

卷積神經網路 (二) 從影像到tensor



山與水你和我

體系結構小白，影像處理小白

11 人贊同了該文章

號 目錄 收起

- 讀取到tensor
- DataLoader
- 數據增強
- 程式碼

上一篇

山與水你和我：卷積神經網路 (一) tensor 定義

定義了張量，這篇解決從影像到tensor。

讀取到tensor

給定一張影像，首先藉由影像庫讀取內容，然後轉換成float 或double 資料類型，依照tensor 的儲存順序存成 $C \times H \times W$ 。

我使用的是OpenCV 函式庫，imread 讀取影像之後，得到的是cv::Mat，資料型別是uchar (unsigned char，表示0-255)，影像內容儲存在Mat.data 中，每個位置依照BGR 的順序儲存三個值，因此可以定義Tensor 轉換的函數

```
// 从图像指针中读取
void Tensor3D::read_from_opencv_mat(const uchar* const img_ptr) {
    // 从 img_ptr 数据中获取图像内容
    const int length = H * W;
    const int length_2 = 2 * length;
    for(int i = 0; i < length; ++i) { // 二维平面每个坐标
        const int p = 3 * i; // 从 p 开始分别是 B, G, R 灰度值
        this->data[i] = img_ptr[p] * 1.f / 255;
        this->data[length + i] = img_ptr[p + 1] * 1.f / 255;
        this->data[length_2 + i] = img_ptr[p + 2] * 1.f / 255;
    }
}
```

其中img_ptr 是讀取影像得到的uchar 指針，影像一般預設3 通道



```
cv::Mat image = cv::imread("1.png");
Tensor3D example(3, image.rows, image.cols);
example.read_from_opencv_mat(image.data);
```

此時，資料從影像儲存到了Tensor，後面為了觀察資料是否儲存正確，也可以寫一個從Tensor 到影像的函數，

```
cv::Mat Tensor3D::opencv_mat(const int CH) const {
    cv::Mat origin;
    if(CH == 3) {
        origin = cv::Mat(H, W, CV_8UC3);
        const int length = H * W;
        for(int i = 0; i < length; ++i) {
            const int p = 3 * i;
            origin.data[p] = cv::saturate_cast<uchar>(255 * data[i]);
            origin.data[p + 1] = cv::saturate_cast<uchar>(255 * data[i + length]);
            origin.data[p + 2] = cv::saturate_cast<uchar>(255 * data[i + length + length]);
        }
    }
    else if(CH == 1) {
        origin = cv::Mat(H, W, CV_8UC1);
        const int length = H * W;
        for(int i = 0; i < length; ++i)
            origin.data[i] = cv::saturate_cast<uchar>(255 * data[i]);
    }
    return origin;
}
```

目前沒有對影像內容做歸一化，一般會對BGR 影像資料做平均值{0.406, 0.456, 0.485}、變異數{0.225, 0.224, 0.229}的歸一化。

DataLoader

上面說明如何依照Tensor 的排列順序儲存一張影像，得到 $C \times H \times W$ 。

一般為了穩定更新參數，會採用batch 的形式，一組資料包含了幾張圖象的訊息，得到 $B \times C \times H \times W$ 。

.....

每個資料夾下方分別是1000 張影像。

遍歷資料夾，取得影像路徑以及對應的標籤，目標是得到類似下面這種結構

"train" : [("./bird/1.png", 2), (./panda/20.png", 1), (./dog/5.png", 0),]

"valid" : [("./dog/1.png", 0), (./bird/7.png", 2), (./panda/9.png", 1),]

"test" : [("./dog/3.png", 0), (./panda/12.png", 1), (./bird/15.png", 2),]

```
using list_type = std::vector<std::pair<std::string, int> >;

std::map<std::string, pipeline::list_type> pipeline::get_images_for_classification(
    const std::filesystem::path dataset_path,
    const std::vector<std::string> categories={"dog", "panda", "bird"},
    const std::pair<float, float> ratios={0.8, 0.1}) {
    // 遍历 dataset_path 文件夹下指定的类别
    list_type all_images_list;
    const int categories_num = categories.size();
    for(int i = 0; i < categories_num; ++i) {
        const auto images_dir = dataset_path / categories[i];
        assert(std::filesystem::exists(images_dir) and std::string(images_dir.string())
        auto walker = std::filesystem::directory_iterator(images_dir);
        for(const auto& iter : walker)
            all_images_list.emplace_back(iter.path().string(), i);
    }
    // 打乱图像列表
    std::shuffle(all_images_list.begin(), all_images_list.end(), std::default_random_e
    // 将数据集划分成三部分
    const int total_size = all_images_list.size();
    assert(ratios.first > 0 and ratios.second > 0 and ratios.first + ratios.second < 1
    const int train_size = int(total_size * ratios.first);
```

```

results.emplace("train", list_type(all_images_list.begin(), all_images_list.begin(
results.emplace("test", list_type(all_images_list.begin() + train_size, all_images
results.emplace("valid", list_type(all_images_list.begin() + train_size + test_siz
std::cout << "train : " << results["train"].size() << "\n" << "test : " << re
return results;
}

```

三個資料集需要分別定義讀取器，以train 為例，需要定義一些超參數，類似batch_size、是否打亂、種子等，如下面的偽代碼

```

class DataLoader {
private:
    list_type images_list; // 数据集列表, image <==> Label
    int images_num;        // 这个子数据集一共有多少张图像和对应的标签
    const int batch_size;  // 每次打包几张图家
    const bool augment;    // 是否要做图像增强
    const bool shuffle;    // 是否要打乱列表
    const int seed;        // 每次随机打乱列表的种子
};

// 数据集路径、指定的类别
const std::filesystem::path dataset_path("../datasets/animals");
const std::vector<std::string> categories({"dog", "panda", "bird"});

// 获取图片
auto dataset = pipeline::get_images_for_classification(dataset_path, categories);

// 构造数据流
DataLoader train_loader(dataset["train"], 4, false, true, ...);

```

之後，明確要提供什麼資料用於訓練或測試 - 1 個batch 的Tensor 指標數組
std::vector<tensor>，以及對應的類別標籤

```
using batch_type = std::pair< std::vector<tensor>, std::vector<int> >;
```

逐次從images_list 中抽取B 張影像，每張影像讀取內容儲存到Tensor 裡，做若干變換之後，返回tensor

```
// 返回一个 batch 的训练/测试数据
```

```

std::vector<int> labels;
images.reserve(this->batch_size);
labels.reserve(this->batch_size);
for(int i = 0; i < this->batch_size; ++i) {
    auto sample = this->add_to_buffer(i); // 获取一个样本, 分别放到 images, labels
    images.emplace_back(sample.first);
    labels.emplace_back(sample.second);
}
return std::make_pair(std::move(images), std::move(labels));
}

// 获取第 batch_index 的图像信息, 填充成 tensor
std::pair<tensor, int> DataLoader::add_to_buffer(const int batch_index) {
    // 获取图像序号
    ++this->iter;
    if(this->iter == this->images_num) { // 取到头了, 重新开始
        this->iter = 0; // 下标置为 0
        if(this->shuffle) { // 然后再一次打乱列表
            std::shuffle(this->images_list.begin(), this->images_list.end(), std::defa
        }
    }
    // 读取图像
    const auto& image_path = this->images_list[this->iter].first;
    const int image_label = this->images_list[this->iter].second;
    cv::Mat origin = cv::imread(image_path);
    // 对图像做预处理
    if(this->augment) {};
    // resize, 必须在数据增强之后
    cv::resize(origin, origin, {W, H});
    // 直接对 buffer 进行填充, 将图像转化成 float 数据, 且是 Tensor 形式, C x H x W;
    this->buffer[batch_index]->read_from_opencv_mat(origin.data);
    // 返回图像内容和标签
    return std::pair<tensor, int>(this->buffer[batch_index], image_label);
}

```

細節：

- 因為C++ 中迭代器實作有點麻煩, 就簡單做了個循環, 所有映像都取了一次之後, 從頭開始取, 如果需要打亂順序, 從頭開始取時對images_list 做shuffle 處理。
- 因為要組成batch, 一般在圖像分類中, 圖像都會resize 成一樣大小, 在這裡是固定為 224\times 224 \times 3。

的空間，每次直接從影像填滿到buffer 中即可。

現在來個測試，可以從每個batch 的tensor 恢復成圖像數據，展示，這是深度學習流程中絕不能錯的一步，數據不能錯。

```
void test_dataloader() {
    std::setbuf(stdout, 0);

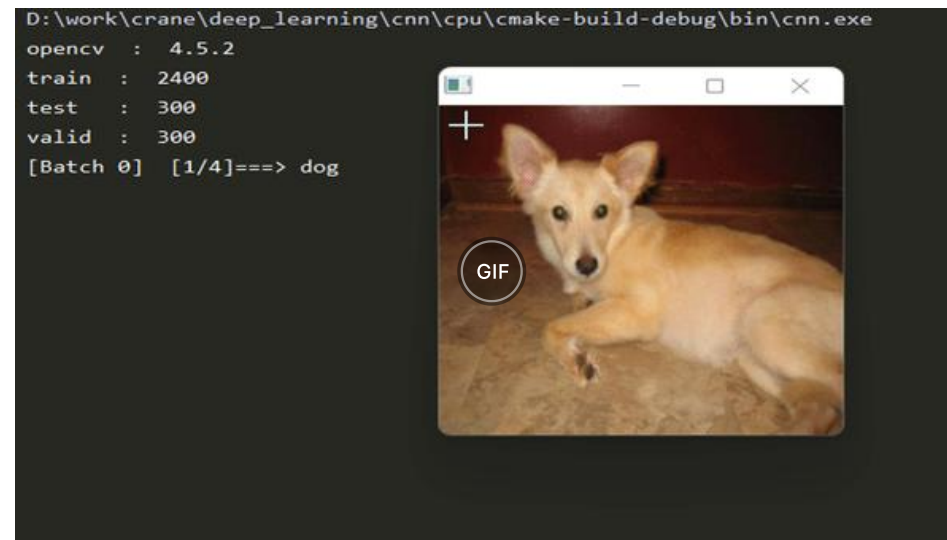
    using namespace architectures;

    // 指定一些参数
    const int train_batch_size = 4;
    const std::tuple<int, int, int> image_size({224, 224, 3});
    const std::filesystem::path dataset_path("../datasets/animals");
    const std::vector<std::string> categories({"dog", "panda", "bird"});

    // 获取图片
    auto dataset = pipeline::get_images_for_classification(dataset_path, categories);

    // 构造数据流
    pipeline::DataLoader train_loader(dataset["train"], train_batch_size, false, true,

    // 开始不断获取 batch
    for(int i = 0; i < 10; ++i) {
        // 获取一个 batch
        auto sample = train_loader.generate_batch();
        // 拆成 tensor 和 类别序号
        const auto& images = sample.first;
        const auto& labels = sample.second;
        // 从 tensor 恢复成 opencv::Mat 格式
        for(int b = 0; b < train_batch_size; ++b) {
            std::cout << "[Batch " << i << " ] " << " [" << b + 1 << "/" << train_batch_size << " ] " << std::endl;
            const auto origin = images[b]->opencv_mat(3);
            cv_show(origin);
        }
    }
}
```



數據增強

在深度學習中，為了減弱過擬合，增強模型的泛化能力，通常會對輸入做資料增強，影像的話，做翻轉、旋轉、裁剪、mask 等操作。

水平鏡像翻轉

豎直鏡像翻轉

旋轉15°

為了增加隨機性，本人設定了一個操作列表[hflip, vflip, crop, rotate]，每個操作都有一個機率p，以p的機率執行，預設的機率是[0.5, 0.2, 0.7, 0.5]；而且每次資料增強，都會先把操作清單打亂，具體操作寫成了一個類

```
class ImageAugmentor {
private:
    std::default_random_engine e, l, c, r; // e 用来获得操作的概率; l 用来打乱操作列表
    std::uniform_real_distribution<float> engine; // 随机数生成器
    std::uniform_real_distribution<float> crop_engine; // 裁剪操作的随机数生成器
    std::uniform_real_distribution<float> rotate_engine; // 旋转操作的随机数生成器
    std::uniform_int_distribution<int> minus_engine; // 角度是否为负数
    std::vector<std::pair<std::string, float> > ops; // 操作列表
public:
    ImageAugmentor(const std::vector<std::pair<std::string, float> >&
        _ops={{"hflip", 0.5}, {"vflip", 0.2}, {"crop", 0.7}, {"rotate", 0.5}})
        : e(212), l(826), c(320), r(520),
```

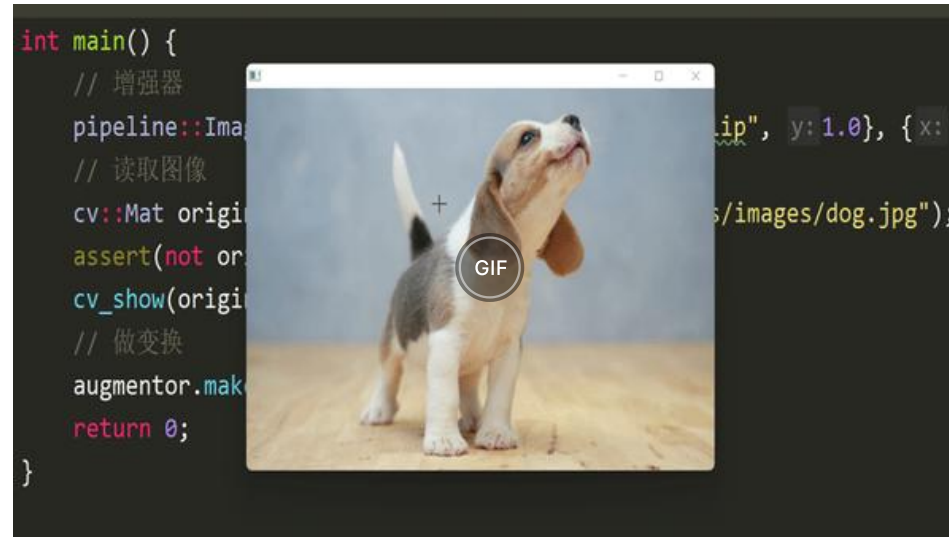
```

make_augment(cv::Mat& origin, const bool show) {
    // 首先打乱操作的顺序
    std::shuffle(ops.begin(), ops.end(), this->l);
    // 遍历这个操作
    for(const auto& item : ops) {
        // 获得一个随机概率
        const float prob = engine(e);
        // 如果概率足够大, 执行操作
        if(prob >= 1.0 - item.second) {
            // 镜像翻转
            if(item.first == "hflip")
                cv::flip(origin, origin, 1);
            else if(item.first == "vflip")
                cv::flip(origin, origin, 0);
            else if(item.first == "crop") {
                // 获取图像信息
                const int H = origin.rows;
                const int W = origin.cols;
                // 获取截取的比例
                float crop_ratio = 0.7f + crop_engine(c);
                const int _H = int(H * crop_ratio);
                const int _W = int(W * crop_ratio);
                // 获取截取的位置
                std::uniform_int_distribution<int> _H_pos(0, H - _H);
                std::uniform_int_distribution<int> _W_pos(0, W - _W);
                // 开始截取图像
                origin = origin(cv::Rect(_W_pos(c), _H_pos(c), _W, _H)).clone();
            }
            else if(item.first == "rotate") {
                // 获取一个随即角度
                float angle = rotate_engine(r);
                if(minus_engine(r) & 1) angle = -angle;
                origin = rotate(origin, angle);
            }
            if(show == true) cv_show(origin);
        }
    }
};

```

头文件: 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832. 833. 834. 835. 836. 837. 838. 839. 840. 841. 842. 843. 844. 845. 846. 847. 848. 849. 850. 851. 852. 853. 854. 855. 856. 857. 858. 859. 860. 861. 862. 863. 864. 865. 866. 867. 868. 869. 870. 871. 872. 873. 874. 875. 876. 877. 878. 879. 880. 881. 882. 883. 884. 885. 886. 887. 888. 889. 890. 891. 892. 893. 894. 895. 896. 897. 898. 899. 900. 901. 902. 903. 904. 905. 906. 907. 908. 909. 910. 911. 912. 913. 914. 915. 916. 917. 918. 919. 920. 921. 922. 923. 924. 925. 926. 927. 928. 929. 930. 931. 932. 933. 934. 935. 936. 937. 938. 939. 940. 941. 942. 943. 944. 945. 946. 947. 948. 949. 950. 951. 952. 953. 954. 955. 956. 957. 958. 959. 960. 961. 962. 963. 964. 965. 966. 967. 968. 969. 970. 971. 972. 973. 974. 975. 976. 977. 978. 979. 980. 981. 982. 983. 984. 985. 986. 987. 988. 989. 990. 991. 992. 993. 994. 995. 996. 997. 998. 999. 1000.

增強例子如下



資料增強的操作**一定要放在imread 之後，resize 之前，因為有crop 裁剪操作。**

個人習慣，不管是什麼深度學習項目，首先處理好數據，處理對數據，數據絕不能出錯。

現在從映像到tensor 的建置完畢，下一步就是將tensor 送到網路層。

程式碼

編輯於2022-02-19 19:10

tensor 卷積神經網路 (CNN) C++

寫下你的評論...



還沒有評論，發表第一個評論吧

文章被以下專欄收錄



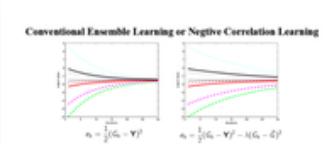
電腦視覺基礎
最簡單的卷積神經網路、ReLU、BN、目標偵測等

推薦閱讀



源图像 A (b) 源图像 B (c) 融合图
图： 基于卷积神经网络的多聚焦图像融合结果

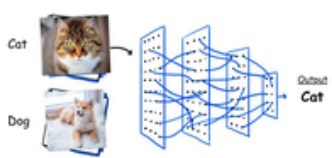
基於卷積神經網路(CNN)的多
聚焦影像融合
AI高級人... 發表於深度人臉識...



基於深度負相關學習的人群計數
方法
SIGAI



【卷積神經網路-演化史】從
LeNet到AlexNet
仙道菜 發表於AutoV...



[CV - Image Classification]
影像分類卷積神經網路模型綜...
Pascal演算法擺渡人

