

Вопрос 1

Верно

Баллов: 1,00 из 1,00

На этом уроке мы с вами реализуем прямой проход сверточного слоя, обратный проход и расчет производных мы трогать не будем.

Вспомним как работает сверточный слой:

- на вход подается массив изображений, еще он называется батчем
- к каждому изображению по границам добавляются нули
- по каждому изображению "скользит" каждый из фильтров сверточного слоя

Давайте начнем с разминки - реализуем функцию, добавляющую padding.

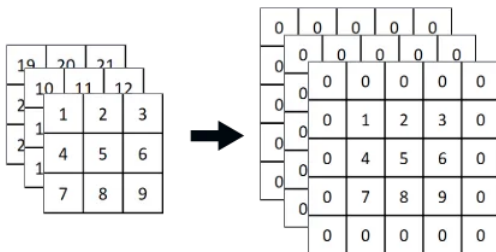
Пусть у нас есть батч `input_images` из двух изображений с тремя каналами (RGB). Размер изображений пусть будет 3×3 . Вспомним, что вход сверточного слоя имеет следующую размерность:

- размер батча
- число каналов
- высота
- ширина

В рассматриваемом случае размерность входа (2, 3, 3, 3).

Если мы добавим вокруг каждого изображения отступ из одного нуля, то размер каждого изображений станет $3+2 \times 1 = 5$ пикселей в ширину и 5 в высоту соответственно (добавляем по одному нулю с каждой стороны изображения).

Напишите любую работающую реализацию.



Ответ: (штрафной режим: 0 %)

Сброс ответа

```
1 import torch
2
3 # Создаем входной массив из двух изображений RGB 3*3
4 input_images = torch.tensor(
5     [[[0, 1, 2],
6        [3, 4, 5],
7        [6, 7, 8]],
8
9        [[9, 10, 11],
10         [12, 13, 14],
11         [15, 16, 17]],
12
13         [[18, 19, 20],
14          [21, 22, 23],
15          [24, 25, 26]]],
16
17         [[[27, 28, 29],
18          [30, 31, 32],
19          [33, 34, 35]],
20
21          [[36, 37, 38],
22           [39, 40, 41],
23           [42, 43, 44]],
24
25           [[45, 46, 47],
26            [48, 49, 50],
27            [51, 52, 53]]]])
28
29
30
31 def get_padding2d(input_images):
32     padded_images = torch.nn.functional.pad(input_images.float(), (1, 1, 1, 1), mode='constant', value=0)
33     return padded_images
34
35
36 correct_padded_images = torch.tensor(
37     [[[0., 0., 0., 0., 0.],
38        [0., 0., 1., 2., 0.],
39        [0., 3., 4., 5., 0.],
40        [0., 6., 7., 8., 0.],
41        [0., 0., 0., 0., 0.]],
42
43         [[[0., 0., 0., 0., 0.],
```

```
44         [0., 9., 10., 11., 0.],
45         [0., 12., 13., 14., 0.],
46         [0., 15., 16., 17., 0.],
47         [0., 0., 0., 0., 0.]],
48
49     [[0., 0., 0., 0., 0.],
50      [0., 18., 19., 20., 0.],
51      [0., 21., 22., 23., 0.],
52      [0., 24., 25., 26., 0.],
53      [0., 0., 0., 0., 0.]]],
54
55
56     [[0., 0., 0., 0., 0.],
57      [0., 27., 28., 29., 0.],
58      [0., 30., 31., 32., 0.],
59      [0., 33., 34., 35., 0.],
60      [0., 0., 0., 0., 0.]],
61
62     [[0., 0., 0., 0., 0.],
63      [0., 36., 37., 38., 0.],
64      [0., 39., 40., 41., 0.],
65      [0., 0., 0., 0., 0.]]],
66
67     [[0., 0., 0., 0., 0.],
68      [0., 42., 43., 44., 0.],
69      [0., 45., 46., 47., 0.],
70      [0., 48., 49., 50., 0.],
71      [0., 0., 0., 0., 0.]]],
72
73     [[0., 0., 0., 0., 0.],
74      [0., 51., 52., 53., 0.],
75      [0., 54., 55., 56., 0.],
76      [0., 57., 58., 59., 0.],
77      [0., 0., 0., 0., 0.]]],
78
79     [[0., 0., 0., 0., 0.],
80      [0., 60., 61., 62., 0.],
81      [0., 63., 64., 65., 0.],
82      [0., 66., 67., 68., 0.],
83      [0., 0., 0., 0., 0.]]],
84
85     [[0., 0., 0., 0., 0.],
86      [0., 69., 70., 71., 0.],
87      [0., 72., 73., 74., 0.],
88      [0., 75., 76., 77., 0.],
89      [0., 0., 0., 0., 0.]]],
90
91     [[0., 0., 0., 0., 0.],
92      [0., 78., 79., 80., 0.],
93      [0., 81., 82., 83., 0.],
94      [0., 84., 85., 86., 0.],
95      [0., 0., 0., 0., 0.]]],
96
97     [[0., 0., 0., 0., 0.],
98      [0., 87., 88., 89., 0.],
99      [0., 90., 91., 92., 0.],
100     [0., 93., 94., 95., 0.],
101     [0., 0., 0., 0., 0.]]],
102
103     [[0., 0., 0., 0., 0.],
104      [0., 96., 97., 98., 0.],
105      [0., 99., 100., 101., 0.],
106      [0., 102., 103., 104., 0.],
107      [0., 0., 0., 0., 0.]]],
108
109     [[0., 0., 0., 0., 0.],
110      [0., 105., 106., 107., 0.],
111      [0., 108., 109., 110., 0.],
112      [0., 111., 112., 113., 0.],
113      [0., 0., 0., 0., 0.]]],
114
115     [[0., 0., 0., 0., 0.],
116      [0., 114., 115., 116., 0.],
117      [0., 117., 118., 119., 0.],
118      [0., 120., 121., 122., 0.],
119      [0., 0., 0., 0., 0.]]],
120
121     [[0., 0., 0., 0., 0.],
122      [0., 123., 124., 125., 0.],
123     [0., 126., 127., 128., 0.],
124     [0., 129., 130., 131., 0.],
125     [0., 0., 0., 0., 0.]]],
126
127     [[0., 0., 0., 0., 0.],
128      [0., 132., 133., 134., 0.],
129      [0., 135., 136., 137., 0.],
130      [0., 138., 139., 140., 0.],
131      [0., 0., 0., 0., 0.]]],
132
133     [[0., 0., 0., 0., 0.],
134      [0., 141., 142., 143., 0.],
135     [0., 144., 145., 146., 0.],
136     [0., 147., 148., 149., 0.],
137     [0., 0., 0., 0., 0.]]],
138
139     [[0., 0., 0., 0., 0.],
140      [0., 150., 151., 152., 0.],
141     [0., 153., 154., 155., 0.],
142     [0., 156., 157., 158., 0.],
143     [0., 0., 0., 0., 0.]]],
144
145     [[0., 0., 0., 0., 0.],
146      [0., 159., 160., 161., 0.],
147     [0., 162., 163., 164., 0.],
148     [0., 165., 166., 167., 0.],
149     [0., 0., 0., 0., 0.]]],
150
151     [[0., 0., 0., 0., 0.],
152      [0., 168., 169., 170., 0.],
153     [0., 171., 172., 173., 0.],
154     [0., 174., 175., 176., 0.],
155     [0., 0., 0., 0., 0.]]],
156
157     [[0., 0., 0., 0., 0.],
158      [0., 177., 178., 179., 0.],
159     [0., 180., 181., 182., 0.],
160     [0., 183., 184., 185., 0.],
161     [0., 0., 0., 0., 0.]]],
162
163     [[0., 0., 0., 0., 0.],
164      [0., 186., 187., 188., 0.],
165     [0., 189., 190., 191., 0.],
166     [0., 192., 193., 194., 0.],
167     [0., 0., 0., 0., 0.]]],
168
169     [[0., 0., 0., 0., 0.],
170      [0., 195., 196., 197., 0.],
171     [0., 198., 199., 200., 0.],
172     [0., 201., 202., 203., 0.],
173     [0., 0., 0., 0., 0.]]],
174
175     [[0., 0., 0., 0., 0.],
176      [0., 204., 205., 206., 0.],
177     [0., 207., 208., 209., 0.],
178     [0., 210., 211., 212., 0.],
179     [0., 0., 0., 0., 0.]]],
180
181     [[0., 0., 0., 0., 0.],
182      [0., 213., 214., 215., 0.],
183     [0., 216., 217., 218., 0.],
184     [0., 219., 220., 221., 0.],
185     [0., 0., 0., 0., 0.]]],
186
187     [[0., 0., 0., 0., 0.],
188      [0., 222., 223., 224., 0.],
189     [0., 225., 226., 227., 0.],
190     [0., 228., 229., 230., 0.],
191     [0., 0., 0., 0., 0.]]],
192
193     [[0., 0., 0., 0., 0.],
194      [0., 231., 232., 233., 0.],
195     [0., 234., 235., 236., 0.],
196     [0., 237., 238., 239., 0.],
197     [0., 0., 0., 0., 0.]]],
198
199     [[0., 0., 0., 0., 0.],
200      [0., 240., 241., 242., 0.],
201     [0., 243., 244., 245., 0.],
202     [0., 246., 247., 248., 0.],
203     [0., 0., 0., 0., 0.]]],
204
205     [[0., 0., 0., 0., 0.],
206      [0., 249., 250., 251., 0.],
207     [0., 252., 253., 254., 0.],
208     [0., 255., 256., 257., 0.],
209     [0., 0., 0., 0., 0.]]],
210
211     [[0., 0., 0., 0., 0.],
212      [0., 258., 259., 260., 0.],
213     [0., 261., 262., 263., 0.],
214     [0., 264., 265., 266., 0.],
215     [0., 0., 0., 0., 0.]]],
216
217     [[0., 0., 0., 0., 0.],
218      [0., 267., 268., 269., 0.],
219     [0., 270., 271., 272., 0.],
220     [0., 273., 274., 275., 0.],
221     [0., 0., 0., 0., 0.]]],
222
223     [[0., 0., 0., 0., 0.],
224      [0., 276., 277., 278., 0.],
225     [0., 279., 280., 281., 0.],
226     [0., 282., 283., 284., 0.],
227     [0., 0., 0., 0., 0.]]],
228
229     [[0., 0., 0., 0., 0.],
230      [0., 285., 286., 287., 0.],
231     [0., 288., 289., 290., 0.],
232     [0., 291., 292., 293., 0.],
233     [0., 0., 0., 0., 0.]]],
234
235     [[0., 0., 0., 0., 0.],
236      [0., 294., 295., 296., 0.],
237     [0., 297., 298., 299., 0.],
238     [0., 300., 301., 302., 0.],
239     [0., 0., 0., 0., 0.]]],
240
241     [[0., 0., 0., 0., 0.],
242      [0., 303., 304., 305., 0.],
243     [0., 306., 307., 308., 0.],
244     [0., 309., 310., 311., 0.],
245     [0., 0., 0., 0., 0.]]],
246
247     [[0., 0., 0., 0., 0.],
248      [0., 312., 313., 314., 0.],
249     [0., 315., 316., 317., 0.],
250     [0., 318., 319., 320., 0.],
251     [0., 0., 0., 0., 0.]]],
252
253     [[0., 0., 0., 0., 0.],
254      [0., 321., 322., 323., 0.],
255     [0., 324., 325., 326., 0.],
256     [0., 327., 328., 329., 0.],
257     [0., 0., 0., 0., 0.]]],
258
259     [[0., 0., 0., 0., 0.],
260      [0., 330., 331., 332., 0.],
261     [0., 333., 334., 335., 0.],
262     [0., 336., 337., 338., 0.],
263     [0., 0., 0., 0., 0.]]],
264
265     [[0., 0., 0., 0., 0.],
266      [0., 339., 340., 341., 0.],
267     [0., 342., 343., 344., 0.],
268     [0., 345., 346., 347., 0.],
269     [0., 0., 0., 0., 0.]]],
270
271     [[0., 0., 0., 0., 0.],
272      [0., 348., 349., 350., 0.],
273     [0., 351., 352., 353., 0.],
274     [0., 354., 355., 356., 0.],
275     [0., 0., 0., 0., 0.]]],
276
277     [[0., 0., 0., 0., 0.],
278      [0., 357., 358., 359., 0.],
279     [0., 360., 361., 362., 0.],
280     [0., 363., 364., 365., 0.],
281     [0., 0., 0., 0., 0.]]],
282
283     [[0., 0., 0., 0., 0.],
284      [0., 366., 367., 368., 0.],
285     [0., 369., 370., 371., 0.],
286     [0., 372., 373., 374., 0.],
287     [0., 0., 0., 0., 0.]]],
288
289     [[0., 0., 0., 0., 0.],
290      [0., 375., 376., 377., 0.],
291     [0., 378., 379., 380., 0.],
292     [0., 381., 382., 383., 0.],
293     [0., 0., 0., 0., 0.]]],
294
295     [[0., 0., 0., 0., 0.],
296      [0., 384., 385., 386., 0.],
297     [0., 387., 388., 389., 0.],
298     [0., 390., 391., 392., 0.],
299     [0., 0., 0., 0., 0.]]],
300
301     [[0., 0., 0., 0., 0.],
302      [0., 393., 394., 395., 0.],
303     [0., 396., 397., 398., 0.],
304     [0., 399., 400., 401., 0.],
305     [0., 0., 0., 0., 0.]]],
306
307     [[0., 0., 0., 0., 0.],
308      [0., 402., 403., 404., 0.],
309     [0., 405., 406., 407., 0.],
310     [0., 408., 409., 410., 0.],
311     [0., 0., 0., 0., 0.]]],
312
313     [[0., 0., 0., 0., 0.],
314      [0., 411., 412., 413., 0.],
315     [0., 414., 415., 416., 0.],
316     [0., 417., 418., 419., 0.],
317     [0., 0., 0., 0., 0.]]],
318
319     [[0., 0., 0., 0., 0.],
320      [0., 420., 421., 422., 0.],
321     [0., 423., 424., 425., 0.],
322     [0., 426., 427., 428., 0.],
323     [0., 0., 0., 0., 0.]]],
324
325     [[0., 0., 0., 0., 0.],
326      [0., 429., 430., 431., 0.],
327     [0., 432., 433., 434., 0.],
328     [0., 435., 436., 437., 0.],
329     [0., 0., 0., 0., 0.]]],
330
331     [[0., 0., 0., 0., 0.],
332      [0., 438., 439., 440., 0.],
333     [0., 441., 442., 443., 0.],
334     [0., 444., 445., 446., 0.],
335     [0., 0., 0., 0., 0.]]],
336
337     [[0., 0., 0., 0., 0.],
338      [0., 447., 448., 449., 0.],
339     [0., 450., 451., 452., 0.],
340     [0., 453., 454., 455., 0.],
341     [0., 0., 0., 0., 0.]]],
342
343     [[0., 0., 0., 0., 0.],
344      [0., 456., 457., 458., 0.],
345     [0., 459., 460., 461., 0.],
346     [0., 462., 463., 464., 0.],
347     [0., 0., 0., 0., 0.]]],
348
349     [[0., 0., 0., 0., 0.],
350      [0., 465., 466., 467., 0.],
351     [0., 468., 469., 470., 0.],
352     [0., 471., 472., 473., 0.],
353     [0., 0., 0., 0., 0.]]],
354
355     [[0., 0., 0., 0., 0.],
356      [0., 474., 475., 476., 0.],
357     [0., 477., 478., 479., 0.],
358     [0., 480., 481., 482., 0.],
359     [0., 0., 0., 0., 0.]]],
360
361     [[0., 0., 0., 0., 0.],
362      [0., 483., 484., 485., 0.],
363     [0., 486., 487., 488., 0.],
364     [0., 489., 490., 491., 0.],
365     [0., 0., 0., 0., 0.]]],
366
367     [[0., 0., 0., 0., 0.],
368      [0., 492., 493., 494., 0.],
369     [0., 495., 496., 497., 0.],
370     [0., 498., 499., 500., 0.],
371     [0., 0., 0., 0., 0.]]],
372
373     [[0., 0., 0., 0., 0.],
374      [0., 501., 502., 503., 0.],
375     [0., 504., 505., 506., 0.],
376     [0., 507., 508., 509., 0.],
377     [0., 0., 0., 0., 0.]]],
378
379     [[0., 0., 0., 0., 0.],
380      [0., 510., 511., 512., 0.],
381     [0., 513., 514., 515., 0.],
382     [0., 516., 517., 518., 0.],
383     [0., 0., 0., 0., 0.]]],
384
385     [[0., 0., 0., 0., 0.],
386      [0., 519., 520., 521., 0.],
387     [0., 522., 523., 524., 0.],
388     [0., 525., 526., 527., 0.],
389     [0., 0., 0., 0., 0.]]],
390
391     [[0., 0., 0., 0., 0.],
392      [0., 528., 529., 530., 0.],
393     [0., 531., 532., 533., 0.],
394     [0., 534., 535., 536., 0.],
395     [0., 0., 0., 0., 0.]]],
396
397     [[0., 0., 0., 0., 0.],
398      [0., 537., 538., 539., 0.],
399     [0., 540., 541., 542., 0.],
400     [0., 543., 544., 545., 0.],
401     [0., 0., 0., 0., 0.]]],
402
403     [[0., 0., 0., 0., 0.],
404      [0., 546., 547., 548., 0.],
405     [0., 549., 550., 551., 0.],
406     [0., 552., 553., 554., 0.],
407     [0., 0., 0., 0., 0.]]],
408
409     [[0., 0., 0., 0., 0.],
410      [0., 555., 556., 557., 0.],
411     [0., 558., 559., 560., 0.],
412     [0., 561., 562., 563., 0.],
413     [0., 0., 0., 0., 0.]]],
414
415     [[0., 0., 0., 0., 0.],
416      [0., 564., 565., 566., 0.],
417     [0., 567., 568., 569., 0.],
418     [0., 570., 571., 572., 0.],
419     [0., 0., 0., 0., 0.]]],
420
421     [[0., 0., 0., 0., 0.],
422      [0., 573., 574., 575., 0.],
423     [0., 576., 577., 578., 0.],
424     [0., 579., 580., 581., 0.],
425     [0., 0., 0., 0., 0.]]],
426
427     [[0., 0., 0., 0., 0.],
428      [0., 582., 583., 584., 0.],
429     [0., 585., 586., 587., 0.],
430     [0., 588., 589., 590., 0.],
431     [0., 0., 0., 0., 0.]]],
432
433     [[0., 0., 0., 0., 0.],
434      [0., 591., 592., 593., 0.],
435     [0., 594., 595., 596., 0.],
436     [0., 597., 598., 599., 0.],
437     [0., 0., 0., 0., 0.]]],
438
439     [[0., 0., 0., 0., 0.],
440      [0., 600., 601., 602., 0.],
441     [0., 603., 604., 605., 0.],
442     [0., 606., 607., 608., 0.],
443     [0., 0., 0., 0., 0.]]],
444
445     [[0., 0., 0., 0., 0.],
446      [0., 609., 610., 611., 0.],
447     [0., 612., 613., 614., 0.],
448     [0., 615., 616., 617., 0.],
449     [0., 0., 0., 0., 0.]]],
450
451     [[0., 0., 0., 0., 0.],
452      [0., 618., 619., 620., 0.],
453     [0., 621., 622., 623., 0.],
454     [0., 624., 625., 626., 0.],
455     [0., 0., 0., 0., 0.]]],
456
457     [[0., 0., 0., 0., 0.],
458      [0., 627., 628., 629., 0.],
459     [0., 630., 631., 632., 0.],
460     [0., 633., 634., 635., 0.],
461     [0., 0., 0., 0., 0.]]],
462
463     [[0., 0., 0., 0., 0.],
464      [0., 636., 637., 638., 0.],
465     [0., 639., 640., 641., 0.],
466     [0., 642., 643., 644., 0.],
467     [0., 0., 0., 0., 0.]]],
468
469     [[0., 0., 0., 0., 0.],
470      [0., 645., 646., 647., 0.],
471     [0., 648., 649., 650., 0.],
472     [0., 651., 652., 653., 0.],
473     [0., 0., 0., 0., 0.]]],
474
475     [[0., 0., 0., 0., 0.],
476      [0., 654., 655., 656., 0.],
477     [0., 657., 658., 659., 0.],
478     [0., 660., 661., 662., 0.],
479     [0., 0., 0., 0., 0.]]],
480
481     [[0., 0., 0., 0., 0.],
482      [0., 663., 664., 665., 0.],
483     [0., 666., 667., 668., 0.],
484     [0., 669., 670., 671., 0.],
485     [0., 0., 0., 0., 0.]]],
486
487     [[0., 0., 0., 0., 0.],
488      [0., 672., 673., 674., 0.],
489     [0., 675., 676., 677., 0.],
490     [0., 678., 679., 680., 0.],
491     [0., 0., 0., 0., 0.]]],
492
493     [[0., 0., 0., 0., 0.],
494      [0., 681., 682., 683., 0.],
495     [0., 684., 685., 686., 0.],
496     [0., 687., 688., 689., 0.],
497     [0., 0., 0., 0., 0.]]],
498
499     [[0., 0., 0., 0., 0.],
500      [0., 690., 691., 692., 0.],
501     [0., 693., 694., 695., 0.],
502     [0., 696., 697., 698., 0.],
503     [0., 0., 0., 0., 0.]]],
504
505     [[0., 0., 0., 0., 0.],
506      [0., 699., 700., 701., 0.],
507     [0., 702., 703., 704., 0.],
508     [0., 705., 706., 707., 0.],
509     [0., 0., 0., 0., 0.]]],
510
511     [[0., 0., 0., 0., 0.],
512      [0., 708., 709., 710., 0.],
513     [0., 711., 712., 713., 0.],
514     [0., 714., 715., 716., 0.],
515     [0., 0., 0., 0., 0.]]],
516
517     [[0., 0., 0., 0., 0.],
518      [0., 717., 718., 719., 0.],
519     [0., 720., 721., 722., 0.],
520     [0., 723., 724., 725., 0.],
521     [0., 0., 0., 0., 0.]]],
522
523     [[0., 0., 0., 0., 0.],
524      [0., 726., 727., 728., 0.],
525     [0., 729., 730., 731., 0.],
526     [0., 732., 733., 734., 0.],
527     [0., 0., 0., 0., 0.]]],
528
529     [[0., 0., 0., 0., 0.],
530      [0., 735., 736., 737., 0.],
531     [0., 738., 739., 740., 0.],
532     [0., 741., 742., 743., 0.],
533     [0., 0., 0., 0., 0.]]],
534
535     [[0., 0., 0., 0., 0.],
536      [0., 744., 745., 746., 0.],
537     [0., 747., 748., 749., 0.],
538     [0., 750., 751., 752., 0.],
539     [0., 0., 0., 0., 0.]]],
540
541     [[0., 0., 0., 0., 0.],
542      [0., 753., 754., 755., 0.],
543     [0., 756., 757., 758., 0.],
544     [0., 759., 760., 761., 0.],
545     [0., 0., 0., 0., 0.]]],
546
547     [[0., 0., 0., 0., 0.],
548      [0., 762., 763., 764., 0.],
549     [0., 765., 766., 767., 0.],
550     [0., 768., 769., 770., 0.],
551     [0., 0., 0., 0., 0.]]],
552
553     [[0., 0., 0., 0., 0.],
554      [0., 771., 772., 773., 0.],
555     [0., 774., 775., 776., 0.],
556     [0., 777., 778., 779., 0.],
557     [0., 0., 0., 0., 0.]]],
558
559     [[0., 0., 0., 0., 0.],
560      [0., 780., 781., 782., 0.],
561     [0., 783., 784., 785., 0.],
562     [0., 786., 787., 788., 0.],
563     [0., 0., 0., 0., 0.]]],
564
565     [[0., 0., 0., 0., 0.],
566      [0., 789., 790., 791., 0.],
567     [0., 792., 793., 794., 0.],
568     [0., 795., 796., 797., 0.],
569     [0., 0., 0., 0., 0.]]],
570
571     [[0., 0., 0., 0., 0.],
572      [0., 798., 799., 800., 0.],
573     [0., 801., 802., 803., 0.],
574     [0., 804., 805., 806., 0.],
575     [0., 0., 0., 0., 0.]]],
576
577     [[0., 0., 0., 0., 0.],
578      [0., 807., 808., 809., 0.],
579     [0., 810., 811., 812., 0.],
580     [0., 813., 814., 815., 0.],
581     [0., 0., 0., 0., 0.]]],
582
583     [[0., 0., 0., 0., 0.],
584      [0., 816., 817., 818., 0.],
585     [0., 819., 820., 821., 0.],
586     [0., 822., 823., 824., 0.],
587     [0., 0., 0., 0., 0.]]],
588
589     [[0., 0., 0., 0., 0.],
590      [0., 825., 826., 827., 0.],
591     [0., 828., 829., 830., 0.],
592     [0., 831., 832., 833., 0.],
593     [0., 0., 0., 0., 0.]]],
594
595     [[0., 0., 0., 0., 0.],
596      [0., 834., 835., 836., 0.],
597     [0., 837., 838., 839., 0.],
598     [0., 840., 841., 842., 0.],
599     [0., 0., 0., 0., 0.]]],
600
601     [[0., 0., 0., 0., 0.],
602      [0., 843., 844., 845., 0.],
603     [0., 846., 847., 848., 0.],
604     [0., 849., 850., 851., 0.],
605     [0., 0., 0., 0., 0.]]],
606
607     [[0., 0., 0., 0., 0.],
608      [0., 852., 853., 854., 0.],
609     [0., 855., 856., 857., 0.],
610     [0., 858., 859., 860., 0.],
611     [0., 0., 0., 0., 0.]]],
612
613     [[0., 0., 0., 0., 0.],
614      [0., 861., 862., 863., 0.],
615     [0., 864., 865., 866., 0.],
616     [0., 867., 868., 869., 0.],
617     [0., 0., 0., 0., 0.]]],
618
619     [[0., 0., 0., 0., 0.],
620      [0., 870., 871., 872., 0.],
621     [0., 873., 874., 875., 0.],
622     [0., 876., 877., 878., 0.],
623     [0., 0., 0., 0., 0.]]],
624
625     [[0., 0., 0., 0., 0.],
626      [0., 879., 880., 881., 0.],
627     [0., 882., 883., 884., 0.],
628     [0., 885., 886., 887., 0.],
629     [0., 0., 0., 0., 0.]]],
630
631     [[0., 
```

Вопрос 2

Верно

Баллов: 1,00 из 1,00

На этом шаге детально рассмотрим из чего состоит сверточный слой.

Сверточный слой это массив фильтров.

Каждый фильтр имеет следующую размерность:

- число слоев во входном изображении (для RGB это 3)
- высота фильтра
- ширина фильтра

В ядре (кernels) все фильтры имеют одинаковые размерность, поэтому ширину и высоту фильтров называют шириной и высотой ядра. Чаще всего ширина ядра равна высоте ядра, в таком случае их называют размером ядра (kernel_size).

Также слой имеет такие параметры:

- padding - на какое количество пикселей увеличивать входное изображение с каждой стороны.
- stride - на сколько пикселей смещается фильтр при вычислении свертки

Попробуйте самостоятельно вывести формулу размерности выхода сверточного слоя, зная параметры входа и ядра.

Правильность формулы проверьте, сравнив ее с формулой из [документации](#).

Чтобы убедиться в правильности вашей формулы, напишите функцию, принимающую на вход:

- входную размерность (число изображений в батче*число слоев в одном изображении*высота изображения*ширина изображения)
- количество фильтров
- размер фильтров (считаем, что высота совпадает с шириной)
- padding
- stride

Функция должна возвращать размерность выхода.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
1 import numpy as np
2
3
4 def calc_out_shape(input_matrix_shape, out_channels, kernel_size, stride, padding):
5     batch_size, channels, height, width = input_matrix_shape
6     out_height = int((height - kernel_size + 2 * padding) / stride) + 1
7     out_width = int((width - kernel_size + 2 * padding) / stride) + 1
8     out_shape = [batch_size, out_channels, out_height, out_width]
9     return out_shape
10
11 print(np.array_equal(
12     calc_out_shape(input_matrix_shape=[2, 3, 10, 10],
13                     out_channels=10,
14                     kernel_size=3,
15                     stride=1,
16                     padding=0),
17     [2, 10, 8, 8]))
18
19 # ... и ещё несколько подобных кейсов
```

Прошли все тесты! ✓

Верно

Баллы за эту попытку: 1,00/1,00.

В этом и следующих шагах мы реализуем сверточный слой различными способами.

Тестировать правильность реализации мы будем, сравнивая результаты работы с выходом [сверточного слоя из PyTorch](#).

Для удобства все наши реализации оформим в виде классов. Интерфейс классов сделаем одинаковым и максимально похожим на интерфейс стандартной реализации.

Тестировать наши реализации слоя будем одной и той же функцией.

*Паддинг вход вы уже умеете - будем считать, что padding = 0.

В этом шаге вам предлагается изучить код для проверки.

```
import torch
from abc import ABC, abstractmethod

# абстрактный класс для сверточного слоя
class ABCConv2d(ABC):
    def __init__(self, in_channels, out_channels, kernel_size, stride):
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.kernel_size = kernel_size
        self.stride = stride

    def set_kernel(self, kernel):
        self.kernel = kernel

    @abstractmethod
    def __call__(self, input_tensor):
        pass

# класс-обертка над torch.nn.Conv2d для унификации интерфейса
class Conv2d(ABCConv2d):
    def __init__(self, in_channels, out_channels, kernel_size, stride):
        self.conv2d = torch.nn.Conv2d(in_channels, out_channels, kernel_size,
                                       stride, padding=0, bias=False)

    def set_kernel(self, kernel):
        self.conv2d.weight.data = kernel

    def __call__(self, input_tensor):
        return self.conv2d(input_tensor)

# функция, создающая объект класса cls и возвращающая свертку от input_matrix
def create_and_call_conv2d_layer(conv2d_layer_class, stride, kernel, input_matrix):
    out_channels = kernel.shape[0]
    in_channels = kernel.shape[1]
    kernel_size = kernel.shape[2]

    layer = conv2d_layer_class(in_channels, out_channels, kernel_size, stride)
    layer.set_kernel(kernel)

    return layer(input_matrix)

# Функция, тестирующая класс conv2d_cls.
# Возвращает True, если свертка совпадает со сверткой с помощью torch.nn.Conv2d.
def test_conv2d_layer(conv2d_layer_class, batch_size=2,
                      input_height=4, input_width=4, stride=2):
    kernel = torch.tensor(
        [
            [[0., 1, 0],
             [1, 2, 1],
             [0, 1, 0]],
            [[1, 2, 1],
             [0, 3, 3],
             [0, 1, 10]],
            [[10, 11, 12],
             [13, 14, 15],
             [16, 17, 18]]]
    )

    in_channels = kernel.shape[1]

    input_tensor = torch.arange(0, batch_size * in_channels *
                                input_height * input_width,
                                out=torch.FloatTensor()) \
        .reshape(batch_size, in_channels, input_height, input_width)

    custom_conv2d_out = create_and_call_conv2d_layer(
        conv2d_layer_class, stride, kernel, input_tensor)
    conv2d_out = create_and_call_conv2d_layer(
```

```
        Conv2d, stride, kernel, input_tensor)

    return torch.allclose(custom_conv2d_out, conv2d_out) \
        and (custom_conv2d_out.shape == conv2d_out.shape)

print(test_conv2d_layer(Conv2d))
```

Вопрос 3

Верно

Баллов: 1,00 из 1,00

Переиспользуем код с предыдущего шага для проверки своей реализации сверточного слоя.

Рассмотрим свертку батча из одного однослойного изображения 3×3 с ядром из одного фильтра 2×2 , $\text{stride} = 1$, то есть, на выходе должна получиться одна матрица 2×2 . Строго записанная размерность выхода равна (1 - изображений в батче, 1 - количество фильтров в ядре, 2 - высота матрицы выхода, 2 - ширина матрицы выхода).

Пусть W - веса ядра, X - вход, Y - выход.

W0	W1
W2	W3

 *

X0	X1	X2
X3	X4	X5
X6	X7	X8

 =

Y0	Y1
Y2	Y3

Вычислить выход можно в цикле:

W0	W1
W2	W3

 *

X0	X1	X2
X3	X4	X5
X6	X7	X8

 =

Y0	

W0	W1
W2	W3

 *

X0	X1	X2
X3	X4	X5
X6	X7	X8

 =

Y0	Y1

W0	W1
W2	W3

 *

X0	X1	X2
X3	X4	X5
X6	X7	X8

 =

Y0	Y1
Y2	

W0	W1
W2	W3

 *

X0	X1	X2
X3	X4	X5
X6	X7	X8

 =

Y0	Y1
Y2	Y3

На каждой итерации цикла фильтр умножается попиксельно на часть изображения, а потом 4 получившиеся числа складываются - получается один пиксель выхода.

Требуемое количество итераций для данного случая - 4, так как может быть 2 положения ядра и 2 по вертикали, общее число итераций - произведение количеств положений, то есть в данном случае $2 \times 2 = 4$.

Давайте перейдем от простого случая к общему.

- Если бы изображение было многослойным, например трехслойное - RGB, значит, фильтры в ядре тоже должны быть трехслойные. Каждый слой фильтра попиксельно умножается на соответствующий слой исходного изображения. То есть в данном случае после умножения получилось бы $4 \times 3 = 12$ произведений, результаты которых складываются, и получается значение выходного пикселя.
- Если бы фильтров в ядре было больше одного, то добавился бы внешний цикл по фильтрам, внутри которого мы считаем свертку для каждого фильтра.
- Если бы во входном батче было более 1 изображения, то добавился бы еще один внешний цикл по изображениям в батче.

Напоминание: во всех шагах этого урока мы считаем bias в сверточных слоях нулевым.

На этом шаге требуется реализовать сверточный слой через циклы.

Обратите внимание, что в коде рассматривается общий случай - батч на входе не обязательно состоит из одного изображения, в ядре несколько слоев.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
29 def __init__(self, in_channels, out_channels, kernel_size, stride):
30     self.conv2d = torch.nn.Conv2d(in_channels, out_channels, kernel_size,
31                                   stride, padding=0, bias=False)
32
33 def set_kernel(self, kernel):
34     self.conv2d.weight.data = kernel
35
36 def __call__(self, input_tensor):
37     return self.conv2d(input_tensor)
38
39
40 def create_and_call_conv2d_layer(conv2d_layer_class, stride, kernel, input_matrix):
41     out_channels = kernel.shape[0]
42     in_channels = kernel.shape[1]
43     kernel_size = kernel.shape[2]
44
45     layer = conv2d_layer_class(in_channels, out_channels, kernel_size, stride)
46     layer.set_kernel(kernel)
47
48     return layer(input_matrix)
49
50
51 def test_conv2d_layer(conv2d_layer_class, batch_size=2,
52                       inut height=4, inut width=4, stride=2):
```

```

53     kernel = torch.tensor(
54         [
55             [0., 1, 0],
56             [1, 2, 1],
57             [0, 1, 0]],
58         [
59             [1, 2, 1],
60             [0, 3, 3],
61             [0, 1, 10]],
62         [
63             [10, 11, 12],
64             [13, 14, 15],
65             [16, 17, 18]]])
66     in_channels = kernel.shape[1]
67
68     input_tensor = torch.arange(0, batch_size * in_channels *
69                                input_height * input_width,
70                                out=torch.FloatTensor()) \
71         .reshape(batch_size, in_channels, input_height, input_width)
72
73     custom_conv2d_out = create_and_call_conv2d_layer(
74         conv2d_layer_class, stride, kernel, input_tensor)
75     conv2d_out = create_and_call_conv2d_layer(
76         Conv2d, stride, kernel, input_tensor)
77
78     return torch.allclose(custom_conv2d_out, conv2d_out) \
79         and (custom_conv2d_out.shape == conv2d_out.shape)
80
81
82 # Сверточный слой через циклы.
83 class Conv2dLoop(ABCConv2d):
84     def __call__(self, input_tensor):
85         batch_size, _, input_height, input_width = input_tensor.shape
86         output_height = (input_height - self.kernel_size) // self.stride + 1
87         output_width = (input_width - self.kernel_size) // self.stride + 1
88         output_tensor = torch.zeros((batch_size, self.out_channels, output_height, output_width))
89
90         for b in range(batch_size):
91             for i in range(0, input_height - self.kernel_size + 1, self.stride):
92                 for j in range(0, input_width - self.kernel_size + 1, self.stride):
93                     for k in range(self.out_channels):
94                         output_tensor[b, k, i//self.stride, j//self.stride] = \
95                             (self.kernel[k] * input_tensor[b, :, i:i+self.kernel_size, j:j+self.kernel_size]).sum()
96         return output_tensor
97
98 # Корректность реализации определится в сравнении со стандартным слоем из pytorch.
99 # Проверка происходит автоматически вызовом следующего кода
100 # (раскомментируйте для самостоятельной проверки,
101 # в коде для сдачи задания должно быть закомментировано):
102 # print(test_conv2d_layer(Conv2dLoop))

```

Прошли все тесты! ✓

Верно

Баллы за эту попытку: 1,00/1,00.

Вопрос 4

Верно

Баллов: 1,00 из 1,00

Переиспользуем код с третьего шага для проверки своей реализации сверточного слоя.

Реализация через циклы очень неэффективна по производительности. Есть целых два способа сделать то же самое с помощью матричного умножения.

На этом шаге будет реализация первым из них.

Рассмотрим свертку одного одноканального изображения размером 4*4 пикселя (значения пикселей обозначены через X).

Сворачивать будем с ядром из одного фильтра размером 3*3, веса обозначены через W.

Для простоты примем stride = 1.

Тогда выход Y будет иметь размерность 1*1*2*2 (в данном случае на входе одно изображение - это первая единица в размерности, в ядре один фильтр - это вторая единица в размерности выхода).

W 0,0	W 0,1	W 0,2
W 1,0	W 1,1	W 1,2
W 2,0	W 2,1	W 2,2

 \times

X 0,0	X 0,1	X 0,2	X 0,3
X 1,0	X 1,1	X 1,2	X 1,3
X 2,0	X 2,1	X 2,2	X 2,3
X 3,0	X 3,1	X 3,2	X 3,3

 $=$

Y 0,0	Y 0,1
Y 1,0	Y 1,1

Оказывается, выход свертки можно получить умножением матриц, как показано ниже.

W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2	0	0	0	0	0
0	W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2	0	0	0	0
0	0	0	0	W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2	0
0	0	0	0	0	W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2

 \times

X 0,0
X 0,1
X 0,2
X 0,3
X 1,0
X 1,1
...
X 3,3

 $=$

Y 0,0
Y 0,1
Y 1,0
Y 1,1

Рекомендуем убедиться в этом, перемножив матрицы на листочке.

Давайте перейдем от простого случая к общему:

- Если фильтров в ядре больше одного. Заметим, что для каждого фильтра, матрица W' будет умножаться на один и тот же вектор изображения. Значит, можно сконкатенировать матрицы фильтров ядра по вертикали и за одно умножение получить ответ для всех фильтров.

W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2	0	0	0	0	0
0	W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2	0	0	0	0
0	0	0	0	W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2	0
0	0	0	0	0	W 0,0	W 0,1	W 0,2	0	W 1,0	W 1,1	W 1,2	0	W 2,0	W 2,1	W 2,2

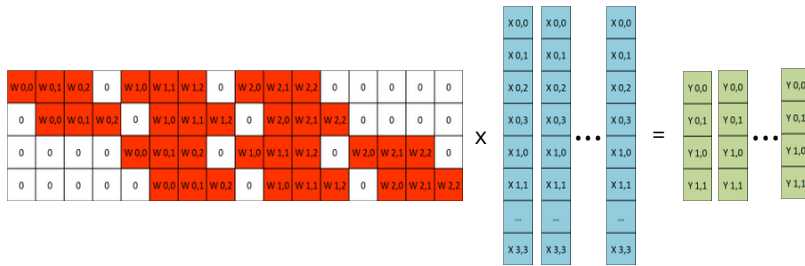
 \times

X 0,0
X 0,1
X 0,2
X 0,3
X 1,0
X 1,1
...
X 3,3

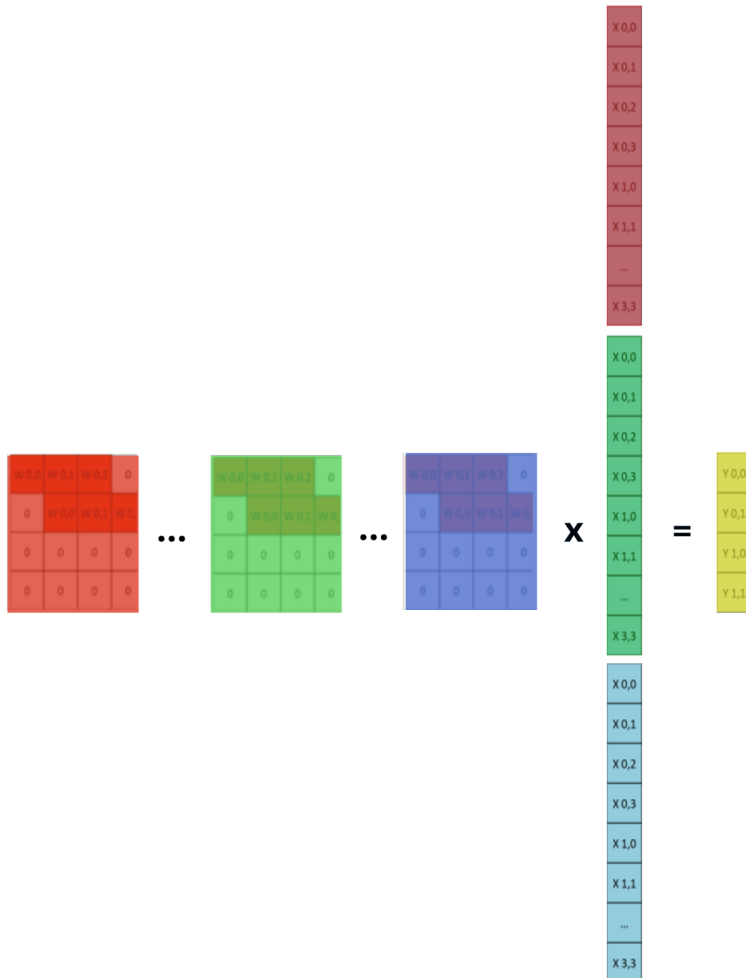
 $=$

Y 0,0
Y 0,1
Y 1,0
Y 1,1

- **Если на входе более одного изображения:** заметим, что матрица W' одинакова для всех изображений батча, то есть, можно каждое изображение вначале вытянуть в столбец, а затем эти столбцы для всех изображений батча сконкатенировать по горизонтали.



- **Если в изображении больше одного слоя,** вначале выполним преобразования входа и ядра для каждого слоя, а затем сконкатенируем: вектора разных слоев входа в один большой вектор, а матрицы ядра соответственно в одну длинную матрицу. И мы получим сложение от выходов по слоям в процессе перемножения матриц.



То есть даже в самом общем случае мы за одно умножение матриц можем получить ответ.

Но рассчитанный таким способом выход не совпадает по размерности с выходом стандартного слоя из PyTorch - нужно изменить размерность.

В коде уже реализовано:

- преобразование входного батча изображений
- умножение матрицы ядра на матрицу входа
- преобразование ответа

Напоминание: во всех шагах этого урока мы считаем bias в сверточных слоях нулевым.

Вам осталось реализовать преобразование ядра в описанный выше формат.

Обратите внимание, что в коде рассматривается общий случай - вход состоит из нескольких многослойных изображений, в ядре несколько слоев.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
1 import torch
2 from abc import ABC, abstractmethod
3
4
```



```

5 def calc_out_shape(input_matrix_shape, out_channels, kernel_size, stride, padding):
6     batch_size, channels_count, input_height, input_width = input_matrix_shape
7     output_height = (input_height + 2 * padding - (kernel_size - 1) - 1) // stride + 1
8     output_width = (input_width + 2 * padding - (kernel_size - 1) - 1) // stride + 1
9
10    return batch_size, out_channels, output_height, output_width
11
12
13 class ABCConv2d(ABC):
14     def __init__(self, in_channels, out_channels, kernel_size, stride):
15         self.in_channels = in_channels
16         self.out_channels = out_channels
17         self.kernel_size = kernel_size
18         self.stride = stride
19
20     def set_kernel(self, kernel):
21         self.kernel = kernel
22
23     @abstractmethod
24     def __call__(self, input_tensor):
25         pass
26
27
28 class Conv2d(ABCConv2d):
29     def __init__(self, in_channels, out_channels, kernel_size, stride):
30         self.conv2d = torch.nn.Conv2d(in_channels, out_channels, kernel_size,
31                                       stride, padding=0, bias=False)
32
33     def set_kernel(self, kernel):
34         self.conv2d.weight.data = kernel
35
36     def __call__(self, input_tensor):
37         return self.conv2d(input_tensor)
38
39
40 def create_and_call_conv2d_layer(conv2d_layer_class, stride, kernel, input_matrix):
41     out_channels = kernel.shape[0]
42     in_channels = kernel.shape[1]
43     kernel_size = kernel.shape[2]
44
45     layer = conv2d_layer_class(in_channels, out_channels, kernel_size, stride)
46     layer.set_kernel(kernel)
47
48     return layer(input_matrix)
49
50
51 def test_conv2d_layer(conv2d_layer_class, batch_size=2,
52

```

Прошли все тесты! ✓

Верно

Баллы за эту попытку: 1,00/1,00.