به نام خدا

مهدی فیروزبخت

تمرین سری ششم

درس بینایی کامپیوتر

الف .

هدف LBP رمزگذاری ویژگی های هندسی یک تصویر با تشخیص لبه ها، گوشه ها، مناطق برجسته یا صاف و خطوط سخت است. به ما امکان می دهد یک نمایش برداری از یک تصویر یا گروهی از تصاویر را تولید کنیم.

با رمزگذاری اشتراکات بین تصاویر یا یک تصویر واحد از یک کلاس ناشناخته معین، ما امکان مقایسه ویژگی های آنها را با تصویر دیگری فراهم می کنیم. از طریق این مقایسه، می‌توانیم سطح شباهت بین نمایش هدفمان و یک تصویر دیده نشده را تعیین کنیم و می‌توانیم احتمال اینکه تصویر ارائه‌شده از همان تنوع یا نوع تصویر هدف باشد را محاسبه کنیم.

اگرچه نسبتاً ساده است و مطمئناً چیز جدیدی نیست، اما این روش یادگیری بدون نظارت قادر است بین تصاویر مشابه به طرز شگفت‌آوری با استفاده از داده‌های کم تفاوت قائل شود. به عنوان یک امتیاز اضافی، این بدون نیاز به آموزش مدل به دست می آید. برای استفاده از این تکنیک، ساختن یک نمایش مقایسه ای کافی است.

LBP را می توان به 4 مرحله کلیدی تقسیم کرد: · ساده سازی · باینری سازی · محاسبه PDF (تابع چگالی احتمال) · مقایسه (از توابع بالا)

ساده سازی

قبل از شروع ایجاد LBP خود، ابتدا باید تصویر خود را ساده کنیم. این مرحله پیش پردازش داده ما است. در اصل، این اولین قدم ما در کاهش ابعاد است، که به الگوریتم ما اجازه می‌دهد تا به جای نگرانی در مورد هر ویژگی بالقوه دیگری، صرفاً بر تفاوت‌های محلی در روشنایی تمرکز کند.

بنابراین، ابتدا تصویر خود را به نمایش یک کانال (معمولاً در مقیاس خاکستری) تبدیل می کنیم.

انجام این کار هم با هر یک از تصاویر هدف ما (تصاویری که انواع یا گروه های ما را نشان می دهند) و هم با تصاویر ورودی ما مهم است. نکته مهم در این مورد این است که ما می‌توانیم با یک تصویر نماینده واحد از یک نوع یا گروه معین کار کنیم و به ما امکان می‌دهد ورودی‌ها را با یک مجموعه داده کوچک، به روشی بدون نظارت طبقه‌بندی کنیم.

دوتایی شدن

سپس، تغییرات درخشندگی محلی نسبی را محاسبه می کنیم. این به ما اجازه می دهد تا یک نمایش محلی، با ابعاد کم و باینری از هر پیکسل بر اساس درخشندگی ایجاد کنیم.

برای هر پیکسل در پنجره ما، k پیکسل اطراف را از "همسایگی" محلی آن می گیریم و هر کدام را به نوبه خود با پیکسل مرکزی مقایسه می کنیم، در جهت عقربه های ساعت یا خلاف جهت عقربه های ساعت حرکت می کنند. جهت و نقطه شروع بی ربط هستند، تا زمانی که ما به یک جهت پایبند باشیم و محاسبه هر پیکسل را به نوبت انجام دهیم. برای هر مقایسه، مقدار باینری 0 یا 1 را به خروجی می‌دهیم که بستگی به این دارد که آیا شدت پیکسل مرکزی (مقدار اسکالر) بیشتر یا کمتر (به ترتیب) از پیکسل مقایسه باشد. این یک مقدار باینری k-bit را تشکیل می دهد که می تواند به عدد پایه 10 تبدیل شود. یک شدت جدید برای آن پیکسل مشخص می شود. این کار را تا زمانی تکرار می‌کنیم که برای هر پیکسل یک شدت پیکسل جدید داشته باشیم، که نشان‌دهنده شدت محلی تجمعی در مقایسه با همسایگان آن است (که در آن مقدار شدت از 0 تا 2^k متغیر است). این باعث می‌شود که نمایش LBP از تصویر اصلی‌مان با ابعاد کاهش‌یافته‌مان باقی بماند.

در عمل، ابتدا پارامترهای LBP خود را تعریف می کنیم. این شامل تنظیم اندازه سلول، شعاع و تعداد نقاط مقایسه ای (k) می شود. اندازه سلول ما به اندازه پیکسل دلخواه MxN اشاره دارد که ممکن است از آن برای تقسیم بیشتر پنجره خود استفاده کنیم. محاسبات فوق ممکن است در هر سلول به طور مستقل به جای کل پنجره استفاده شود. این امکان پردازش موازی سریع‌تر و کارآمدتر تصویر را فراهم می‌کند و همچنین امکان استفاده از نواحی سلول‌های همپوشانی برای دریافت الگوهای محلی را فراهم می‌کند، که اگر LBP را با استفاده از کل پنجره محاسبه کنیم، ممکن است به شدت تقسیم شود. به عنوان استاندارد، اندازه سلول را 16x16 پیکسل قرار می دهیم.

شعاع اندازه همسایگی را که از آن پیکسل های مقایسه ای خود را نمونه برداری می کنیم، برای هر پیکسل مرکزی در تصویرمان در هنگام تولید نمایش LBP تعریف می کند (یعنی تعداد پیکسل های دورتر از پیکسل مرکزی هر پیکسل مقایسه ای است). این دقیقاً منظور ما از "محلی" را مشخص می کند.

در نهایت، مقدار k ما به تعداد نقاطی در همسایگی ما برای نمونه گیری اشاره دارد. به طور معمول، این 8 است، بنابراین یک مقدار 8 بیتی برای هر پیکسل ایجاد می کند. مقادیر شدت پیکسل نهایی ما بین 0 تا 255 (2^8) باشد.
هنگامی که نمایش های LBP k-bit خود را برای هر سلول در پنجره خود تولید کردیم، سپس آماده هستیم تا آنها را ترکیب کنیم تا بردارهای ویژگی خود را تشکیل دهیم.

محاسبه PDF
بنابراین چگونه می‌توانیم نمایش‌های تصویر LBP خود را به چیزی کمی مفیدتر تبدیل کنیم؟
باید آن ها را به بردار ویژگی تبدیل کنیم . در اصل، ما یک هیستوگرام ایجاد می کنیم. ما این نتایج را در قالب یک هیستوگرام رسم می‌کنیم و نمایش‌های LBP را از هر سلول به هم متصل می‌کنیم تا یک نمایش بردار ویژگی در سطح پنجره ایجاد کنیم.
اکنون ما بازنمایی ویژگی‌های محلی و کم‌بعد خود را داریم، می‌توانیم از آنها به عنوان ورودی‌های قابل آموزش برای طبقه‌بندی‌کننده‌های قدرتمندی مانند ماشین‌های بردار پشتیبانی یا می‌توانیم این بردارهای ویژگی را برای کشف کاوش کنیم. ویژگی های هندسی بالقوه مهمی که تصاویر ما را مشخص می کند.

محاسبه مقایسه
در این مرحله میتوانیم از Kullback-Leibler Divergence استفاده کنیم.
این الگوریتم بیزی قدرتمند به ما امکان می دهد دو تابع چگالی یا جرمی احتمال (یا اساساً هر جفت توزیع) را با هم مقایسه کنیم، و یک توزیع احتمال جدید برای توضیح رابطه بین آنها ایجاد کنیم. سپس این به ما می گوید که چقدر احتمال دارد نقاط داده ما از توزیع 'q' از همان توزیع اصلی 'p' آمده باشند.
همانطور که می بینید، LBP به ما اجازه می دهد تا تصاویر خود را با ابعاد کم تولید کنیم، که بر ویژگی های توپوگرافی محلی تأکید دارد. اینها را می توان برای طبقه بندی تصاویر بدون برچسب، با مقایسه ویژگی های بصری کلیدی آنها، برای تعیین احتمال نمونه برداری هر تصویر از یک جامعه استفاده کرد.
با توجه به توضیحات ارائه شده ، این الگوریتم برای بردار ها از یک هیستوگرام استفاده میکند که باعث میشود بردار ما همیشه اندازه یکسانی داشته باشد و ارتباطی با مکان پنجره ندارد. همچنین از لحاظ سرعت نیز به سادگی میتوان از پردازش موازی استفاده کرد و سرعت را بالاتر برد زیرا پنجره ها مستقل از یکدیگر هستند. همچنین مدل طبقه بندی کننده ارائه شده نیز از سرعت بالایی برخوردار است که به سادگی طبقه بندی را انجام میدهد.
ب.
با توجه به اینکه در روش LBP از هیستوگرام ویژگی های پنجره استفاده شده است ، پس قابل درک است که این روش از اساس هیستوگرام و مقایسه هیستوگرام استفاده میکند پس احتمالا این احتمال وجود دارد که بتوانیم به جای این روش برای استخراج ویژگی از سایر روش های هیستوگرامی مانند SIFT یا HOG یا سایر روش های دیگر استفاده کرد.
در اینجا منظور از روش های هیستوگرامی این است که در این روش هیستوگرام ویژگی های استخراجی محاسبه شده و از آن استفاده میکند. در روش SIFT نیز از هیستوگرام ویژگی های استخراجی یک بردار ۱۲۸ بعدی میسازد یا در روش SURF یک بردار ۶۴ بعدی میسازد که از نظر بنده این بردار ویژگی ساخته شده قابل جایگزینی به جای روش LBP میباشد.

ج.

ما برای این سوال ابتدا فایل های مربوطه را درون گوگل درایو اپلود میکنیم تا بتوانیم از آن درون کولب استفاده کنیم. سپس فایل اصلی را میخوانیم.

ابتدا اتصالات به درایو را برقرار میکنیم :

```
from google.colab import drive
import sys
sys.path.insert(0,'/content/drive/My Drive/ColabNotebooks')
drive.mount('/content/drive')
%cd /content/drive/My\ Drive/Colab\ Notebooks
```

سپس به فولدر مربوطه رفته و npy بارگذاری شده را خوانده و از آن استفاده میکنیم . اما برای استفاده از آن نیاز است درون فایل DatasetGenerator تغییراتی را ایجاد نماییم پس وارد این فایل میشویم.

در این فایل نیاز داریم تا ۲ تابع را بازنویسی کنیم. ابتدا تابع extract_lbp :

این تابع باید بتواند lbp را برای هر پنجره کاندید به دست بیاورد سپس آن را نرمال نماید و هیستوگرام آن را به دست آورد:

```
def extract_lbp(self, image, radius, n_points, method='uniform'):
    lbp = local_binary_pattern(image, n_points, radius, method=method).astype('float32')
    lbp -= lbp.min()
    lbp /= lbp.max()
    hist, _ = np.histogram(lbp.ravel(), bins=n_points, range=(0, 1))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
```

در مرحله بعد تابع extract_features_roi قرار دارد که درون این تابع باید مشخص کنیم که چه پنجره ای و چه سایزی و در کل اطلاعات مربوط به LBP را درون آن بنویسیم .درون این تابع باید تابع قبلی را فراخوانی کرده تا اعمال شود.

```
def extract_features_roi(self, image, candidate):
    radius = 3
    n_points = 8 * radius
    x1, y1, x2, y2 = candidate
    detected_image = image[y1:y2, x1:x2]
    features = self.extract_lbp(detected_image, radius, n_points)
    return features
```

این تابع نیز در بدنه تابع extract_features_targets_pairs فراخوانی شده است. این تابع درون کد فایل اصلی برای دریافت ویژگی ها نوشته شده است. به فایل اصلی باز میگردیم. درون این فایل ابتدا ویژگی ها ، لیبل ها و target_deltas را به دست می آوریم :

```
"""Get the dataset and extract features"""
dl = DataLoader()
dg = DatasetGenerator(dl)
"""This method is relatively slow. You can download pre-computed dataset here
https://drive.google.com/file/d/1SGJWgt3z6S44uo6GI9sFuX_WGvOoZ-8z/view?usp=sharing
"""
dg.prepareDataset()
f1, t1, t2 = dg.extract_features_targets_pairs()
```

```
There are 204 images in dataset
193 images for train set and 11 for test set
Dataset classes {1: 'C', 2: 'dead', 3: 'T', 4: 'weapon', 0: 'background'}
Loading Dataset ...
Done
100%|██████████| 193/193 [03:11<00:00,  1.01it/s]
```

سپس با استفاده از کد های قرار داده شده آن ها را به دو گروه تست و آموزش تقسیم میکنیم . در مرحله بعد نیاز داریم این ویژگی ها را به روی یک طبقه بندی کننده آموزش دهیم. برای این کار از دسته از آموزش دهنده ها استفاده میکنیم تا بهترین آن را انتخاب کنیم :

```
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="rbf", C=0.025, probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    GaussianNB(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis()]

# Logging for Visual Comparison
log_cols=["Classifier", "Accuracy", "Log Loss"]
log = pd.DataFrame(columns=log_cols)

for clf in classifiers:
    clf.fit(X_train, y_train)
    name = clf.__class__.__name__

    print("="*30)
    print(name)

    print('****Results****')
    train_predictions = clf.predict(X_test)
    acc = accuracy_score(y_test, train_predictions)
    print("Accuracy: {:.4%}".format(acc))

    train_predictions = clf.predict_proba(X_test)
    ll = log_loss(y_test, train_predictions)
    print("Log Loss: {}".format(ll))

    log_entry = pd.DataFrame([[name, acc*100, ll]], columns=log_cols)
    log = log.append(log_entry)
```

نتیجه آن به صورت زیر است :

```
==============================
KNeighborsClassifier
****Results****
Accuracy: 93.6852%
Log Loss: 1.0463670863319512
==============================
SVC
****Results****
Accuracy: 54.9087%
Log Loss: 0.9119897237255008
==============================
DecisionTreeClassifier
****Results****
Accuracy: 94.4746%
Log Loss: 1.9084079705130765
==============================
RandomForestClassifier
****Results****
Accuracy: 97.7800%
Log Loss: 0.11142149640943125
==============================
AdaBoostClassifier
****Results****
Accuracy: 58.7074%
Log Loss: 1.3217428883620685
==============================
GradientBoostingClassifier
****Results****
Accuracy: 95.8066%
Log Loss: 0.1720510162624159
==============================
GaussianNB
****Results****
Accuracy: 56.3888%
Log Loss: 3.496387091467124
==============================
LinearDiscriminantAnalysis
****Results****
Accuracy: 68.8209%
Log Loss: 0.902965918337204
==============================
QuadraticDiscriminantAnalysis
****Results****
Accuracy: 84.6571%
Log Loss: 1.1045515829346237
==============================
```

که با این نتیجه جنگل تصادفی بهترین حالت را ایجاد میکند. در مرحله بعد بنده لازم دیدم تا به روی این دسته بندی کننده عملیات Hyper Parameters Tuning انجام دهم تا نتیجه بهتری ایجاد شود .

```
rf = RandomForestClassifier()

rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 10, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train, y_train)
```

که نتیجه آن به صورت زیر میباشد :

```
rf_random.best_params_
```

```
{'n_estimators': 2000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 60,
 'bootstrap': False}
```

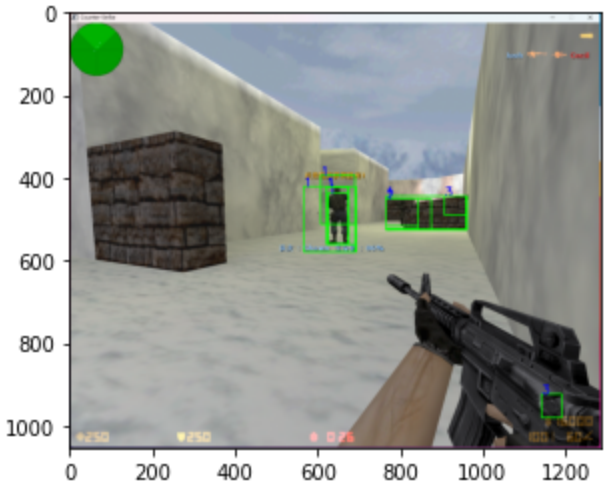
و مقایسه نتیجه با حالت کلی :

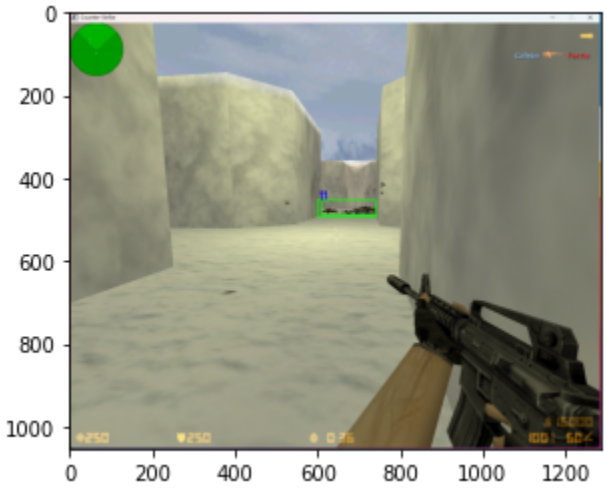
```
Accuracy: 97.8293%
Accuracy: 98.1253%
Improvement of 0.30%.
```

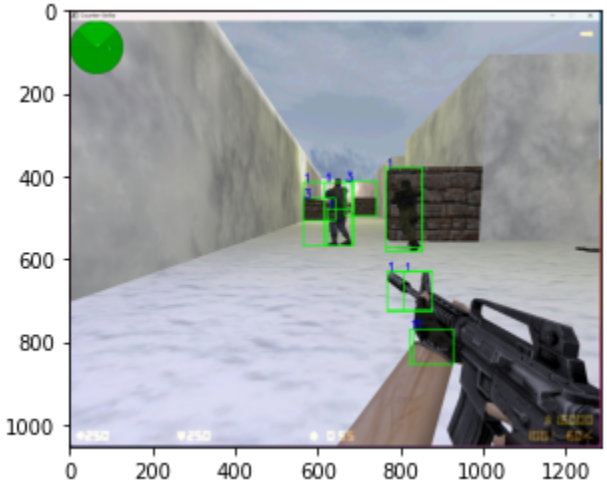
پس از حالت بهتر استفاده میکنیم. در مرحله بعد نیاز داریم تا نتیجه را به روی داده های تست به دست آورده و به کمک این ویژگی ها و دسته بندی ها نتایج را به روی تصویر اعمال کنیم. پس ابتدا برای هر تصویر ، پنجره های کاندید را پیدا میکنیم. سپس با استفاده از تابع نوشته شده در فایل دوم ویژگی LBP هر پنجره محاسبه شده و به مدل آموزش دیده وارد میشود. نتیجه هر پنجره مشخص شده و در متغیر classes قرار داده میشود. سپس بررسی میکنیم اگر طول و عرض هر کاندید از حد مشخصی بیشتر بود آن را رسم میکنیم. شماره کلاس منصوب نیز درون شکل مشخص شده است . کد آن به صورت زیر میباشد :

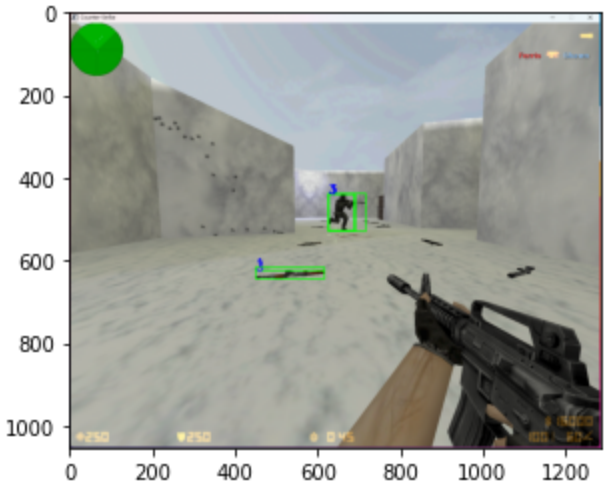
```
import cv2
"""
Object detection on test images
"""
for image in dl.getAllTestData():
    candidates = extract_candidates(image)
    features = []

    # Extract features of each ROI
    for candidate in candidates:
        features.append(dg.extract_features_roi(skimage.color.rgb2gray(image), candidate))

    features = np.array(features)
    classes = clf.predict(features)

    # Show Bounding Boxes with their classes.
    for i in range(len(candidates)):
        candidate = candidates[i]
        if classes[i] != '0':
            distance = candidate[2] - candidate[0]
            width = candidate[3] - candidate[1]
            if distance < 225 and width < 225:
                cv2.rectangle(image, (candidate[0], candidate[1]), (candidate[2], candidate[3]), (0, 255, 0), 2)
                cv2.putText(image, str(classes[i]), (candidate[0], candidate[1]), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_AA)

    plt.imshow(image)
    plt.show()
```

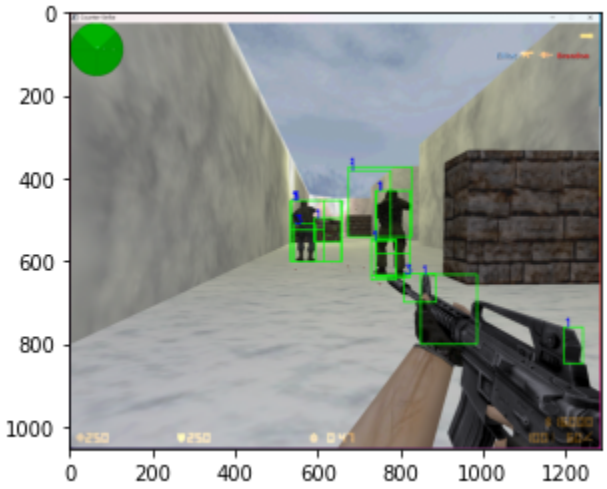
نتیجه برای هر تصویر نیز به صورت زیر میباشد :

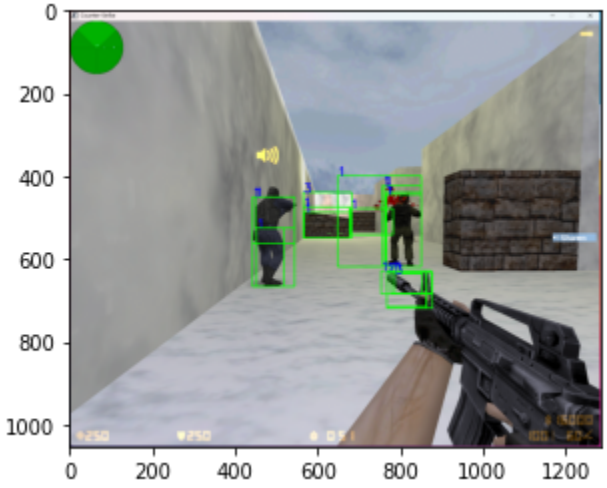

0
200
400
600
800
1000
0
200
400
600
800
1000
1200

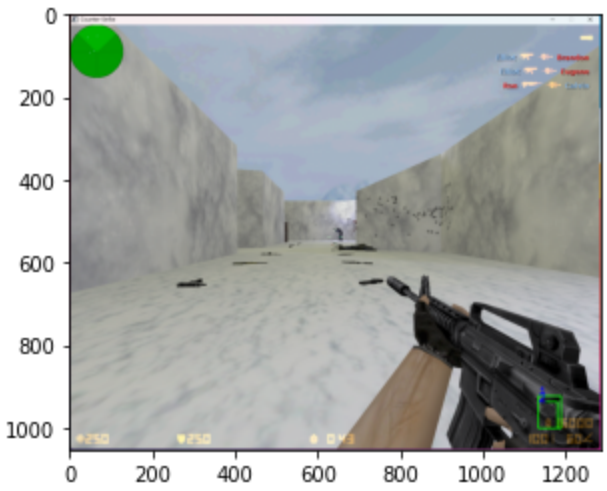

0
200
400
600
800
1000
0
200
400
600
800
1000
1200

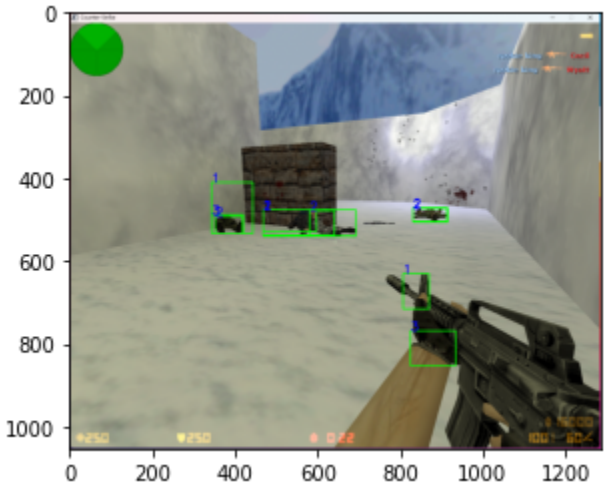

0
200
400
600
800
1000
0
200
400
600
800
1000
1200

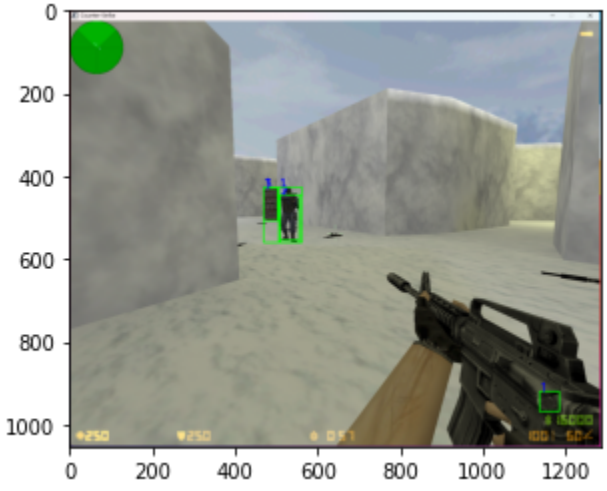

0
200
400
600
800
1000
0
200
400
600
800
1000
1200

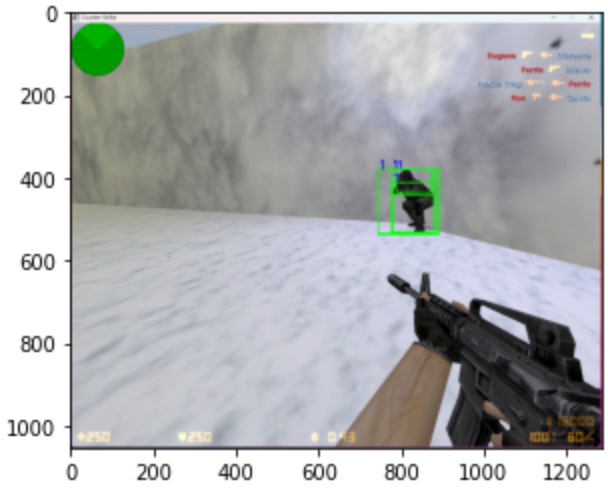

0
200
400
600
800
1000
0
200
400
600
800
1000
1200

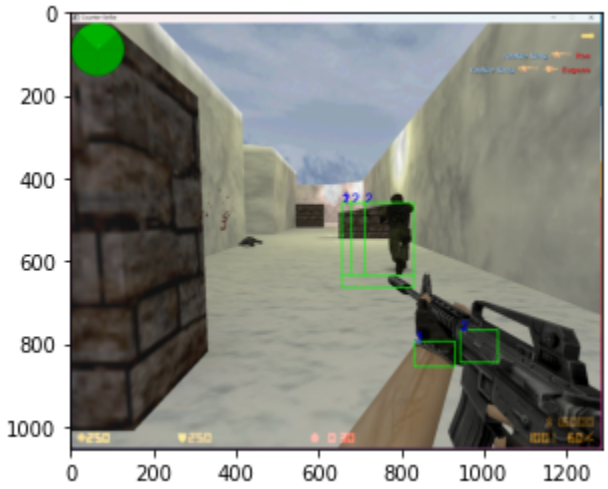

0
200
400
600
800
1000
0
200
400
600
800
1000
1200


0
200
400
600
800
1000
0
200
400
600
800
1000
1200


0
200
400
600
800
1000
0
200
400
600
800
1000
1200


0
200
400
600
800
1000
0
200
400
600
800
1000
1200



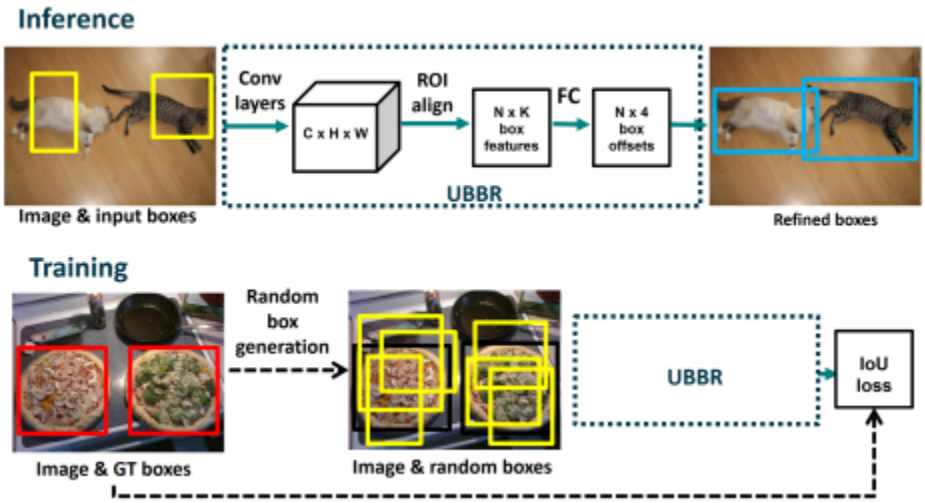


خروجی چندین مشکل عمده دارد. یک سرعت استخراج پنجره ها و استخراج نتیجه اندکی پایین است که باعث میشود نتوانیم از آن به صورت Real-Time استفاده کرد. همچنین مشکل اصلی که به نظر بنده خروجی ها دارد این است که برخی پنجره ها به شدت درون یکدیگر قرار دارند و شاید بتوان این پنجره ها را با یکدیگر ادغام کرد و پنجره بزرگتری ایجاد کرد که در ادامه آن به مشکلی دیگری بر می خوریم که پنجره های ایجاد شده بسیار کوچک هستند و در برخی موارد مانند نشان دادن اسلحه فقط یک نقطه را نمایش میدهد که تشکیل آن برای کاربر سخت و دشوار خواهد بود.

د.

به منظور بهبود عملکرد بومی سازی، طراحان R-CNN یک مرحله رگرسیون جعبه مرزی را برای یادگیری اصلاحات در مکان و اندازه جعبه مرزی پیش بینی شده در نظر می گیرند. رگرسیون جعبه مرزی یک تکنیک محبوب برای اصلاح یا پیش‌بینی جعبه‌های محلی‌سازی در رویکردهای اخیر تشخیص شی است. به طور معمول، رگرسیورهای جعبه مرزی برای پسرفت از هر دو پیشنهاد منطقه یا جعبه های لنگر ثابت به جعبه های مرزی نزدیک کلاس های شی هدف از پیش تعریف شده، آموزش داده می شوند.به طور خلاصه، جعبه های مرزبندی یک روش حاشیه نویسی تصویر برای آموزش مدل های یادگیری ماشین مبتنی بر هوش مصنوعی هستند. برای تشخیص اشیا و شناسایی هدف در طیف وسیعی از کاربردها، از جمله روبات‌ها، هواپیماهای بدون سرنشین، وسایل نقلیه خودران، دوربین‌های نظارتی و سایر دستگاه‌های بینایی ماشین استفاده می‌شود.

میزان اهمیت آن دقیقا در جایی مانند نتیجه خروجی الگوریتم LBP نمایان میکند. همانطوری که در این الگوریتم دیدیم ، نتیجه خروجی پنجره های بسیار کوچکی بودند که در مواقعی نمی توانستند نمایندگی شکل تشخیص داده شده باشند. در این حالت این ماژول رگرسیون جعبه مرزی یا BBox regression میتواند کمک شایانی در این امر داشته باشد.

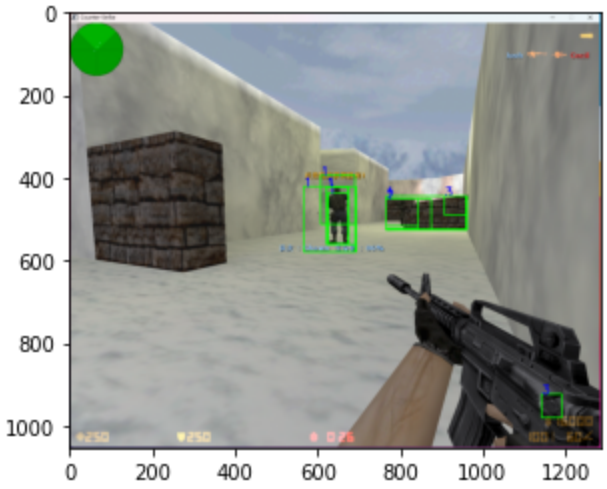


برای مثال همچین حالتی بیان کننده خاصیت و کارکرد این ماژول را دارد.

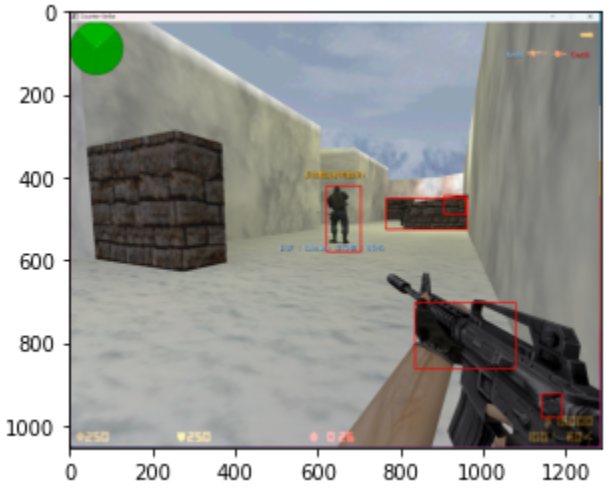
همانند این تصویر ، باید ماژولی را ایجاد کنیم که عکس و باکس ها را دریافت کرده ، مقدار باکس ها را بتواند تغییر دهد سپس برای آموزش رگرسیور از IoU loss استفاده کند تا به بهترین نتیجه برسد.

برای پیاده سازی این حالت از IoU loss استفاده کرده ایم. ابتدا همانند روش قبلی کاندید ها را پیدا کرده ، ویژگی ها را استخراج کرده ، وارد کلاس کرده و مستطیل ها را میابیم. سپس با مستطیل هایی که جز زمینه نیستند را جدا میکنیم. سپس این مستطیل های باقی مانده را با مستطیل های کاندید مقایسه میکنیم و برای مقایسه از IoU استفاده میکنیم. هر چقدر این مستطیل ها شباهت بیشتری داشته باشند به معنی این است که از یک کلاس خواهند آمد. پس یک مقدار آستانه قرار میدهیم و مستطیل هایی که بیشتر از این آستانه هستند را با یکدیگر ترکیب میکنیم. همچنین یک مقایسه ای درون این لیست جدید انجام میدهیم و همین کار را برای آن تکرار میکنیم. برای مثال نتیجه به دست آمده برای یکی از تصاویر به صورت زیر میباشد :

تصویر بدون الگوریتم :



تصویر با الگوریتم :



قابل مشاهده است که نتیجه به دست آمده بسیار بهتر است.

ه.

روش سرکوب حداکثری روشی است که در آن در یک همسایگی به دنبال این هستیم که فقط مقدار حداکثر ویژگی را انتخاب کنیم. برای مثال در روش Canny از این الگوریتم برای اینکه مقدار لبه را تک مقداری یا تک نقطه ای کنیم استفاده شده است. از این روش در مرحله اول که مرحله پیشنهاد نواحی است میتوانیم استفاده کنیم. در ادامه چگونگی استفاده از این الگوریتم بیان شده است.

خط لوله تشخیص شی معمولی دارای یک جزء برای ایجاد پیشنهادات برای طبقه بندی است. پیشنهادها چیزی جز مناطق کاندید برای موضوع مورد علاقه نیستند. اکثر رویکردها از یک پنجره کشویی بر روی نقشه ویژگی استفاده می کنند و امتیازهای پیش زمینه/پس زمینه را بسته به ویژگی های محاسبه شده در آن پنجره اختصاص می دهند. پنجره های محله تا حدودی امتیازات مشابهی دارند و جزء مناطق کاندید محسوب می شوند. این منجر به صدها پیشنهاد می شود. از آنجایی که روش تولید پروپوزال باید یادآوری بالایی داشته باشد، در این مرحله محدودیت‌های سست را حفظ می‌کنیم. با این حال، پردازش این بسیاری از پیشنهادات از طریق شبکه طبقه‌بندی دشوار است. این منجر به تکنیکی می شود که پیشنهادات را بر اساس برخی معیارها به نام Non-Maximum Suppression فیلتر می کند:

ورودی: فهرستی از جعبه‌های پیشنهاد B، امتیازات مربوطه S و آستانه همپوشانی N.

خروجی: فهرستی از پیشنهادات فیلتر شده D.

الگوریتم:

پیشنهادی را با بالاترین امتیاز انتخاب کنید، آن را از B حذف کنید و به لیست پیشنهادی نهایی D اضافه کنید (در ابتدا D خالی است).

اکنون این پیشنهاد را با همه پیشنهادات مقایسه کنید - IOU (تقاطع روی اتحاد) این پیشنهاد را با هر پیشنهاد دیگری محاسبه کنید. اگر IOU بزرگتر از آستانه N است، آن پیشنهاد را از B حذف کنید.

این روند تا زمانی تکرار می شود که دیگر پیشنهادی در B باقی نماند.

حال اگر الگوریتم بالا را مشاهده کنید، کل فرآیند فیلتر کردن به مقدار آستانه منفرد بستگی دارد. بنابراین انتخاب مقدار آستانه کلید عملکرد مدل است. با این حال، تنظیم این آستانه دشوار است.

در این تصویر چگونگی استفاده از این الگوریتم و نتیجه را میتوانیم مشاهده کنیم.