

Rank	Confidence Score	TP or FP	Correct	Precision	Recall
1	0.95	TP	True		
2	0.82	TP	False		
3	0.58	FP	True		
4	0.50	FP	False		
5	0.44	TP	True		
6	0.39	FP	False		
7	0.24	TP	True		
8	0.10	FP	False		

مقادیر ستون های recall و precision را از روابط زیر محاسبه میکنیم:

$$recall = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

مقدار threshold را در مراحل مختلف تغییر میدهیم، در هر مرحله مقدار threshold را بین مقادیر دو ردیف قرار میدهیم.

مرحله ۱ : 0.95 و 0.82 بین threshold=0.90

$$recall = \frac{TP}{TP + FN} = \frac{1}{1 + 3} = 0.25$$

$$precision = \frac{TP}{TP + FP} = \frac{1}{1 + 0} = 1$$

مرحله ۲ : 0.58 و 0.50 بین threshold=0.70

$$recall = \frac{TP}{TP + FN} = \frac{2}{2 + 2} = 0.5$$

$$precision = \frac{TP}{TP + FP} = \frac{2}{2 + 0} = 1$$

مرحله ۳ : 0.50 و 0.44 بین threshold=0.55

$$recall = \frac{TP}{TP + FN} = \frac{2}{2 + 2} = 0.5$$

$$precision = \frac{TP}{TP + FP} = \frac{2}{2 + 1} = 0.6667$$

مرحله ۴ : 0.44 و 0.39 بین threshold=0.46

$$recall = \frac{TP}{TP + FN} = \frac{2}{2+2} = 0.5$$

$$precision = \frac{TP}{TP + FP} = \frac{2}{2+2} = 0.5$$

مرحله 5 : threshold=0.42 بین 0.39 و 0.44

$$recall = \frac{TP}{TP + FN} = \frac{3}{3+1} = 0.75$$

$$precision = \frac{TP}{TP + FP} = \frac{3}{3+2} = 0.6$$

مرحله 6 : threshold=0.31 بین 0.24 و 0.39

$$recall = \frac{TP}{TP + FN} = \frac{3}{3+1} = 0.75$$

$$precision = \frac{TP}{TP + FP} = \frac{3}{3+3} = 0.5$$

مرحله 7 : threshold=0.18 بین 0.10 و 0.24

$$recall = \frac{TP}{TP + FN} = \frac{4}{4+0} = 1$$

$$precision = \frac{TP}{TP + FP} = \frac{4}{4+3} = 0.5714$$

مرحله 8 : threshold=0.05 بین 0.0 و 0.10

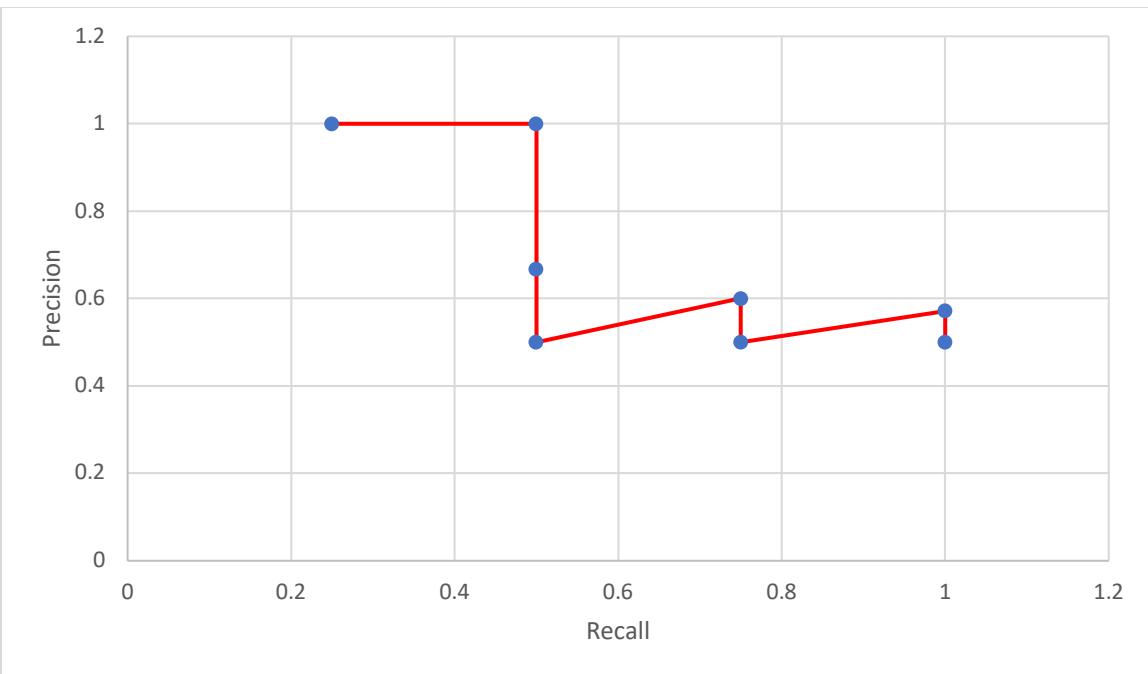
$$recall = \frac{TP}{TP + FN} = \frac{4}{4+0} = 1$$

$$precision = \frac{TP}{TP + FP} = \frac{4}{4+4} = 0.5$$

در انتهای جدول زیر به دست می آید:

Rank	Confidence Score	TP or FP	Correct	Precision	Recall
1	0.95	TP	True	1	0.25
2	0.82	TP	False	1	0.5
3	0.58	FP	True	0.6667	0.5
4	0.50	FP	False	0.5	0.5
5	0.44	TP	True	0.6	0.75
6	0.39	FP	False	0.5	0.75
7	0.24	TP	True	0.5714	1
8	0.10	FP	False	0.5	1

نمودار precision بر حسب recall را به صورت زیر رسم میشود:



بر اساس راهنمای ارائه شده، AP را با استفاده از روابط زیر و بر اساس point_interpolation11 محاسبه میکنیم

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} \rho_{interp}(r)$$

$$\rho_{interp} = \max_{\tilde{r}: \tilde{r} \geq r} \rho(\tilde{r})$$

مقادیر precision بر حسب recall به صورت زیر به دست می آید:

recall	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
precision	1	1	1	1	1	1	0.6	0.6	0.5714	0.5714	0.5714

مقدار AP برابر است با:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} \rho_{interp}(r)$$

$$\begin{aligned}
 AP &= \frac{1}{11} (1 + 1 + 1 + 1 + 1 + 1 + 0.6 + 0.6 + 0.6 + 0.5714 + 0.5714 + 0.5714) \\
 &= \frac{8.9142}{11} = 0.8103
 \end{aligned}$$

یک راه حل این است که به جای استفاده از الگوریتم kalman معمولی از روش پیشنهاد شده در مقاله زیر استفاده کرد
<https://www.mdpi.com/1424-8220/22/23/9106/pdf>

در این روش به جای Kalman filter از unscented Kalman filter استفاده شده و الگوریتم deep sort ارتقا پیدا کرده است.

بلوک دیاگرام این روش به صورت زیر است:

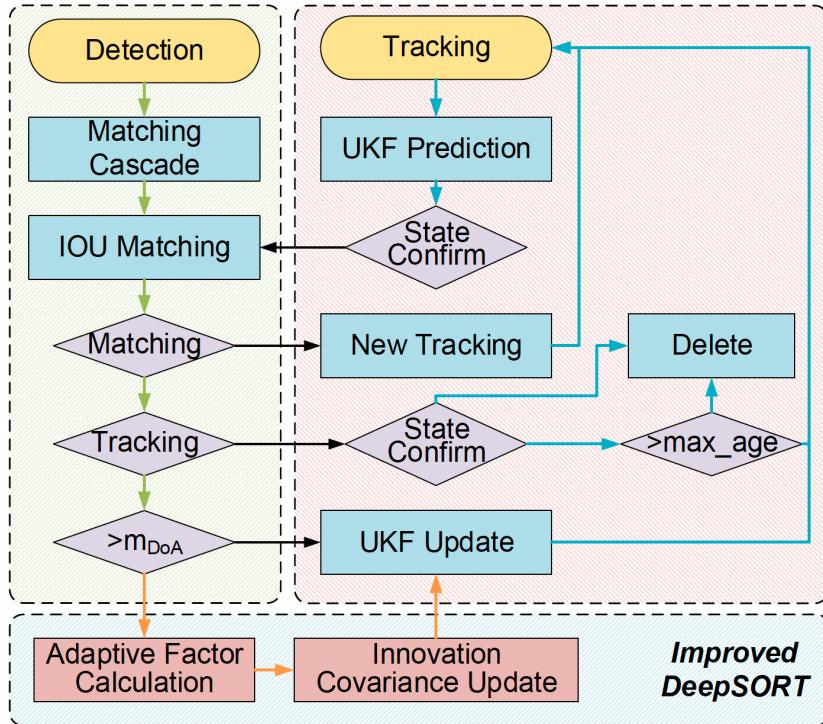


Figure 2. Improved DeepSORT algorithm framework.

پیشنهاد دیگر این است که به جای استفاده از Yolo v4 از نسخه های جدید تر مثل Yolo v7 که بسیار قوی تر هستند استفاده کرد

Code reference:

<https://github.com/abewley/sort/blob/master/sort.py>

نحوه عملکرد الگوریتم Kalman میتوان با در نظر گرفتن مدل ریاضی حرکت، مکان بعدی شیء مورد نظرمان را تخمین بزنیم، اما این تخمین دقیق نیست و خطأ دارد. به این مرحله predict گفته میشود. همچنانیم میتوانیم با الگوریتم های تشخیص اشیا نیز مکان آن شیء را به صورت تقریبی پیدا کنیم ولی این موقعیت پیدا شده نیز دقیق نیست و خطأ دارد. به این مرحله measurement گفته میشود. برای رسیدن به بهترین نتیجه، الگوریتم کالمن این دو نتیجه را با هم ترکیب میکند و براساس مقدار خطأ یا کواریانس آن ها، بین آن دو نقطه پیشنهاد شده نقطه ای را به عنوان موقعیت شیء در نظر میگیرد. این مرحله update نام دارد.

```
class Track:
    count = 0
    def __init__(self, box):
        """
        Initializes a track given its first bounding box. The class also initializes an instance of `KalmanFilter`
        using the same bounding box.

        Parameters
        -----
        box: tuple of number
            Tuple containing `x_min`, `y_min`, `width`, and `height` of the first detection.
        """
        self.unseen = 0
        self.history = []
        #####
        self.filter = KalmanFilter(dim_x, dim_z)
        self.filter.F = np.array(F)
        self.filter.H = np.array(H)
        self.filter.P = np.array(P)
        self.filter.Q = np.array(Q)
        self.filter.R = np.array(R)
        center_x, center_y, area, ratio = self.xywh_to_xxar(box[0], box[1], box[2], box[3])
        self.filter.x[:4] = np.array([center_x, center_y, area, ratio]).reshape((4,1))
        self.id = Track.count
        Track.count += 1
        self.hits = 0
        self.hit_streak = 0
        self.age = 0
        #####
```

```
def predict(self):
    """
    Predicts the next bounding box using the `KalmanFilter` instance.

    Returns
    -----
    x_min: number
        Horizontal coordinates of the top-left corner of the box.
    y_min: number
        Vertical coordinates of the top-left corner of the box.
    width: number
        Width of the bounding box.
    height: number
        Height of the bounding box.
    """
#####
if((self.filter.x[6]+self.filter.x[2])<=0):
    self.filter.x[6] *= 0.0
self.filter.predict()
self.age += 1
if(self.unseen>0):
    self.hit_streak = 0
self.unseen += 1
center_x, center_y, area, ratio = self.filter.x[:4]
self.history.append(np.array(self.xxar_to_xywh(center_x, center_y, area, ratio)).reshape((1,4)))
XXAR = self.history[-1]
return XXAR
#####
```

```
def update(self, box):
    """
    Updates the `KalmanFilter` instance with the new measurement, and resets `unseen`.

    Parameters
    -----
    box: tuple of number
        Tuple containing `x_min`, `y_min`, `width`, and `height` of the first detection.
    """
#####
# Your code goes here.
self.history = []
self.hits += 1
self.hit_streak += 1
center_x, center_y, area, ratio = self.xywh_to_xxar(box[0], box[1], box[2], box[3])
self.filter.update(np.array([center_x, center_y, area, ratio]).reshape((4,1)))
#####
self.unseen = 0
```

```

@staticmethod
def xywh_to_xxar(x_min, y_min, width, height):
    """
    Utility function that transforms the bounding box formatting.

    Parameters
    -----
    x_min: number
        Horizontal coordinates of the top-left corner of the box.
    y_min: number
        Vertical coordinates of the top-left corner of the box.
    width: number
        Width of the bounding box.
    height: number
        Height of the bounding box.

    Returns
    -----
    center_x: number
        Horizontal coordinates of the center of the box.
    center_y: number
        Vertical coordinates of the center of the box.
    area: number
        Horizontal coordinates of the center of the box.
    ratio: number
        Horizontal coordinates of the center of the box.
    """
    #####
    # Your code goes here.
    center_x = x_min + width/2.
    center_y = y_min + height/2.
    area = width * height
    ratio = width / float(height)
    #####
    return center_x, center_y, area, ratio

```

```

@staticmethod
def xxar_to_xywh(center_x, center_y, area, ratio):
    """
    Utility function that transforms the bounding box formatting.

    Parameters
    -----
    center_x: number
        Horizontal coordinates of the center of the box.
    center_y: number
        Vertical coordinates of the center of the box.
    area: number
        Horizontal coordinates of the center of the box.
    ratio: number
        Horizontal coordinates of the center of the box.

    Returns
    -----
    x_min: number
        Horizontal coordinates of the top-left corner of the box.
    y_min: number
        Vertical coordinates of the top-left corner of the box.
    width: number
        Width of the bounding box.
    height: number
        Height of the bounding box.
    """
    #####
    # Your code goes here.
    width = np.sqrt(area * ratio)
    height = area / width

    x_min = center_x - width/2
    y_min = center_y - height/2

    #####
    return x_min, y_min, width, height

```

```

def get_state(self):
    """
    Returns the current bounding box estimate.
    """
    return self.xxar_to_xywh(self.filter.x[:4][0], self.filter.x[:4][1],
                           self.filter.x[:4][2], self.filter.x[:4][3])

```

طريقه عملکرد کد:

ابتدا برای اولین فریم ورودی با استفاده از مدل شبکه عصبی استفاده شده یک detection صورت میگیرد. این detection به الگوریتم sort داده میشود، الگوریتم باید بررسی کند آیا این detection با آبجکت هایی که از قبل در حال track کردن آن ها بوده همخوانی دارد یا نه.

به دلیل اینکه در فریم اول چیزی از قبل در حال track شدن نبوده، الگوریتم sort این detection ها را به لیستی به نام Tracks اضافه میکند و از این به بعد آن ها را Track میکند.

سینک کردن بین detection ها و track ها در متده به نام update صورت میگیرد.

از فریم دوم به بعد که تعدادی track داریم، در هر مرحله بعد از اینکه detection شده این track ها با

ها به صورت زیر مقایسه میشوند تا همخوانی آنها بررسی شود.

الگوریتم tracker نقطه بعدی را برای هر bbox تخمین میزند و این تخمین باید با مکانی که detection پیشنهاد میدهد مقایسه شود تا همخوانی آنها بررسی شود.

ابتدا درتابع update به ازای همه track ها و detection ها دو به دو iou محاسبه میشود.

bbox هایی که مقدار iou آنها از threshold iou بزرگتر هستند به عنوان bounding box هایی در نظر گرفته میشوند که درست track شده اند.

آن دسته از bbox هایی که مقدار iou آنها از threshold iou کوچکتر است به دو دسته تقسیم میشوند

- ۱ - آن دسته از آبجکت هایی که درست track نشده اند
- ۲ - آن دسته از آبجکت هایی که درست track نشده اند

آن دسته از آبجکت هایی که detect شده اند ولی برای آنها iou مناسب پیدا نشده است، به عنوان آبجکت هایی جدیدی درنظر گرفته میشوند که جدیدا وارد فریم شده اند و برای آنها یک tracker از صفر درنظر گرفته میشود.

```
def linear_assignment(self, cost_matrix):  
    try:  
        import lap  
        _, x, y = lap.lapjv(cost_matrix, extend_cost=True)  
        return np.array([[y[i], i] for i in x if i >= 0]) #  
    except ImportError:  
        from scipy.optimize import linear_sum_assignment  
        x, y = linear_sum_assignment(cost_matrix)  
        return np.array(list(zip(x, y)))
```

```

def associate_detections_to_trackers(self, detections, tracks, iou_threshold = 0.3):
    """
    Assigns detections to tracked object (both represented as bounding boxes)
    Returns 3 lists of matches, unmatched_detections and unmatched_trackers
    """
    if(len(tracks)==0):
        return np.empty((0,2),dtype=int), np.arange(len(detections)), np.empty((0,5),dtype=int)

    iou_matrix = self.iou(detections, tracks)

    if min(iou_matrix.shape) > 0:
        a = (iou_matrix > iou_threshold).astype(np.int32)
        if a.sum(1).max() == 1 and a.sum(0).max() == 1:
            matched_indices = np.stack(np.where(a), axis=1)
        else:
            matched_indices = self.linear_assignment(-1*iou_matrix)
    else:
        matched_indices = np.empty(shape=(0,2))

    unmatched_detections = []
    for d, det in enumerate(detections):
        if(d not in matched_indices[:,0]):
            unmatched_detections.append(d)
    unmatched_trackers = []
    for t, trk in enumerate(tracks):
        if(t not in matched_indices[:,1]):
            unmatched_trackers.append(t)

    #filter out matched with low IOU
    matches = []
    for m in matched_indices:
        if(iou_matrix[m[0], m[1]]<iou_threshold):
            unmatched_detections.append(m[0])
            unmatched_trackers.append(m[1])
        else:
            matches.append(m.reshape(1,2))
    if(len(matches)==0):
        matches = np.empty((0,2),dtype=int)
    else:
        matches = np.concatenate(matches,axis=0)

    return matches, np.array(unmatched_detections), np.array(unmatched_trackers)

def update(self, detections, scores):
    """
    Updates the tracks based on the passed detections and their scores.

    Parameters
    -----
    detections: list of tuple
        Current frame detections, each containing `x_min`, `y_min`, `width`, and `height`.
    scores: list of float
        Corresponding confidence scores of the detected objects.

    Returns
    -----
    track_id: list of int
        Track ID associated with the detection. `-1` indicates that the detection was rejected.
    """
    #####
    # Your code goes here.
    dets = [[detects[0], detects[1], detects[2], detects[3], score]
            for detects, score in zip(detections, scores)]
    self.frame_count += 1
    # get predicted locations from existing trackers.
    trks = np.zeros((len(self.tracks), 5))
    to_del = []
    ret = []
    for t, trk in enumerate(trks):
        pos = self.tracks[t].predict()[0]
        trk[:] = [pos[0], pos[1], pos[2], pos[3], 0]
        if np.any(np.isnan(pos)):
            to_del.append(t)
    trks = np.ma.compress_rows(np.ma.masked_invalid(trks))
    for t in reversed(to_del):
        self.tracks.pop(t)
    matched, unmatched_dets, unmatched_trks = self.associate_detections_to_trackers(dets, trks, self.iou_threshold)

```

```

# update matched trackers with assigned detections
for m in matched:
    self.tracks[m[1]].update(dets[m[0]][:-1])

# create and initialise new trackers for unmatched detections
for i in unmatched_dets:
    trk = Track(dets[i][:-1])
    self.tracks.append(trk)
i = len(self.tracks)
for trk in reversed(self.tracks):
    d = trk.get_state()[0]
    if (trk.unseen < 1) and (trk.hit_streak >= self.keep_alive or self.frame_count <= self.keep_alive):
        ret.append(np.concatenate((d,[trk.id+1])).reshape(1,-1))
    i -= 1
    # remove dead tracklet
    if(trk.unseen > self.max_age):
        self.tracks.pop(i)
if(len(ret)>0):
    return np.concatenate(ret)
return np.empty((0, 5))
#####
#####

def iou(self, box_a, box_b):
#####
iou_matrix = np.zeros((len(box_a), len(box_b)), dtype='float32')
for i in range(len(box_a)):
    for j in range(len(box_b)):
        xmin_a, ymin_a, w, h, _ = box_a[i]
        xmin_b, ymin_b, wb, hb, _ = box_b[j]

        xmax_a = xmin_a + w
        ymax_a = ymin_a + h
        xmax_b = xmin_b + wb
        ymax_b = ymin_b + hb

        #####
        xA = max(xmin_a, xmin_b)
        yA = max(ymin_a, ymin_b)
        xB = min(xmax_a, xmax_b)
        yB = min(ymax_a, ymax_b)
        # compute the area of intersection rectangle
        interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
        # compute the area of both the prediction and ground-truth
        # rectangles
        boxAArea = (xmax_a - xmin_a + 1) * (ymax_a - ymin_a + 1)
        boxBArea = (xmax_b - xmin_b + 1) * (ymax_b - ymin_b + 1)
        # compute the intersection over union by taking the intersection
        # area and dividing it by the sum of prediction + ground-truth
        # areas - the intersection area
        iou = interArea / float(boxAArea + boxBArea - interArea)
        iou_matrix[i][j] = iou
#####

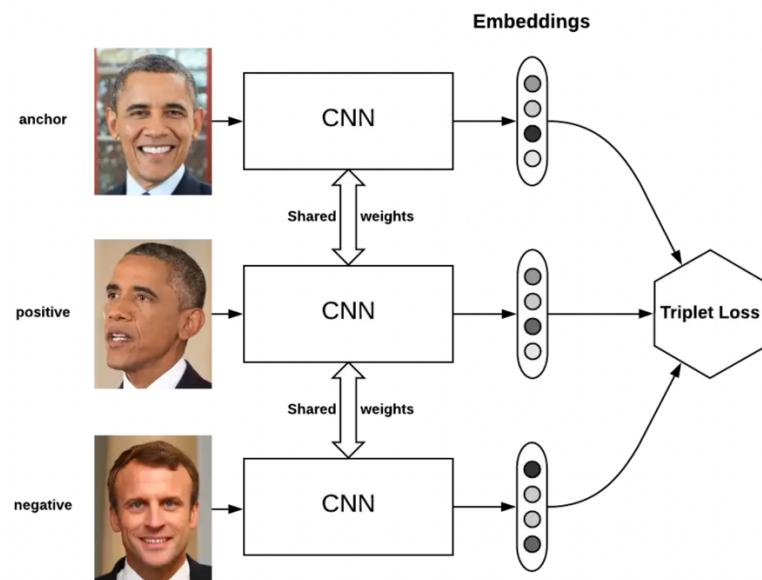
return iou_matrix

```

triplet loss

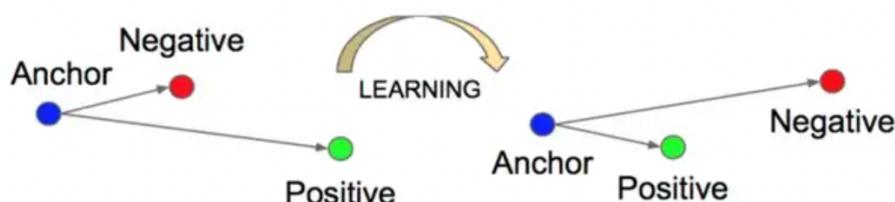
triplet loss به جای گرفتن دو ورودی، سه ورودی anchor و positive و negative میگیرد. Anchor ورودی مرجع است و میتواند از هر کلاسی باشد، positive ورودی است که کلاس یکسان با anchor دارد و negative ورودی است که کلاس متفاوت با anchor دارد.

برای مثال در مورد face recognition، همانگونه که در تصویر زیر نشان داده شده است، ورودی ها میتوانند به این صورت باشند که anchor و positive یک فرد یکسان را نشان بدهند و negative فردی متفاوت را نشان بدهد.



حال این سه ورودی را به یک شبکه CNN مشابه میدهیم تا از آنها بردار ویژگی استخراج کند و این سه بردار ویژگی را به تابع triplet loss میدهیم.

در تصویر زیر این فرایند به صورت شهودی نشان داده شده است.



Before (left) and after (right) minimizing triplet loss function (Source: [FaceNet](#))

ایده تابع triplet loss این است که فاصله anchor و positive را کمینه و فاصله anchor و negative را بیشینه کند.

برای این منظور اختلاف anchor و positive را با تابع فاصله $d(a, p)$ اندازه میگیریم و به صورت $d(a, p)$ نشان میدهیم که در حالت ایده آل باید پایین باشد. همچنین فاصله بین anchor و negative را با $d(a, n)$ نشان میدهیم که در حالت ایده آل باید زیاد باشد.

بنابراین هدف ما این است که $d(a, p) - d(a, n)$ کمتر از $d(a, n)$ باشد. به صورت ریاضی آن را به صورت $-d(a, n) < 0$ نشان میدهیم. اگرچه به دلیل اینکه نمیخواهیم مقدار خطای منفی باشد، اگر منفی شد آن را صفر در نظر میگیریم. از این رو میتوانیم تابع خطای را به صورت $\max(d(a, p) - d(a, n), 0)$ تعریف کنیم.

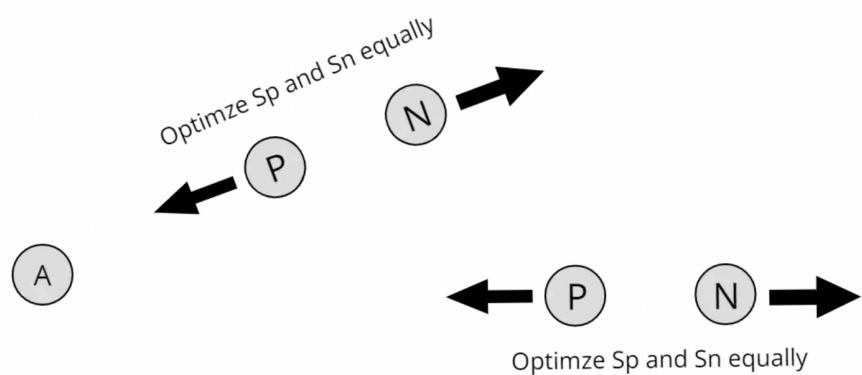
با این حال مشکل معادله قبلی این است که وقتی **positive** و **negative** فاصله یکسانی با **anchor** دارند و یا هنگامی که فقط کمی بیشتر به **anchor** نزدیک تر است نسبت به **negative**، خطای صفر خواهد شد و اصلاحی صورت نخواهد گرفت اگرچه بهتر بود **positive** بازهم به **anchor** نزدیک تر شود و **negative** باز هم دورتر شود. برای حل این مشکل یک مقدار ثابت به $d(a, p) - d(a, n)$ اضافه میکنیم و آنرا **margin** مینامیم. در نتیجه میتوانیم نهایتاً تابع خطای **triplet loss** را به صورت زیر تعریف کنیم.

$$l = \max(d(a, p) - d(a, n) + M, 0)$$

Circle loss

حال که **triplet loss** را درک کردیم میتوانیم ایده **circle loss** را بررسی کنیم. در مقاله مربوطه ادعا شده است که روش قبلی بهینه سازی انعطاف پذیری ندارد. برای نمونه مثال زیر را درنظر بگیرید که در آن **margin** بین دو جفت **positive** و **negative** یکسان است اما یکی از آن دو جفت نسبت به دیگری بسیار به **anchor** نزدیک تر است.

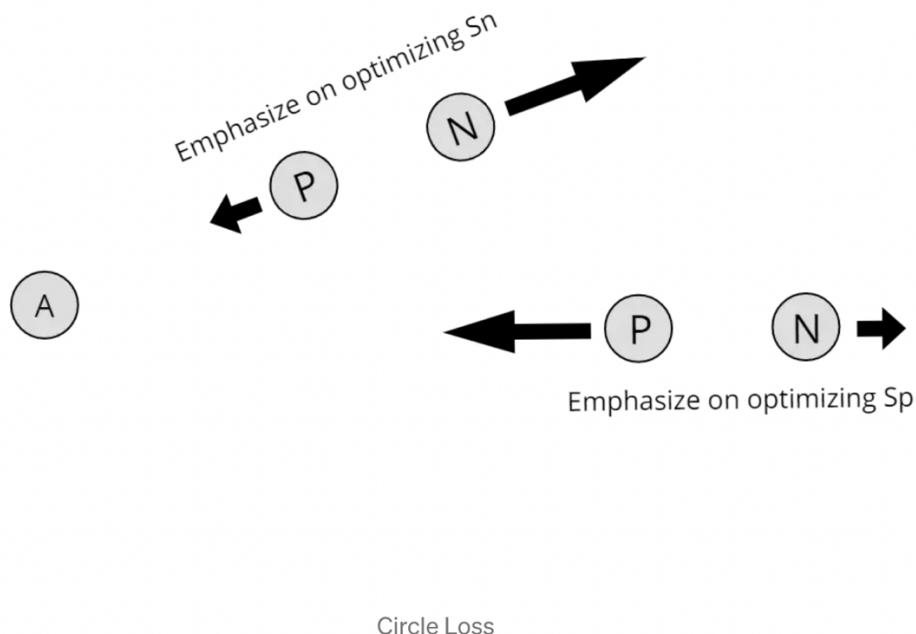
این دو جفت را به صورت یکسان میبیند و سعی میکند هر دو آنها را با نزدیک کردن **positive** و دور کردن **negative** به صورت یکسان بهینه کند.



Triplet Loss

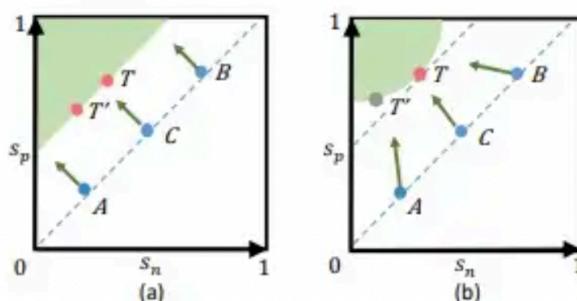
Sp is the within-class similarity score, and **Sn** is between class similarity score.
Note that the similarity score is inversely proportional to distance.

با این حال این روش خیلی هم بهینه نیست. وقتی anchor به اندازه کافی به positive نزدیک است باید روی دور کردن negative بیشتر تمرکز کنیم. و وقتی anchor دور هستند باید روی نزدیک کردن positive بیشتر تمرکز کنیم. این ایده ایست که circle loss پیاده سازی کرده است.



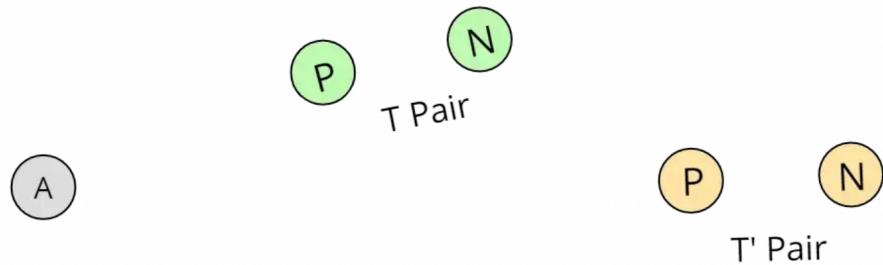
Circle loss با اعمال جریمه های متفاوت به s_n و s_p بهینه سازی انعطاف پذیر تری را ارائه میدهد. از این رو ما $(an * Sn - ap * Sp)$ را به $(Sn - Sp)$ تبدیل میکنیم که در آن an و ap دو فاکتور وزن مختلف و مستقل هستند و به s_n و s_p اجازه میدهد با سرعت های مختلفی یادبگیرند.

علاوه بر این circle loss نقطه همگرایی مشخص تری ارائه میدهد. قبلا در triplet loss هر دو جفت ها بهینگی یکسانی داشتند. برخلاف آن در circle loss جفتی را ترجیح میدهد که خیلی به anchor نزدیک نباشد (زیرا باعث میشود به anchor نزدیک باشد) و همچنین خیلی هم از anchor دور نباشد (زیرا باعث میشود negative دور باشد).



Comparison between the popular optimization manner of reducing $(s_n - s_p)$ and the proposed optimization manner of reducing $(anS_n - apS_p)$. [Source]

در این مثال هر دو جفت T و T' margin یکسانی دارند، سایر توابع loss نظری triplet در این وضعیت دچار ابهام میشوند.
 با این حال circle loss، T را ترجیح میدهد و یک هدف مشخص برای همگرایی تعیین میکند. مرز تصمیم نیز از $(an * Sn - ap * Sp) = m$ به $(Sn - Sp) = m$ تبدیل میشود که یک مرز دایره‌ای ایجاد میکند و نامگذاری این روش هم به همین دلیل است.



Assuming the same margin for both pairs, T is preferred in Circle Loss.

Reference:

<https://medium.com/vitrox-publication/understanding-circle-loss-bdaa576312f7#:~:text=Previously%20in%20Triplet%20Loss%20it,be%20too%20far%20from%20anchor>