІКО-МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА АЛГЕБРИ І КОМП'ЮТЕРНОЇ МАТЕМАТИКИ

Освітній ступінь: магістр

за спеціальністю 111 - математика
за освітніми програмами - математика

кваліфікаційна робота на ступінь магістра математики
на тему "Застосування машинного навчання у задачі знаходження
відповідностей для стерео пари"

студента 2 курсу магістратури
Швеця Максима Сергійовича

Допущений до захисту в ЕК
Протокол №11 засідання кафедри
алгебри і комп'ютерної математики
від 21 квітня 2020 року

Науковий керівник
Доктор фізико-математичний наук
Лавренюк Я. В.

Київ-2020

Зміст

1 Огляд

В цій роботі розглянемо задачу стереобачення та метод її розв'зання за допомогою машинного навчання запропонований в статті "Efficient Deep Learning for Stereo Matching" [2]. Модифікуємо цей метод та перевіримо його ефективність на датасеті KITTI.

2 Стереобачення

Задача стереобачення полягає в знаходженні інформації про тривимірні об'єкти використовуючи їх зображення. Зазвичай розглядають два зображення отримані з двох камер розташованих на одній горизонтальній осі, саме з такими данними працює алгоритм описаний в цій роботі.

Маючи два зображення однієї сцени можна зрозуміти наскільки далеко розташовані від камери зображені об'єкти спостерігаючи за зміною їх положення на зображеннях. Чим сильніше змінюється положення об'єкта між зображеннями, тим ближчий він до камери. Приклад цього можна побачити на рис. 1 взятому із датасету MIddlebury [3].

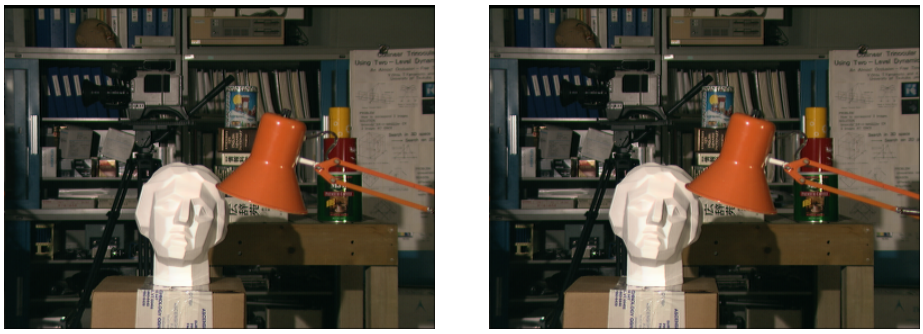


Рис. 1: Настільна лампа розташована ближче до камери і тому ссуває-ться сильніше ніж, наприклад, голова чи стіл

Якщо для деякої точки сцени відомі відповідні їй точки на зображеннях, то ми можемо розрахувати глибину цієї точки за формулою (рис. 2).

$$h = \frac{fb}{d}$$

де f - це фокусна відстань, b - відстань між камерами, d - різниця між кординатами точок зображенньь. Також, можна розрахувати відстань від

оптичної вісі камери за формулою

$$r = \frac{bx_r}{d}$$

де x_r - відстань точки від центра зображення

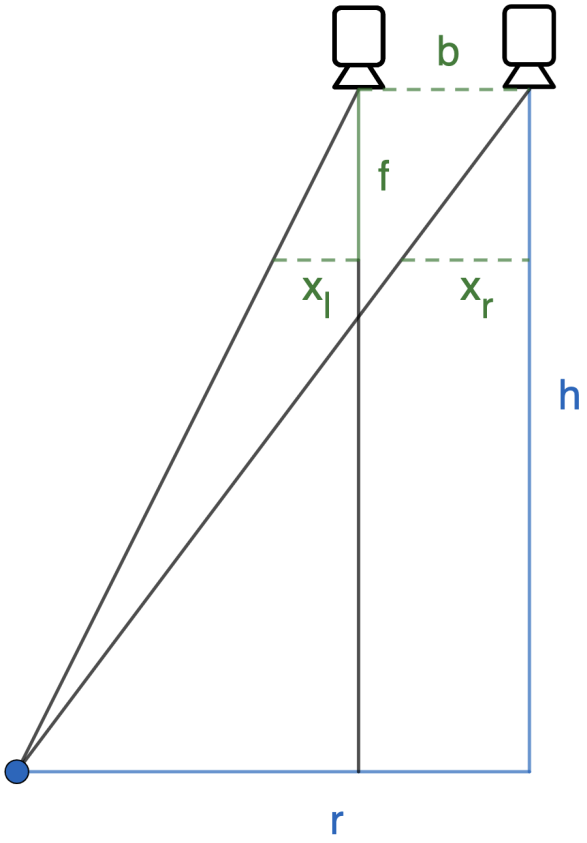


Рис. 2: Відомі нам значення (відстань між камерами b, фокусна відстань f, координати точок на зображеннях x_l, x_r позначено синім. Значення які можемо знайти (глибина зображення - h, відстань від вісі камери - r) позначено зеленим)

Отже, маючи алгоритм який співставляє точкам одного зображення відповідні їм точки іншого зображення ми можемо знаходити відстані до точок сцени. Зручний спосіб представити ці відстані - чорно-біле зображення в якому кожен піксель тим світліший чим більше значення d його

зсуву на іншому зображенні. Це зображення називають **мапою зсувів** (англ. disparity map). За попередньої форму видно що зсув оберненопропорційний глибині, тому об'єкти на такому зображені будуть тим світліші чим ближче вони до камери. Мапу зсувів для стереопари зображенної на рис. 1 можна побачити на рис. 3.

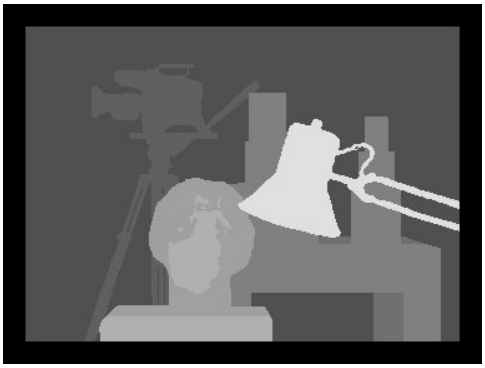


Рис. 3: Лампа знаходиться ближче до камери тому зсув пікселів на яких зображена лампа вищий, а отже на малюнку вони світліші.

Розгянутий алгоритм приймає два зображення і обчислює мапу зсувів.

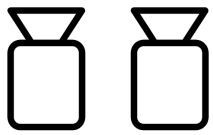
2.1 Виправлення зображеннь

Знаходити відповідні пікселі легше за все коли камери зміщені лише горизонтально і направлені однаково. При такому налаштуванні пікселі будуть зсуватись між зображеннями лише по горизонталі, а отже ми можемо шукати відповідні пікселі лише в одному вимірі.

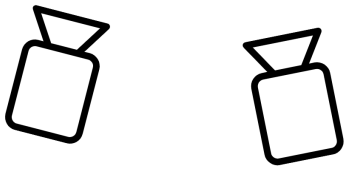
У випадках коли цих ідеальних умов неможливо досягти, до зображень можна застосувати процедуру виправлення (англ. rectification), яка полягає в проектуванні зображеннь на одну площину. Результат застосування такої процедури можна побачити на рис. 5.

2.2 Релевантність

Задача стереобачення важлива в таких областях як роботехніка, об'ємна відбудова (англ. 3D scene reconstruction), безпілотне вождіння та побудова доповненої реальності.



(а) Зручне розташування



(б) Незручне розташування

3 Алгоритм

3.1 Базовий алгоритм

Типовий алгоритм розв'язання задачі стереобачення починає з підрахунку cost-функції для кожного можливого значення зсуву. Ми хочемо щоб cost-функція мала низьке значення для зсувів близьких до реального і велике значення для зсувів далеких від реального.

Приклад. *Cost-функцію можна визначити як суму абсолютних різниць.*

$$Cost(x_0, y_0, d) = \sum_{(x,y)\in W(x_0,y_0)} \left|I^L(x,y) - I^R(x-d,y)\right|$$

Тут $I^L(x,y), I^R(x,y)$ інтенсивності пікселів з координатами (x,y) на лівому та правому малюнку відповідно, а $W(x,y)$ окіл пікселя (x,y). Отже, ця функція порівнює інтенсивності пікселів в околах (x_0,y_0) і (x_0-d,y_0). Якщо ці пікселі відповідають одній і тій самій 3D точці, то інтенсивності в їх околах скоріш за все будуть майже однаковими і значення функції буде відносно низьким.

Простий вибір тих зсувів для яких значення cost-функції найбільш низьке зазвичай приводить до поганих результатів (як ми побачимо пізніше), тому результати обчислення cost-функції додатково оброблюються. Методи обробки не залежать від способу підрахунку cost-функції,

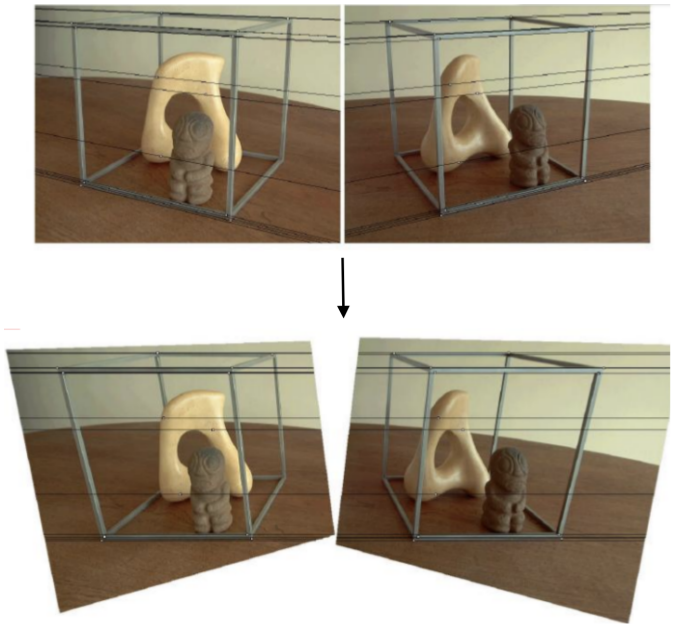


Рис. 5: Приклад виправлення зображень

для розглянутого алгоритму використані добре відомимі методи які буде наведено пізніше.

З появою великих датасетів таких як KITTI и Middleburry, в яких для різних стереопар доступна справжня карта зсувів (отримана за допомогою LIDAR чи структурованого світла), стало можливим застосування машинного навчання для того щоб обчислювати cost-функцію. Ми використовуємо датасет KITTI щоб натренувати нейронну мережу що буде розраховувати cost-функцію.

3.2 Структура нейронної мережі

На рис. 6 зображенно структуру нейронної мережі. Далі наведемо визначення використаних блоків:

- *ConvBNReLu(in, out, w, h)* - поєднання декількох інших блоків: $Convolution(in, out, w, h) \rightarrow BatchNormalization(out, 0.001) \rightarrow ReLU$
- *Convolution(in, out, w, h)* - приймає тензор з вимірами $in \times x \times y$, виконує *out* згорток з $in \times w \times h$ тензором і повертає результат у

вигляді $out \times x^* \times y^*$ тензора, де

$$x^* = x - w + 1,\ y^* = y - h + 1$$

- *BatchNormalization(in, esp)* - приймає тензор з вимірами $in \times x \times y$ і повертає тензор з такими самими вимірами. Детальніше про цей шар можна почитати в статі [1].
- *ReLU* - застосовує $max(0, x)$ до кожного елемента тензора
- *DotProduct* - підраховує скалярний добуток тензорів
- *LogSoftMax* - приймає вектор $(x_i)_{i=1}^n$ і розраховує

$$\left(log\left(\frac{e^{x_i}}{S}\right)\right)_{i=1}^n$$

де $S = \sum_{i=1}^n e^{x_i}$

Позначення $5*ConvBNReLU$ значить блок $ConvBNReLU$ застосований 5 разів.

3.3 Застосування нейронної мережі

Щоб застосувати нейронну мережу до зображень зі стереопари, зображення подаються у вигляді тензорів з вимірами $c \times w \times h$, де $c = 1$ якщо зображення чорно-біле (один канал кольору) і $= 3$ якщо зображення кольорове (три канали кольору). Будемо позначати ці тензори $P^L_{c,w,h}$ і $P^R_{c,w,h}$ для правого і лівого зображення відповідно. Також, оскільки кожен згорточний шар нейронної мережі зменшує ширину та висоту тензора, до зображень додається прослойка так щоб на виході ширина та висота тензора була рівна ширині та висоті зображення. Отже, в нашому випадку в нейронну мережу передаються тензори $P^L_{c,w+36,h+36}, P^R_{c,w+36,h+36}$, бо кожен з 9 згорткових шарів нейронної мережі зменшує виміри на 4, і на виході отримуються тензори

$$O^L_{64,w,h} := (o^l_{i,j,k})^{64,w,h}_{i=1,j=1,k=1}$$

$$O^R_{64,w,h} := (o^r_{i,j,k})^{64,w,h}_{i=1,j=1,k=1}$$

Cost-функція розраховується за формулою:

$$Cost(x, y, d) = -\langle O^L(x, y), O^R(x - d, y)\rangle$$

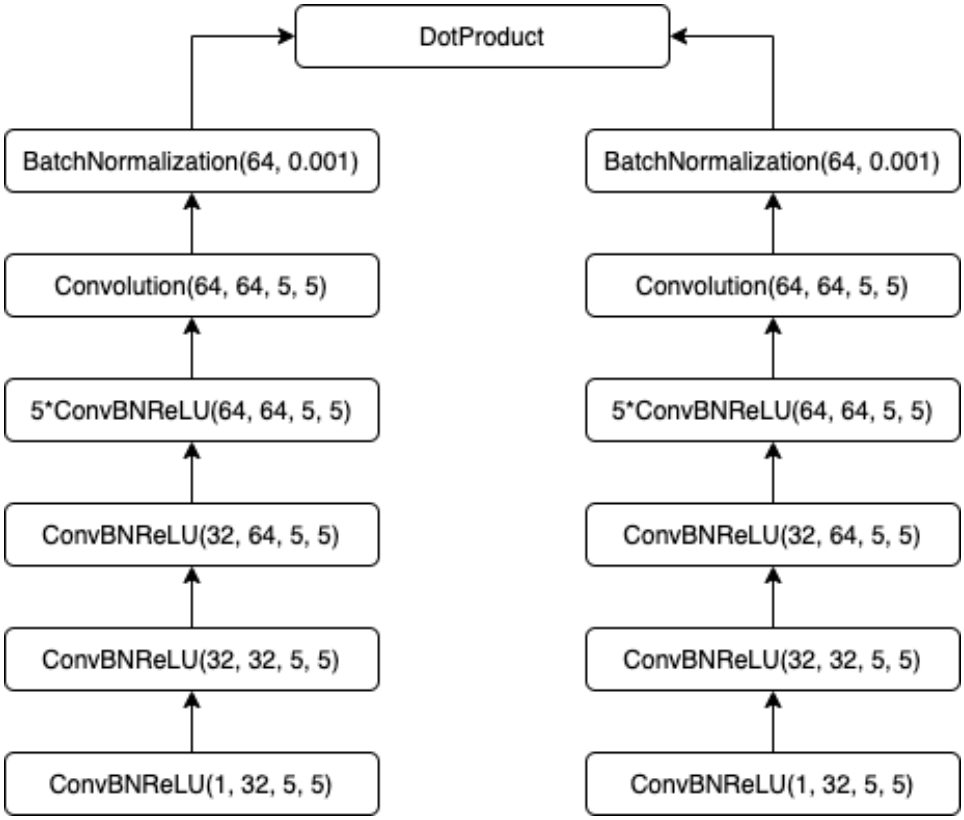


Рис. 6: Структура нейронної мережі

де $O^L(x,y) = (o^l_{i,x,y})_{i=1}^{64}$, $O^R(x,y) = (o^r_{i,x,y})_{i=1}^{64}$, а $\langle\cdot,\cdot\rangle$ позначає скалярний добуток.

Як інтерпретувати цю cost-функцію? Кожна гілка нейронної мережі знаходить деякі характерні риси в частині малюнка навколо пікселя. Які саме риси буде шукати мережа визначається під час тренування, оскільки ми використовуємо однакові параметри для обох гілок, вони будуть шукати однакові риси. Кожне з 64 чисел на позиції (x,y) показує наскільки властива одна з 64 рис частині малюнка навколо (x,y). Отже, кожна i-та компонента вектора $O^L(x,y)$ показує наскільки властива цій частині зображення i-та риса, так само для правого зображення і вектора $O^R(x,y)$. Чим більш схожі риси описані в цих векторах, тим більшим буде скалярний добуток і тим менше буде cost-функція

Якщо для кожного пікселя (x,y) просто взяти те значення d для якого значення cost-функції найменше, отримаємо досить поганий результат

(див. рис. 7), тому результати нейронної мережі додадтково оброблюються. Як саме буде описано далі.

Рис. 7: Приклад результатів без застосування згладжування

4 Тренування

Чи буде мережа добре обчислювати cost-функцію залежить від значень параметрів в її згорткових блоках. Параметрами згорткового блока є числа в тензорах з якими він виконує згортку.

Приклад. *Блок Convolution*$(32, 64, 5, 5)$ *виконує* 64 *згортки з різними тензорами вимірів* $32 \times 5 \times 5$. *Результатом кожної згортки є матриці* $x^* \times y^*$ *з яких формується результат. Числа в кожному з тензорів є параметрами цього блоку. Отже, блок Convolution*$(32, 64, 5, 5)$ *має* $64 * 32 * 5 * 5 = 51200$ *параметрів. Вся нейронна мережа містить* 693536 *параметрів*

Очевидно, кількість параметрів занадто велика для того щоб обрати їх значення вручну. Тому значення параметрів обираються програматично, спираючись на набір стереопар для яких відомі мапи зсувів. Узагальнене описання алгоритму тренування таке:

1. Ініціалізуемо параметри випадковими значеннями
2. Підраховуємо функцію витрат для поточних значень параметрів.
3. Підраховуємо градієнт функції витрат для поточних параметрів
4. Змінюємо параметри у напрямі протилежному напряму градієнту
5. Повторюємо кроки 2-4 доки не знайдемо мінімум функції витрат

Функція витрат оцінює наскільки сильно результати роботи мережі відрізнются від справжніх результатів (які нам наперед відомі). Чим більша різниця між отриманими та справжніми результатами, тим більше значення функції витрат для параметрів. Її значення зазвичай підраховується шляхом застосування нейронної мережі і порівняння її результатів з наперед відомими правильними результатами. Для того щоб підрахувати градієнт функції витрат зазвичай використовують алгоритм backpropagation. Далі опишемо процес тренування більш детально.

4.1 Побудова датасету для тренування

Введемо деякі позначення. Будемо позначати $W^L_{w,h}(x, y)$ вікно розміру $w \times h$ з центром в пікселі (x, y). $W^R_{w,h}(x, y)$ буде означати те саме для правого зображення.

Для того щоб побудувати датасет на якому тренується нейронна мережа використано данні з датасету KITTI, який містить приблизно 200 стереопар для яких відомі значення зсувів. З кожної стереопари, для тих пікселів лівого зображення (x, y) для яких відоме значення зсуву d, добирається пара $W^L_{37,37}(x, y)$, $W^R_{37+2D,h}(x - d, y)$, де D - максимальне абсолютне значення зсуву. Всі такі пари і складають остаточний датасет.

4.2 Підхід до тренування

При тренуванні задача стереобачення розглядається як задача класифікаціЇ. Тобто, структура нейронної мережї дещо змінюється так, щоб на виході вона давала вектор довжини D, в якому i-а компонента містить ймовірність того що значення зсуву рівне i. Далі детальніше опишемо ці зміни.

На відміну від нейронної мережі яка використовується на реальних данних, під час тренування результати гілок об'єднуються матричним добутком. Оскільки при обробці $W^L_{37,37}(x, y)$ лівою гілкою отримуємо вектор з 64 значень, а при обробці $W^R_{37+2D,h}(x - d, y)$ отримуємо матрицю $D \times 64$, то матричний добуток дає вектор з D значень. До цього вектора застосовується $LogSoftMax$ функція (визначення якої наведено вище), що й обчислює шукані ймовірності.

Оскільки єдина части нейронної мережі що має параметри це її гілки, то така зміна структури не впливає на роботу остаточної нейронної мережі.

4.3 Функція витрат

В якості функції витрат використовується

$$L(x, d_t) = -x_{d_t}$$

де, x - вектор ймовірностей розрахований нейронною мережею, d_t - справжнє значення зсуву.

Тобто, щоб порахувати функцію витрат для заданого набору параметрів, нейронна мережа спочатку застосовується до стереопари, а потім береться ймовірність порахована для справжнього значення зсуву зі знаком мінус.

5 Згладжування

Як ми вже побачили, однієї лише нейронної мережі недостатньо для того щоб отримати якісну мапу зсувів, тому результати нейронної мережі проходять пост-процессинг. Для пост-процессингу використовуємо методи детально описані в статі [4]. Тут надамо лише їх поверхневий опис.

5.1 Cost агрегація

Для кожного пікселя рахує середнє значеннь cost-функції в околі розміру 5×5. Зауважимо, що середнє підраховується окремо для кожного значення зсуву.

5.2 Перехрестна cost агрегація

Цей метод спочатку будує окіл для кожного пікселя уникаючи того щоб околи переходили через края об'єктів на зображенні. Щоб досягти цього, побудова околу починається з одного пікселя, до якого додаються сусідні пікселі доки їх інтенсивність і відстань до початкового пікселя знаходится в заданих межах.

Маючи околи, метод рахує середнє значень cost-функцій в них. Цей крок повторюється чотири рази

5.3 Semiglobal matching

Визначає енергетичний функціонал який залежить від мапи зсувів і штрафує мапи з великими значеннями cost-функцій та мапи в яких значення зсувів для сусідніх пікселів сильно відрізняються. Цей функціонал мінімізується у декількох напрямках.

5.4 Поєднання методів

Наведені вище методи поєднуються наступним чином:

- Використовується cost агрегація
- Використосується перехрестна cost агрегація
- Використовується semiglobal matching
- Використосується перехрестна cost агрегація

Після наведенних вище кроків до отриманні значення додатково оброблюються методами описанними в [4]

5.5 Результати згладжування

На рис. 8 можна побачити результати застосування згладжування до прикладу з рис. 7

Рис. 8: Результати після застосування згладжування

Література

[1] Sergey Ioffe та Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[2] W. Luo, A. Schwing та R. Urtasun. "Efficient Deep Learning for Stereo Matching". в: *International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[3] D. Scharstein та R. Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". в: *International Journal of Computer Vision* 47.11 (2002), с. 7—42.

[4] Jure Zbontar та Yann LeCun. "Stereo matching by training a convolutional neural network to compare image patches". в: *Journal of Machine Learning Research* 17 (2016), с. 1—32.

Додаток 1

Процедура вибору данних для тренувального датасету

```
function D = read_disparities(filename)
  % loads disparity map D from png file

  I = imread(filename);
  D = double(I)/256;
  D(I==0) = -1;
end

function [patches_data] = extract_patches(
  images_dir, images_ids, half_patch_size, half_disparity_range
)
num_available_values = 0

for i = 1:length(image_ids)
  id = images_ids[i];

  filename = sprintf('%s/disp_noc_0/%06d_10.png', images_dir, id);
  disparity_map = read_disparities(fn);
  num_available_values = num_available_values ...
                         + sum(disparity_map(:)~=-1);
end

patches_data = zeros(4, length(images_ids))

for i = 1:length(image_ids)
  id = images_ids[i];

  filename = sprintf('%s/disp_noc_0/%06d_10.png', images_dir, id);
  disparity_map = read_disparities(fn);
  [row, col] = find(disparity_map~=-1);
  [height, width] = size(disparity_map);

  for pixel_index = 1:length(row)
    left_patch_center_x = col(pixel_index);
    left_patch_center_y = row(pixel_index);
```

```
      disparity = disparity_map(
        left_patch_center_x, left_patch_center_y
      );

      right_patch_center_x = round(left_patch_center_x - disparity);
      right_patch_center_y = left_patch_center_y;

      left_patch_fits = ...
        left_patch_center_x + half_patch_size <= width ...
        && left_patch_center_x - half_patch_size > 0   ...
        && left_patch_center_y + half_patch_size <= height ...
        && left_patch_center_y - half_patch_size > 0;

      half_right_patch_size = half_disparity_range + half_patch_size;
      right_patch_fits = ...
        right_patch_center_x - half_right_patch_size > 0 ...
        && right_patch_center_x + half_right_patch_size <= width ...
        && right_patch_center_y - half_patch_size > 0 ...
        && right_patch_center_y + half_patch_size <= height;

      if ll && rr_type1
        patches_data(:,i) = [
          id; left_patch_center_x;
          left_patch_center_y; right_patch_center_x
        ];
      end
    end
  end
```