Computer Vision
Dr. Mohammadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW12

۱. Underfitting یعنی مدل نه روی داده‌های آموزشی و نه روی داده‌های اعتبارسنجی دقت و عملکرد قابل قبولی ارائه ندهد یعنی مدل یا الگوریتم ما به اندازه کافی با داده‌ها مطابقت و همچنین تعمیم‌پذیری ندارد.

دلایل ایجاد Underfitting :

بایاس زیاد و واریانس کم داده‌ها

اندازه مجموعه داده آموزشی مورد استفاده کافی نیست.

گاهی ممکن است داده‌های زیادی داشته باشیم اما کیفیت داده‌ها مناسب نباشد. یعنی مجموعه‌ی کاملی از ویژگی‌ها نباشند.

مدل خیلی ساده است و توانایی زیادی برای پردازش داده‌ها ندارد.

داده‌های آموزشی نویز دارند.

الگوریتم یادگیری خوب نیست.

راهکارهایی برای رفع Underfitting :

افزایش پیچیدگی و توانایی مدل

افزایش تعداد ویژگی‌ها، انجام feature engineering

حذف نویز از داده ها و جمع‌آوری داده‌های تمیزتر

افزایش تعداد ایپاک‌ها یا مدت زمان آموزش برای دریافت نتایج بهتر

Overfitting یعنی عملکرد و دقت مدل روی داده‌های آموزشی خوب باشد اما روی داده‌های validation خوب نباشد به عبارتی مدل قابلیت تعمیم‌دهی نداشته باشد. هنگامی که یک مدل با

داده‌های زیادی آموزش می‌بیند، شروع به یادگیری و حفظ کردن نویز و ورودی داده‌های نادرست در مجموعه داده‌های ما می‌کند.

دلایل ایجاد Overfitting :

واریانس بالا و بایاس کم

پیچیدگی زیاد مدل و حفظ کردن داده‌ها

اندازه‌ی داده‌های آموزشی

راهکارهایی برای رفع Overfitting :

افزایش داده‌های آموزشی

کاهش پیچیدگی مدل و کاهش ابعاد شبکه

توقف زودهنگام یا early stopping در طول آموزش، یعنی اگر از یه حدی مدل بهتر نشد آموزش را متوقف کنیم.

انجام عملیات منظم‌سازی (داده‌افزایی، dropout، جریمه‌ی پارامترها، منظم‌سازی L2 و L1 و ..)

(لینک کمکی)

۲. ابتدا با استفاده از ابزار labelme صورت بازیکنان را لیبل می‌زنیم و محتوای فایل json خروجی را به صورت زیر لود می‌کنیم.

```
f = open("Melli.json", encoding="utf8")
data = json.load(f)
f.close()
```

سپس مختصات باکس تصاویر صورت را در لیست points ذخیره می‌کنیم.

```
points = []
for i, shape in enumerate(data["shapes"]):
  points.append([shape['points'][0][0], shape['points'][0][1], shape['points'][2][0], shape['points'][2][1]])
  print(f"\nFace{i + 1}:\n{points[i]}")
```

```
Face1:
[151.27118644067798, 123.72881355932203, 251.271186440678, 238.135593220339]

Face2:
[307.2033898305085, 125.42372881355934, 406.3559322033899, 237.28813559322035]

Face3:
[481.77966101694915, 77.11864406779661, 577.542372881356, 181.35593220338984]

Face4:
[669.0677966101695, 122.88135593220339, 754.6610169491526, 227.11864406779662]

Face5:
[829.2372881355932, 118.64406779661017, 902.1186440677966, 214.40677966101697]

Face6:
```

با استفاده از الگوریتم selective search از تصویر proposal استخراج می‌کنیم. این الگوریتم بر اساس segmentation کار می‌کند. همانطور که می‌بینیم خروجی آن برای تصویر ما ۸۷۵۰ proposal است. (لینک کمکی)

```
im = cv2.imread("Melli.jpg")
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(im)
ss.switchToSelectiveSearchFast()
rects = ss.process()
print('Total Number of Region Proposals: {}'.format(len(rects)))
```

```
Total Number of Region Proposals: 8750
```

در فایل کد ضمیمه‌شده برخی از این proposal ها نمایش داده‌شده‌اند.

سپس تابع IOU دو باکس را به صورت زیر تعریف می‌کنیم. (لینک کمکی)

```python
def get_iou(a, b):
    x1 = max(a[0], b[0])
    y1 = max(a[1], b[1])
    x2 = min(a[2], b[2])
    y2 = min(a[3], b[3])

    width = (x2 - x1)
    height = (y2 - y1)

    if (width < 0) or (height < 0):
        return 0.0
    area_overlap = width * height

    area_a = (a[2] - a[0]) * (a[3] - a[1])
    area_b = (b[2] - b[0]) * (b[3] - b[1])
    area_combined = area_a + area_b - area_overlap

    iou = area_overlap / (area_combined)
    return iou
```

IOU دو باکس از تقسیم مساحت ناحیه‌ی اشتراک بر اجتماع دو ناحیه بدست می‌آید.

در مرحله‌ی بعد، هر proposal را با هر ground_truth مقایسه کرده و IOU دو باکس را بدست می‌آوریم. سپس به مقادیر زیر 0.3 لیبل BackGround ، به مقادیر بالای 0.6 لیبل Face و به مقادیر بین این دو هیچ لیبلی اختصاص نمی‌دهیم.

```python
result = []
for proposal in rects:
  for gt in points:
    x, y, w, h = proposal
    boxA = [x, y, x + w, y + h]
    iou = get_iou(boxA, gt)
    if(iou > 0.6):
      result.append([boxA, "Face"])
    if(iou < 0.3):
      result.append([boxA, "BackGround"])
    else:
      continue
```

در انتها نیز ۵ پنجره به همراه لیبل آن‌ها را نمایش می‌دهیم.

Class: Face

Class: Face

Class: BackGround

Class: BackGround

Class: BackGround

۳. برای این سوال ابتدا تصویر خود را می‌خوانیم. سپس فایل json سوال قبل را دوباره لود کرده و مختصات باکس‌ها را در لیست points ذخیره می‌کنیم. سپس این لیست را بعنوان BoundingBox ها به توابع کتابخانه‌ی imgaug می‌دهیم.

```
ia.seed(1)
image = cv2.imread("Melli.jpg")

bbs = ia.augmentables.bbs.BoundingBoxesOnImage(
    [ia.augmentables.bbs.BoundingBox(x1=p[0], y1=p[1], x2=p[2], y2=p[3]) for p in points], shape=image.shape)
```

۱۰ تبدیل هندسی خود را به صورت زیر تعریف می‌کنیم.

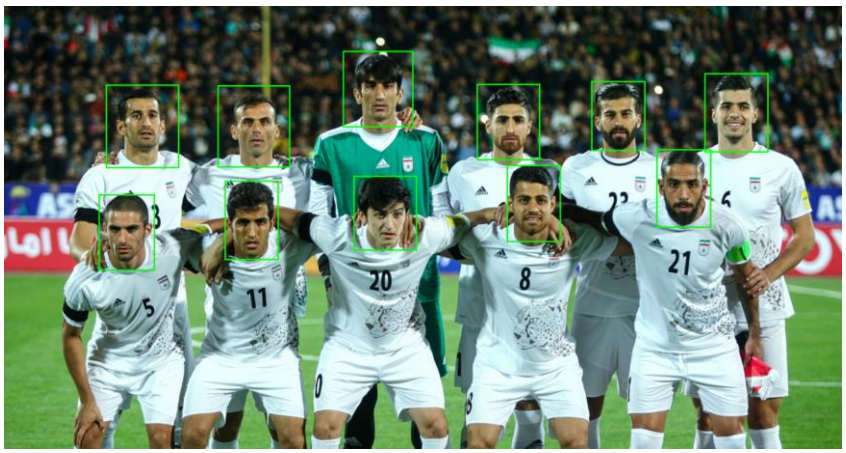
```
seq = iaa.Sequential([
    iaa.Crop(percent=(0, 0.1)),
    iaa.Affine(rotate=(-45, 45)),
    iaa.Affine(translate_px={"x": 40, "y": 60}),
    iaa.Affine(scale=(0.5, 0.7)),
    iaa.Affine(shear=(-16, 16)),
    iaa.Affine(translate_percent={"x": -0.20}, mode=ia.ALL, cval=(0, 255)),
    iaa.ScaleX((0.5, 1.5)),
    iaa.ScaleY((0.5, 1.5)),
    iaa.TranslateX(px=(-20, 20)),
    iaa.ShearX((-20, 20))
])
```

حال ۱۰ تبدیل را روی عکس اعمال می‌کنیم و مختصات قبلی و جدید باکس‌ها را نمایش می‌دهیم.

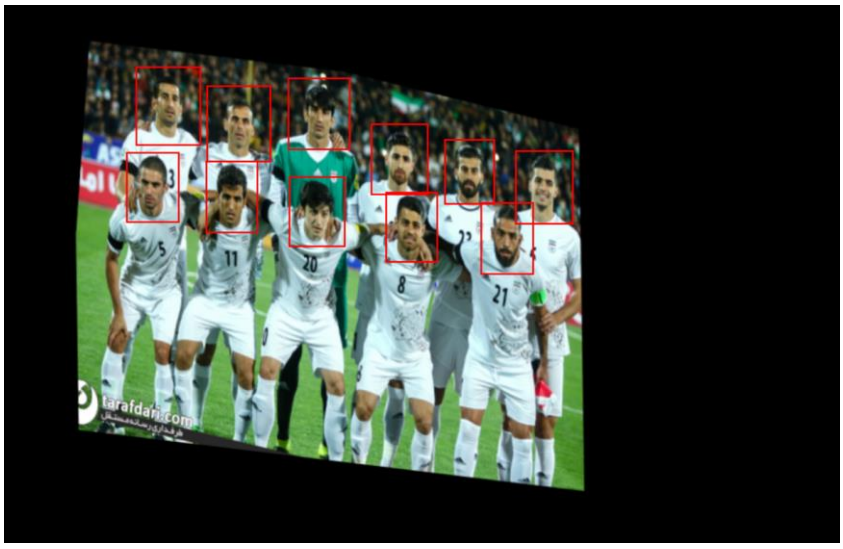
```
image_aug, bbs_aug = seq(image = image, bounding_boxes = bbs)

for i in range(len(bbs.bounding_boxes)):
    before = bbs.bounding_boxes[i]
    after = bbs_aug.bounding_boxes[i]
    print("BB %d: (%.4f, %.4f, %.4f, %.4f) -> (%.4f, %.4f, %.4f, %.4f)" % (
        i,
        before.x1, before.y1, before.x2, before.y2,
        after.x1, after.y1, after.x2, after.y2)
    )
```

```
BB 0: (151.2712, 123.7288, 251.2712, 238.1356) -> (181.9230, 191.4931, 255.1774, 280.0155)
BB 1: (307.2034, 125.4237, 406.3559, 237.2881) -> (263.7980, 212.5133, 336.1591, 299.2441)
BB 2: (481.7797, 77.1186, 577.5424, 181.3559) -> (358.2815, 203.5043, 427.5102, 284.7518)
BB 3: (669.0678, 122.8814, 754.6610, 227.1186) -> (454.3463, 257.1038, 518.1403, 337.0532)
BB 4: (829.2373, 118.6441, 902.1186, 214.4068) -> (539.4221, 274.8553, 594.9551, 347.5704)
BB 5: (986.8644, 108.4746, 1074.1525, 216.1017) -> (621.0679, 288.2976, 686.3548, 370.7080)
BB 6: (919.9153, 214.4068, 994.4915, 321.1864) -> (581.3453, 348.4926, 639.6920, 428.7194)
BB 7: (710.5932, 235.5932, 784.3220, 340.6780) -> (470.9391, 335.5326, 528.5394, 414.5288)
BB 8: (497.0339, 250.8475, 582.6271, 353.3898) -> (358.6082, 318.1881, 422.1087, 397.0152)
BB 9: (316.5254, 258.4746, 390.2542, 366.1017) -> (263.2272, 300.0626, 321.2679, 380.7423)
BB 10: (141.1017, 277.1186, 217.3729, 381.3559) -> (170.8581, 289.7910, 229.6703, 368.5506)
```

ورودی:



خروجی:



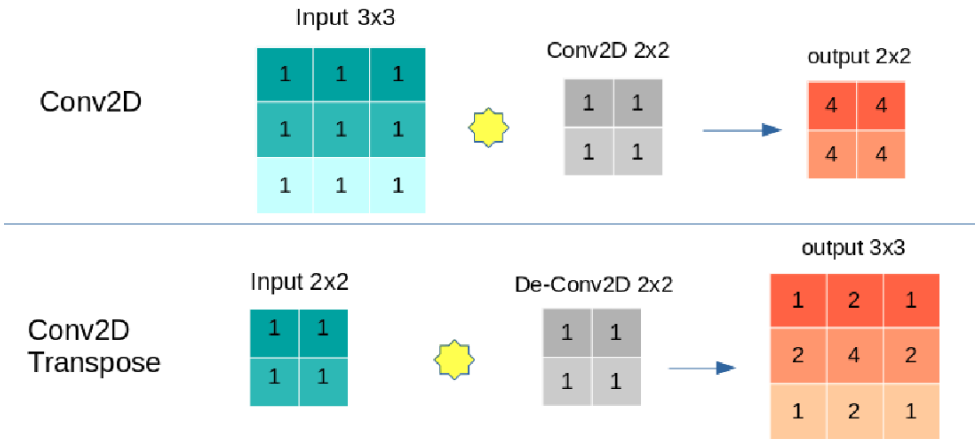
همانطور که مشاهده می‌کنیم، تبدیل‌هایی که روی تصویر اعمال کردیم، روی باکس‌ها نیز اعمال شده‌اند. (لینک کمکی)

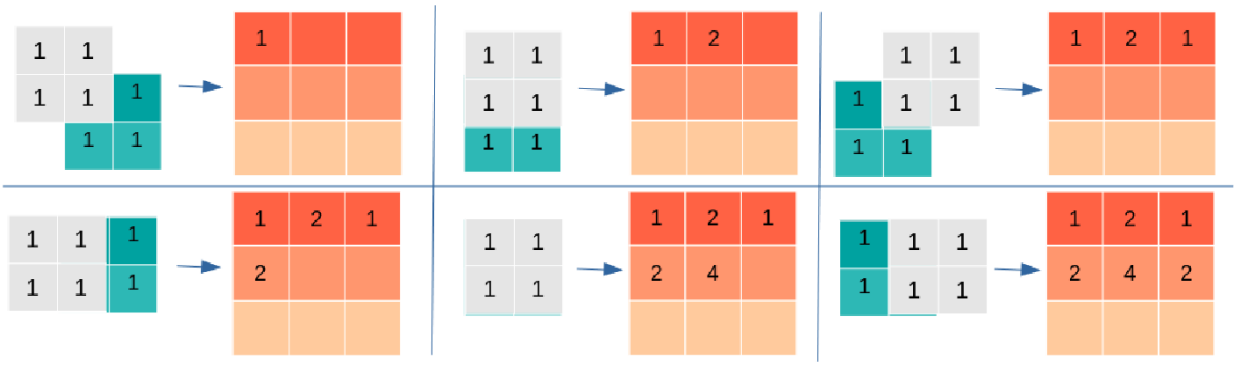
۴. ۱) padding = same به معنای این است که ما zero-padding داریم و نوع دیگر آن padding = valid است یعنی کلا padding نداریم. نوع اول باعث می‌شود که خروجی ما هم-اندازه با ابعاد ورودی باشد ولی نوع اول باعث تغییر ابعاد تصویر در خروجی می‌شود.

۲) تابع فعال سازی با پس از محاسبه مجموع وزن‌ها و اضافه کردن بایاس به آن تصمیم می‌گیرد که آیا یک نورون باید فعال شود یا نه. هدف از تابع فعال‌سازی، وارد کردن غیرخطی بودن به خروجی یک نورون است. چون ضرب و جمع اعمالی خطی هستند و یک شبکه عصبی بدون تابع فعال‌سازی در اصل فقط یک مدل رگرسیون خطی است. تابع فعال‌سازی تبدیل غیرخطی را به ورودی انجام می دهد و آن را قادر به یادگیری و انجام وظایف پیچیده‌تر می‌کند. (لینک کمکی)

۳) شبکه عصبی باید با مقداری وزن شروع شود و سپس به طور مکرر آن‌ها را به مقادیر بهتر به روز کند و یکی از مسائل مهم مقداردهی وزن‌های اولیه است. زیرا ممکن است مقداردهی اولیه‌ی اشتباه منجر به واگرایی شود. اصطلاح kernel_initializer یعنی از توزیع آماری یا تابع برای مقداردهی اولیه وزن‌ها استفاده می‌شود که بهتر از مقداردهی تصادفی است و به همگرایی بهتر کمک می‌کند. در صورت توزیع آماری، کتابخانه اعدادی را از آن توزیع آماری تولید می‌کند و به عنوان وزن‌های شروع استفاده می‌کند. مثلا در اینجا "he_normal" از توزیع گاوسی استفاده می‌شود و دارای واریانس خوبی است.

۴) conv2D عملیات کانولوشن را روی ورودی اعمال می‌کند و برعکس conv2DTranspose یک عملیات deconvolutional را روی ورودی اعمال می‌کند. Conv2D عمدتاً زمانی استفاده می‌شود که می‌خواهیم ویژگی‌ها را شناسایی کنیم و ممکن است شکل ورودی را کوچک کند. برعکس، Conv2DTranspose برای ایجاد ویژگی‌ها استفاده می‌شود و شکل ورودی را بزرگتر می‌کند. (لینک کمکی)





۵) ابتدا تابع double_conv_block() تعریف شده که از آن در توابع بعدی استفاده می‌شود. این تابع دو بلاک کانولوشنی با ابعاد فیلترهای ۳ در ۳، padding = same ، تابع فعالساز relu و توزیع "he_normal" برای مقداردهی اولیه‌ی کرنل‌ها استفاده شده است. تعداد فیلترها نیز به عنوان ورودی تابع دریافت می‌شود. یک تابع downsample_block() برای نمونه‌برداری پایین یا استخراج ویژگی تعریف شده تا در توابع بعدی استفاده شود. در این تابع با استفاده از تابع قبلی، دو بلاک کانولوشنی ساخته و تنها یک لایه max-pooling و یک لایه dropout با احتمال ۰.۳ به آن اضافه می‌شود. در نهایت، یک تابع upsample_block() برای مسیر گسترش U-Net تعریف می‌شود که ابتدا یک لایه conv2DTraspose با تعداد فیلترهایی که از ورودی تابع گرفته می‌شود، ابعاد فیلتر ۳ در ۳، padding = same و گام ۲ اضافه می‌شود. این کار یک تکنیک نمونه‌برداری است که اندازه‌ی تصویر را افزایش می‌دهد. سپس خروجی این بلاک با بلاک conv_features که پارامتر ورودی است، concatenate می‌شود. به دلیل این که اطلاعات

لایه‌های قبلی را با هم ترکیب کنیم تا بتوانیم پیش‌بینی دقیق‌تری داشته باشیم. سپس یک لایه dropout با احتمال ۰.۳ و دو بلاک کانولوشنی نیز اضافه می‌شود. تمام این لایه‌ها نیز به صورت sequential اضافه می‌شوند و توابع modular هستند.

۶) بهینه‌ساز تابع یا الگوریتمی است که ویژگی‌های شبکه عصبی مانند وزن‌ها و نرخ یادگیری را تغییر می دهد و به پارامترهای قابل یادگیری مدل مانند وزن و بایاس وابسته است. بنابراین، به کاهش loss کلی و بهبود دقت کمک می‌کند.

۷) تابع compile() تابع ضرر، بهینه‌ساز و معیارها را تعریف می‌کند. ربطی به وزن‌ها ندارد و می‌توان هر چند بار که بخواهیم مدلی را کامپایل کنیم، بدون اینکه برای وزنه‌های از پیش آموخته‌شده مشکلی ایجاد شود. برای آموزش به یک مدل کامپایل شده نیاز داریم. (زیرا آموزش از تابع ضرر و بهینه ساز استفاده می کند.) (لینک کمکی)

۸) categorical_crossentropy یک تابع ضرر است که برای بهینه‌سازی مسائل طبقه‌بندی چندکلاسه استفاده می‌شود و مسئله‌ی ما نیز همین است. فرمت ورودی نیز باید به صورت one-hot باشد.

۹) تکنیک توقف زودهنگام یا early stopping برای جلوگیری از overfit شدن است. به این صورت که هرگاه معیار دلخواه ما به یک حدی رسید آموزش متوقف می‌شود. در اینجا معیار توقف val_loss است و min_delta حد آستانه‌ای برای تعیین کمیت است. یعنی اگر val_loss کمتر از min_delta باشد، به‌عنوان عدم بهبود کمی تلقی می‌شود و آن را صفر رها می‌کنیم. patience تعداد ایپاک‌هایی است که پس از اینکه ضرر شروع به افزایش کرد صبر می‌کنیم تا آموزش متوقف شود. یعنی به مدل فرصت بیشتری می‌دهیم. در انتها نیز بهترین وزن‌ها ذخیره می‌شوند تا بتوان آن‌ها را بازیابی کرد. اگر False باشد، از وزن‌های مدل به دست‌آمده در آخرین مرحله آموزش استفاده می‌شود.

۱۰) همانطور که گفته شد، تابع کامپایل برای تعریف توابع ضرر، بهینه‌ساز و متریک‌هاست اما تابع fit() برای آغاز فرآیند آموزش مدل است تا پارامترها را بهینه کند. باید دیتاهای آموزشی، اعتبارسنجی، callbackها، تعداد ایپاک‌ها، سایز دسته‌ها و .. را در این تابع به مدل بدهیم.

۱۱) تعداد ایپاک تعداد مراحل آموزش است که چندبار تکرار شود و هر ایپاک یک گذر کامل از مجموعه داده آموزشی است. اما batch_size تعداد نمونه‌هایی از تابع آموزشی است که در هربار وارد مدل می‌شوند. زیرا ما نمی‌توانیم یکباره کل دیتا را وارد شبکه کنیم لذا آن را به دسته‌های کوچک‌تر به نام batch تقسیم می‌کنیم.

۵. این سوال در فایل Q5 پیاده‌سازی شده است.