



دانشگاه صنعتی ارومیه
وزارت علوم، تحقیقات و فناوری

پروژه پایانی کارشناسی رشته مهندسی کامپیوتر

عنوان پروژه:

تشخیص اشیا با استفاده از یادگیری ماشین و ابزار Tensorflow

تهیه و نگارش:

پارسا شریفی

شماره دانشجویی:

۹۷۱۱۵۲۱۳۴

استاد راهنما:

دکتر اردلان قاسم زاده

فهرست مطالب

4	چکیده
5	مقدمه
6	کتابخانه ها و ابزار ها
6	TensorFlow
6	OpenCV
6	NumPy
6	Matplotlib
6	Pandas
6	Keras
6	Python Imaging Library (PIL) یا همان Pillow
7	Protocol Buffers (protobuf)
7	PyQt5
7	Jupyter Notebooks
7	Virtual Environments (venv)
7	Git
7	CUDA (Compute Unified Device Architecture)
7	cuDNN (CUDA Deep Neural Network)
8	اهداف پروژه
8	بیان واضح اهداف
8	محدوده و اشیاء هدف
8	برنامه های کاربردی
8	معیار های عملکرد
8	نتایج مورد انتظار
9	جمع آوری داده ها
9	منبع و توضیحات مجموعه داده
10	حاشیه نویسی (Annotation) و ابزار حاشیه نویسی سفارشی
13	چالش ها و پیش پردازش
14	آموزش و یادگیری مدل
15	چرا باید از CUDA و cuDNN استفاده کنیم؟
15	شتاب پردازنده گرافیکی
15	بهینه سازی های cuDNN
15	سازگاری

15مقیاس پذیری
16cuDNN و CUDA برای ما چگونه کار میکنند؟
16محاسبات تسریع شده
16عملیات شبکه عصبی بهینه شده
16مقیاس پذیری
16عملکرد بهبود یافته
17مدل انتخابی
17معماری
18چگونه کار میکنند؟
19Underfitting و Overfitting
20سایر ویژگی های مدل
21آموزش دادن مدل
27ارزیابی مدل
31بارگذاری آخرین چک پوینت مدل آموزش دیده و تشخیص اشیاء
34جمع بندی و نتیجه گیری

چکیده

در این پروژه دانشگاهی، ما اصول تشخیص شی را در حالی که چیزها را در دسترس و عملی نگه می داریم، بررسی می کنیم. رویکرد ما بر روی استفاده از TensorFlow، یک چارچوب یادگیری ماشینی همه کاره، و استقرار یک مدل TensorFlow Zoo از پیش آموزش دیده در ماشین محلی ما متمرکز است. ما یک محیط ایزوله ایجاد کرده ایم تا فرآیند توسعه روان و قابل مدیریت را تضمین کنیم، و برای سهولت آزمایش به Python Jupyter notebook تکیه کرده ایم.

هدف اصلی ما این است که نشان دهیم که تشخیص مؤثر شیء را می توان بدون وارد شدن به پیچیدگی های یادگیری عمیق یا زیرساخت های دقیق به دست آورد. با استفاده از یک مدل از پیش آموزش دیده، هدف ما ارائه بینشی در مورد جنبه عملی اجرای تشخیص اشیا در یک محیط آموزشی و کنترل شده است.

این پروژه شامل مراحل مانند آماده سازی داده ها، یکپارچه سازی مدل و ارزیابی عملکرد است که همه بر روی یک کامپیوتر محلی انجام می شوند. اگرچه رویکرد ما ممکن است پیشگامانه نباشد، اما به عنوان یک تجربه یادگیری ارزشمند عمل می کند و نشان می دهد که چگونه می توان تکنیک های یادگیری ماشین را در یک کار معمول بینایی رایانه در یک محیط ساده به کار برد.

از طریق این تحقیق، هدف ما ارائه یک مقدمه عملی برای تشخیص اشیا در چارچوب یک محیط توسعه محلی، با تاکید بر کاربردی بودن و دسترسی به دانشجویان و فراگیران است.

مقدمه

در زمینه بینایی کامپیوتر، تشخیص اشیا یک کار کلیدی با برنامه های کاربردی متعدد در دنیای واقعی است. در پروژه دانشگاهی خود، سفری را آغاز می کنیم تا این کار را ابهام زدایی کنیم و آن را برای دانشجویان و دانش آموزان به طور یکسان در دسترس قرار دهیم. پروژه ما مبتنی بر سادگی است و بر استفاده از TensorFlow 2 و یک مدل از قبل آموزش دیده شده TensorFlow Zoo در ماشین محلی ما تمرکز دارد.

برای اطمینان از یک فرآیند توسعه کنترل شده و قابل مدیریت، ما یک محیط ایزوله ایجاد کرده ایم که از درگیری ها و عوارض احتمالی محافظت می کند. ما نوتبوک های پایتون ژوپیتر را به عنوان ابزار اصلی خود برای آزمایش انتخاب کرده ایم که امکان گردش کار یکپارچه و تعاملی را فراهم می کند.

هدف اصلی ما تاکید بر جنبه های عملی تشخیص شی است. با استفاده از یک مدل از پیش آموزش دیده موجود در TensorFlow Zoo، هدف ما نشان دادن این است که تشخیص شیء مؤثر را می توان بدون فرو رفتن در معماری های پیچیده یادگیری عمیق یا تکیه بر منابع محاسباتی گسترده به دست آورد.

سفر پروژه ما شامل مراحل مانند آماده سازی داده ها، یکپارچه سازی مدل، و ارزیابی دقیق عملکرد است که همگی در رایانه شخصی ما انجام می شوند. در حالی که رویکرد ما ممکن است تکنیک های انقلابی را معرفی نکند، به عنوان یک تجربه یادگیری ارزشمند عمل می کند، که نشان می دهد چگونه یادگیری ماشین را در یک زمینه آموزشی و عملی به کار گیریم.

در این مقدمه، ما با تاکید بر سادگی آن و در دسترس بودن ابزار انتخابی خود، زمینه را برای پروژه خود فراهم کردیم. ما معتقدیم که با به اشتراک گذاشتن بینش و تجربیات خود، می توانیم به دانشجویان کمک کنیم تا تشخیص اشیا را در یک محیط آشنا و کنترل شده شروع کنند.

کتابخانه ها و ابزار ها

در این پروژه از کتابخانه ها و ابزار های نسبتاً زیادی استفاده شده است. در این جا ما برخی از مهم ترین ها و کارآمد ترین ها را لیست کرده و درباره آن ها مختصر توضیحی خواهیم داد.

TensorFlow

TensorFlow یک چارچوب یادگیری ماشین منبع باز است که توسط گوگل توسعه یافته است. TensorFlow یک API کاربرپسند برای ساخت و آموزش مدل های یادگیری ماشین ارائه می کند. در این پروژه، از TensorFlow برای کار با مدل ها و چارچوب های تشخیص اشیا مانند TensorFlow Zoo استفاده کرده ایم.

OpenCV

OpenCV یک کتابخانه محبوب برای وظایف بینایی کامپیوتر است. این طیف گسترده ای از عملکردها را برای پردازش تصویر و ویدئو ارائه می دهد که آن را برای کارهایی مانند پیش پردازش تصویر، استخراج ویژگی و تجسم بسیار ارزشمند می کند.

NumPy

NumPy یک کتابخانه اساسی برای محاسبات عددی در پایتون است. پشتیبانی از آرایه ها و ماتریس های چند بعدی بزرگ، همراه با توابع ریاضی برای کار بر روی این آرایه ها را ارائه می دهد. NumPy برای دستکاری داده ها و عملیات عددی در یادگیری ماشین بسیار مهم است.

Matplotlib

Matplotlib یک کتابخانه قدرتمند برای ایجاد تجسم های ثابت، متحرک و تعاملی در پایتون است. اغلب برای تولید نمودارها، نمودارها و نمودارها برای تجسم داده ها و عملکرد مدل استفاده می شود.

Pandas

Pandas یک کتابخانه برای دستکاری و تجزیه و تحلیل داده ها است. این ساختارهای داده مانند DataFrames و Series را فراهم می کند که برای سازماندهی، تمیز کردن و کاوش مجموعه داده ها ضروری هستند.

Keras

Keras یک API یادگیری عمیق منبع باز است که به عنوان یک رابط برای چارچوب های یادگیری عمیق مختلف از جمله TensorFlow عمل می کند. این فرآیند ساخت، آموزش و ارزیابی شبکه های عصبی را ساده می کند و آن را به گزینه ای محبوب برای توسعه مدل های یادگیری ماشین تبدیل می کند.

Python Imaging Library (PIL) یا همان Pillow

PIL برای باز کردن، دستکاری و ذخیره فرمت های مختلف فایل های تصویری استفاده می شود. Pillow یک fork مدرن از PIL است و اغلب برای کارهای پردازش تصویر ترجیح داده می شود.

Protocol Buffers (protobuf)

Protocol Buffers روشی است که توسط گوگل برای سریال سازی داده های ساخت یافته توسعه یافته است. در پروژه ها، protobuf ممکن است برای تبادل کارآمد داده بین بخش های مختلف pipeline تشخیص شی یا برای تعریف قالب های داده سفارشی استفاده شود.

PyQt5

PyQt5 یک اتصال پایتون برای چارچوب برنامه Qt است. ما را قادر می سازد تا برنامه های دسکتاپ را با رابط کاربری گرافیکی (GUI) ایجاد کنیم. در پروژه ها، PyQt5 ممکن است برای طراحی و توسعه یک رابط کاربر پسند برای تعامل با مدل تشخیص شی یا برای تجسم نتایج استفاده شده باشد.

Jupyter Notebooks

Jupyter Notebook یک ابزار مبتنی بر وب تعاملی برای ایجاد و به اشتراک گذاری اسناد است که کدهای زنده، معادلات، تجسم ها و متن روایت را ترکیب می کند. این به ویژه برای انجام آزمایش ها، مستندسازی کد و به اشتراک گذاری یافته های تحقیقاتی مفید است و آن را به انتخابی عالی برای تجزیه و تحلیل داده ها و پروژه های یادگیری ماشین تبدیل می کند.

Virtual Environments (venv)

از محیط های مجازی برای ایجاد محیط های ایزوله پایتون برای مدیریت وابستگی ها و نسخه های بسته استفاده می شود. آنها اطمینان می دهند که وابستگی های پروژه ما با بسته های سراسر سیستم تداخلی ندارد.

Git

Git یک سیستم کنترل نسخه توزیع شده است که به طور گسترده برای ردیابی تغییرات در مخازن کد استفاده می شود. این امکان توسعه مشارکتی، مدیریت نسخه و ادغام آسان با پلتفرم هایی مانند GitHub، GitLab یا Bitbucket را فراهم می کند و به اشتراک گذاری کد و همکاری یکپارچه بین اعضای تیم را امکان پذیر می کند.

CUDA (Compute Unified Device Architecture)

CUDA یک پلت فرم محاسباتی موازی و API است که توسط NVIDIA برای استفاده از منابع GPU (واحد پردازش گرافیک) در محاسبات علمی و عددی توسعه یافته است. این به طور قابل توجهی آموزش یادگیری عمیق و سایر وظایف محاسباتی فشرده را با بارگذاری آنها در GPU تسریع می کند.

cuDNN (CUDA Deep Neural Network)

cuDNN یک کتابخانه با شتاب GPU برای شبکه های عصبی عمیق است. عملکرد چارچوب های یادگیری عمیق مانند TensorFlow را با ارائه اولیه های شتاب دهنده GPU سطح پایین برای عملیات هایی که معمولاً در شبکه های عصبی عمیق استفاده می شوند، بهینه می کند.

اهداف پروژه

در این پروژه دانشگاهی، ما سفری را آغاز می‌کنیم تا اصول تشخیص اشیا را بررسی کنیم و در عین حال اهداف را ساده و آموزشی نگه داریم. هدف اصلی ما این است که بینشی در مورد فرایندها و نتایج تشخیص اشیا با استفاده از تکنیک‌های یادگیری ماشین و یادگیری عمیق به دست آوریم و در عین این حال وارد پیچیدگی‌ها دنیای یادگیری عمیق نشویم.

بیان واضح اهداف

هدف اصلی این پروژه توسعه یک مدل پایه تشخیص اشیا با استفاده از روش‌های یادگیری ماشین سنتی است. هدف ما درک مراحل ضروری در تشخیص اشیا، از آماده‌سازی داده تا ارزیابی مدل است.

محدوده و اشیاء هدف

دامنه این پروژه شامل اشیاء معمولی روزمره است که با استفاده از تکنیک‌های یادگیری ماشین قابل شناسایی هستند. ما روی اشیاء تخصصی یا دامنه خاص تمرکز نمی‌کنیم.

برنامه‌های کاربردی

اگرچه این پروژه در درجه اول آموزشی است، بینش‌های به دست آمده را می‌توان در برنامه‌های مختلف که در آن تشخیص اشیا نقش دارد، اعمال کرد. ما بر ارزش آموزشی پروژه تأکید می‌کنیم که می‌تواند به سناریوهای دنیای واقعی نیز تعمیم یابد.

معیارهای عملکرد

معیارهای عملکرد، مانند دقت، و یادآوری، برای ارزیابی اثربخشی مدل‌های تشخیص شیء ما استفاده می‌شوند. با مقایسه عملکرد مدل‌های یادگیری ماشین سنتی با مدل‌های یادگیری عمیق، هدف ما نشان دادن تفاوت‌ها در عملکرد و کارایی است.

نتایج مورد انتظار

نتایج مورد انتظار این پروژه شامل مدل‌های تشخیص شیء کاربردی با استفاده از روش‌های یادگیری ماشین سنتی است. تا پایان پروژه، ما پیش‌بینی می‌کنیم که بینش‌های ارزشمندی در مورد فرایندهای درگیر داشته باشیم و درک روشنی از عملکردهای تشخیص اشیاء مبتنی بر یادگیری ماشین و یادگیری عمیق داشته باشیم.

جمع آوری داده ها

در این بخش، فرآیند جمع‌آوری داده‌ها را شرح می‌دهیم که شامل استفاده از تصاویر و annotation های سفارشی ما برای آزمایش مدل و بررسی نرخ‌های یادگیری است:

منبع و توضیحات مجموعه داده

مجموعه داده مورد استفاده برای این پروژه کاملاً شامل تصاویری است که با استفاده از وب کم گرفته شده است. این انتخاب منعکس کننده‌ی یک منبع عملی و واقعی از داده‌ها برای آزمایش مدل های تشخیص اشیا است. تصاویر موجود در مجموعه داده، در درجه اول به دلیل محدودیت‌های ذاتی سخت‌افزار وب‌کم، تغییراتی را در وضوح نشان می‌دهند. علیرغم کیفیت متفاوت، این مجموعه داده دنیای واقعی منبع ارزشمندی برای کاوش تشخیص اشیا و چالش های آن است.

در این بخش ما با استفاده از یکی از کتابخانه های قدرتمند پایتون، کتابخانه ی OpenCV به فرآیند جمع آوری داده می‌پردازیم. بدین وسیله که با فراخوانی این کتابخانه و با استفاده از متد VideoCapture از طریق وب کم اقدام به ثبت عکس کرده و سپس آن ها را در دایرکتوری های ایجاد شده مربوط به هر مجموعه داده، ذخیره می‌کنیم. در اینجا برای شروع کار ما از تعداد ۵ عکس برای هر مجموعه داده استفاده کردیم. در کل برای شروع ۴ مجموعه داده با عنوان های ThumbsUp، ThumbsDown، LiveLong و ThankYou ساخته ایم که جزو زبان های اشاره ای می‌باشند.

```
labels = ['livelong', 'thumbsup', 'thumbsdown', 'thankyou']
number_imgs = 5
```

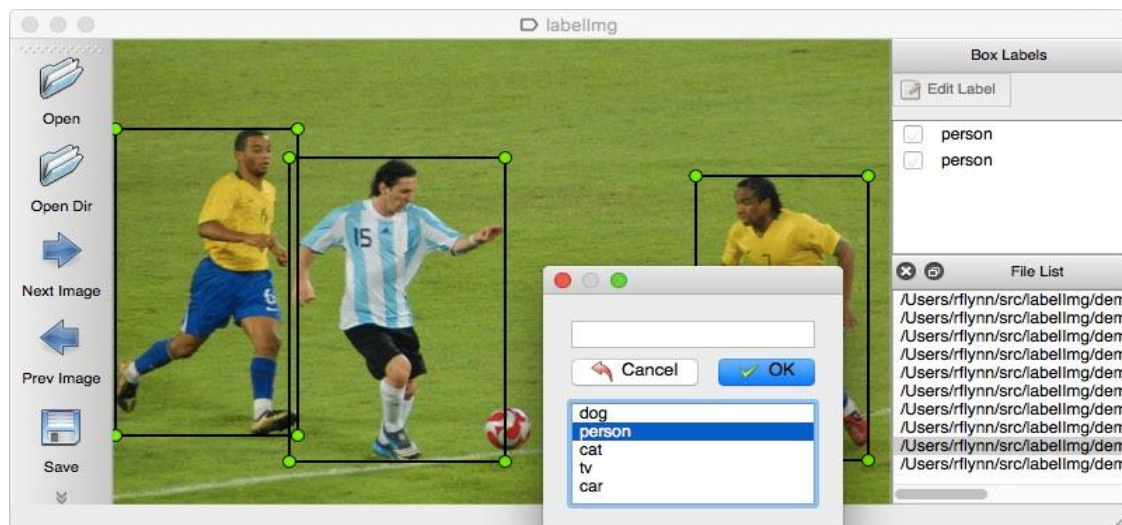
```
for label in labels:
    cap = cv2.VideoCapture(0)
    print('Collecting images for {}'.format(label))
    time.sleep(5)
    for imgnum in range(number_imgs):
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()
        imgname = os.path.join(IMGES_PATH, label, label+'_'+str(uuid.uuid1())+'.jpg')
        cv2.imwrite(imgname, frame)
        cv2.imshow('frame', frame)
        time.sleep(2)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

بعد از ذخیره کردن عکس ها به دایرکتوری های مدنظر رفته و عکس های ذخیره شده را بررسی کردیم تا اگر مشکلی در عکس ها اعم از تاری بودن عکس یا در کادر نبودن اشاره‌ی مورد نظر مربوط به مجموعه داده در آن ها مشاهده شد، آن ها را حذف کنیم یا در صورت نیاز اقدام به ثبت عکس های جدید و بهتری کنیم. هر مجموعه داده به انضمام داده های جمع آوری شده در مسیر Tensorflow\workspace\images\collectedimages قرار دارد.

حاشیه نویسی (Annotation) و ابزار حاشیه نویسی سفارشی

پس از ذخیره کردن عکس ها یا همان داده های ما، نوبت به حاشیه نویسی یا به اصطلاح همان Labeling یا ساختن Annotation ها می رسد. برای حاشیه نویسی دقیق مجموعه داده، از یک ابزار حاشیه نویسی سفارشی که از مخزن HumanSignal/labelImg در GitHub استفاده شده است. LabelImg، ابزار محبوب حاشیه نویسی تصویر که توسط Tzutalin با کمک ده ها مشارکت کننده ایجاد شده است، دیگر به طور فعال توسعه نمی یابد و به بخشی از جامعه Label Studio تبدیل شده است. این ابزار سفارشی امکان برچسب گذاری (Labeling) کارآمد اشیاء درون تصاویر را فراهم می کرد. حاشیه نویسی ها در فایل های XML ذخیره می شوند، که جزئیات ضروری تصویر را در بر می گیرد. این جزئیات شامل عرض، ارتفاع و تخصیص برچسب هایی است که به صورت دستی برای هر شیء مورد علاقه در تصاویر اعمال می شوند. بنابراین، حاشیه نویسی آموزش و ارزیابی مدل های تشخیص شی را تسهیل می کند.



ابتدا، ما به دو کتابخانه‌ی PyQt5 و LXML برای پردازش عکس ها و لیبل زدن نیاز داریم. پس از نصب (یا آپگرید کتابخانه ها در صورت وجود)، نیاز است تا ابزار LabelImg، که در بالا توضیح داده شده، را برای کار کردن آماده سازی کنیم. برای این کار کافیت تا مخزن را در یک مسیر مشخص که صرفاً به این ابزار نسبت داده شده، شبیه سازی یا به اصطلاح clone کنیم. این کار به سادگی انجام میگیرد.

```
LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')
```

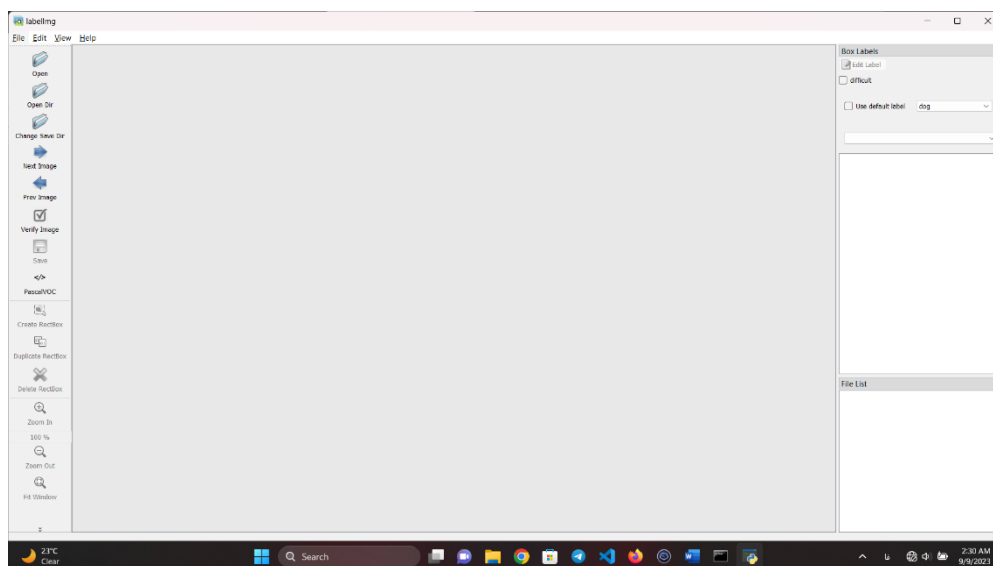
کلون کردن مخزن:

```
if not os.path.exists(LABELIMG_PATH):
    !mkdir {LABELIMG_PATH}
    !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}
```

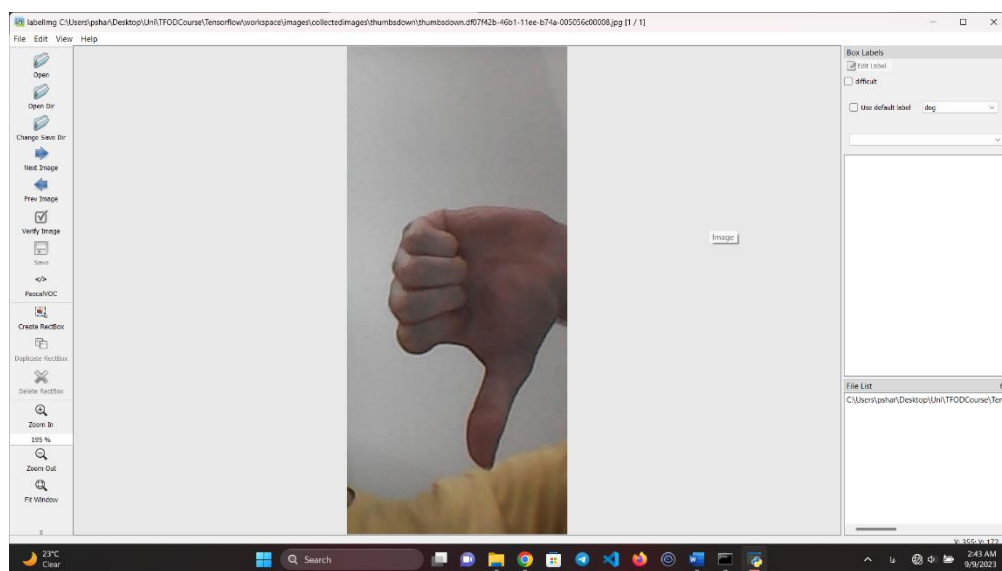
اجرای ابزار:

```
!cd {LABELIMG_PATH} && python labelImg.py
```

لازم به ذکر است که در قسمت {LABELIMG_PATH} کافیت مسیر مورد نظر برای clone شدن این مخزن را وارد کنیم. سپس تنها نیاز است تا با رفتن به مسیر مورد نظر و اجرای دستور python labelImg.py این ابزار را اجرا کنیم. محیطی به شکل زیر را برای ما به نمایش خواهد گذاشت.

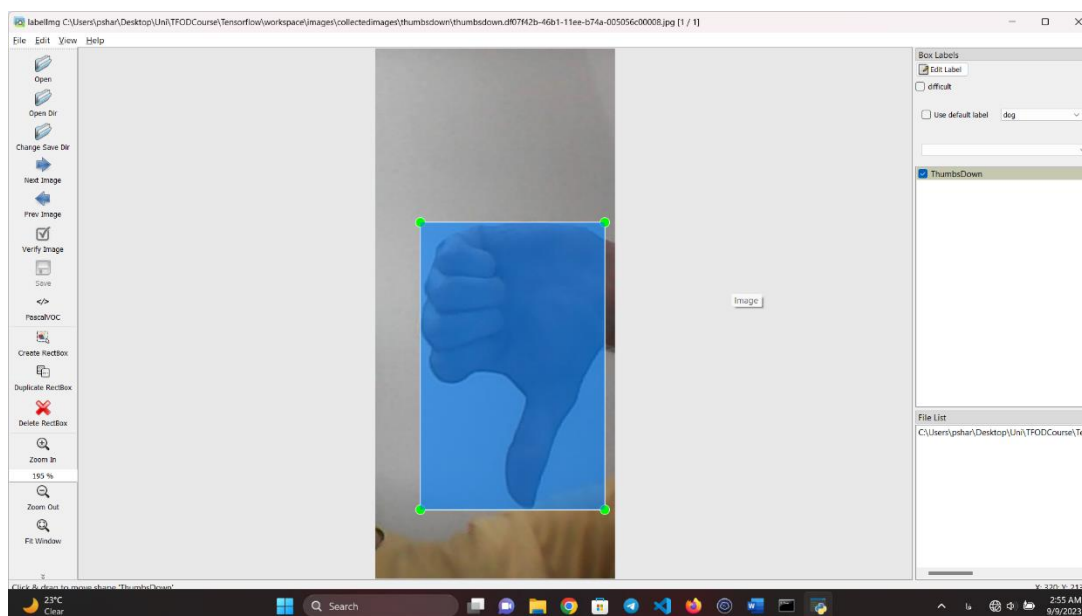


حال نیاز است تا از منوی سمت چپ از قسمت Open Dir به دایرکتوری هایی که داده ها را در آن ها ذخیره کردیم برویم. این کار بایستی برای هر مجموعه داده انجام بگیرد. در اینجا چون ما برای شروع از ۴ مجموعه داده که قبلا ذکر شد استفاده کردیم، نیاز هست که پس از حاشیه نویسی برای هر داده در هر دایرکتوری به دایرکتوری بعدی برویم. در اینجا ما فقط مراحل را برای مجموعه داده ی ThumbsDown ذکر میکنیم. پس در مرحله ی اول نیاز است تا به دایرکتوری ThumbsDown برویم.



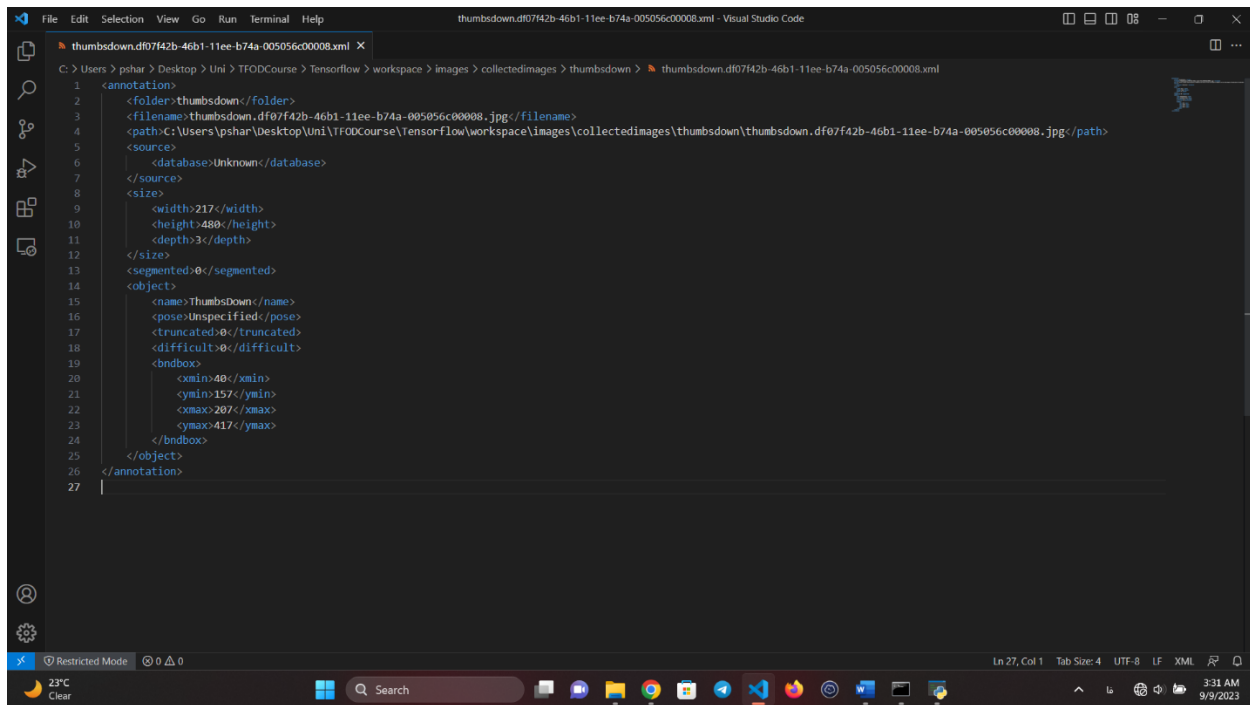
همچنین از قسمت File List میتوان به سایر داده ها یا همان عکس ها دسترسی داشت. با کلیک بر روی گزینه ی Create RectBox یا دکمه ی W بر روی کیبورد ما قادر به این خواهیم بود که یک باکس یا یک کادر برای قرار دادن شی مورد نظر داخل آن باکس جهت لیبل زدن بر روی عکس ایجاد کنیم. توصیه میشود این باکس تا جایی که امکانش فراهم است تنگ باشد و چیز دیگری بجز شی مورد نظر ما را در بر نگیرد. این کار باعث بهبود فرایند یادگیری و پایین

آمدن درصد خطای مدل ما در آینده خواهد شد. پس از کشیدن باکس در این مرحله باید نام مخصوصی برای لیبل مورد نظر خود انتخاب کنیم. لازم به ذکر است که این نام بایستی برای تمامی داده های دیگر در همان دایرکتوری یا به عبارت دیگر مجموعه داده ی ما یکی باشد. توصیه میشود به نحوه ی اختصاص دادن نام به لیبل دقت شود زیرا در آینده نام لیبل خاصی که به مدل برای یادگیری داده میشود بایستی حتما با این نام یکی باشد. در غیر این صورت، فرایند یادگیری مدل با خطا مواجه خواهد شد. پس از اختصاص دادن نام و ثبت آن حال کافیت تا با کلیک بر روی گزینه ی Save یا CTRL + S بر روی کیبورد، لیبل ها (یا لیبل ها را در صورت وجود بیشتر از یک لیبل) ذخیره کنیم.



همانطور که تصویر بالا میتوان مشاهده کرد، ما بدین وسیله اولین عکس یا داده ی خود را با نام ThumbsDown لیبل زده ایم. فایل Annotation لیبل مربوط به این عکس در همین دایرکتوری، با نام اصلی عکس و با فرمت XML ذخیره خواهد شد. این Annotation ها در آینده هم برای یادگیری مدل به عنوان داده های train و هم برای آزمایش کردن درستی مدل به عنوان داده های test مورد استفاده قرار میگیرند.

این Annotation ها شامل یک سری جزئیات در رابطه با داده های ما نظیر طول، عرض، عمق، نام و مختصات لیبل میباشد به صورت خودکار توسط ابزار LabelImg ویرایش شده و توصیه میشود از ایجاد هرگونه تغییری در پارامترهای آن خودداری شود زیرا این کار باعث اختلال و بروز مشکل در نحوه ی یادگیری مدل و عملکرد آن برای تشخیص داده های train میشود.



```
1 <annotation>
2   <folder>thumbsdown</folder>
3   <filename>thumbsdown.df07f42b-46b1-11ee-b74a-005056c00008.jpg</filename>
4   <path>C:\Users\pshar\Desktop\Uni\TFODCourse\Tensorflow\workspace\images\collectedimages\thumbsdown\thumbsdown.df07f42b-46b1-11ee-b74a-005056c00008.jpg</path>
5   <source>
6     <database>Unknown</database>
7   </source>
8   <size>
9     <width>217</width>
10    <height>480</height>
11    <depth>3</depth>
12  </size>
13  <segmented>0</segmented>
14  <object>
15    <name>ThumbsDown</name>
16    <pose>Unspecified</pose>
17    <truncated>0</truncated>
18    <difficult>0</difficult>
19    <bndbox>
20      <xmin>40</xmin>
21      <ymin>15</ymin>
22      <xmax>287</xmax>
23      <ymax>41</ymax>
24    </bndbox>
25  </object>
26 </annotation>
27
```

پس از اینکه تمامی مراحل بالا را برای تمامی داده ها و تمامی مجموعه داده ها انجام دادیم، لازم است تا داده ها به همراه فایل Annotation مربوط به خود آن داده را بنا به نیاز خود در پوشه های train و test خود که بایستی در همان مسیر Tensorflow\workspace\images ساخته شوند، انتقال می دهیم و با این کار یک گام به مرحله ی یادگیری مدل نزدیک تر می شویم. لازم به ذکر است در صورتی که بخواهیم این داده ها یا هر داده ی دیگری را برای یک محیط دیگر مانند CoLab آماده کنیم، کافیت تا با استفاده از دستور tar -czf {ARCHIVE_PATH} {TRAIN_PATH} {TEST_PATH} که در آن پارامتر اول مسیر ذخیره شدن فایل فشرده شده، پارامتر دوم مسیر دایرکتوری داده های train و پارامتر سوم مسیر دایرکتوری داده های test میباشد، این کار را انجام داده و سپس این فایل فشرده را در CoLab آپلود کنیم.

چالش ها و پیش پردازش

چالش اصلی مجموعه داده در وضوح تصویر آن به دلیل محدودیت های وب کم است. این تغییرات شرایط دنیای واقعی را منعکس می کنند و به کار تشخیص اشیا پیچیدگی می بخشند. با وجود این، مجموعه داده یک فرصت منحصر به فرد برای ارزیابی عملکرد مدل در شرایط کمتر از ایده آل ارائه می دهد. اگر چه این امکان برای ما فراهم است تا مراحل پیش پردازش در صورت نیاز روی تصاویر یا حاشیه نویسی اعمال شود. پیش پردازش می تواند شامل تغییر اندازه تصاویر برای یکنواختی، عادی سازی مقادیر پیکسل یا سایر پیشرفت ها برای آماده سازی مجموعه داده برای آموزش باشد.

آموزش و یادگیری مدل

در این بخش به جزئیات یادگیری مدل می پردازیم. همانطور که قبلا گفته شد، ما برای سهولت کار از مدل های از پیش آموزش دیده‌ی Tensorflow2 Zoo استفاده کرده ایم تا مشکلاتی از قبیل Overfit شدن مدل و امثال آن ها در کار ما اختلال ایجاد نکند. هدف ما این بود که برای یادگیری مدل بتوانیم علاوه بر توان پردازشی CPU از GPU هم کمک بگیریم. برای این کار نیاز به نصب و راه اندازی یک سری ابزار هایی مانند CUDA و cuDNN بود که در رابطه با آن ها شرح خواهیم داد. از آن جایی که امکان هماهنگ سازی CUDA با Tensorflow بر روی ماشین ویندوز تنها تا نسخه ی 2.10.0 امکان پذیر بوده و نسخه های جدید تر از این قابلیت پشتیبانی نخواهند کرد، لذا در این پروژه ما نیز از همان نسخه ی 2.10.0 استفاده کردیم. این نسخه برای راه اندازی نیاز به یکسری ابزار هایی دارد که عبارتند از:

- Python version 3.7-3.10
- Compiler MSVC 2019
- Bazel Build tools 5.1.1
- cuDNN 8.1
- CUDA 11.2

برای نصب پایتون به سایت رسمی python.org رجوع کرده و اقدام به نصب آن کردیم.

برای نصب کامپایلر MSVC 2019 تنها کافیست تا Visual Studio 2019 را بر روی سیستم نصب کرده و با انتخاب نصب زبان C++ این ابزار به صورت خودکار بر روی سیستم ما نصب خواهد شد. یک راه دیگر برای نصب این است که این کامپایلر را بصورت مستقیم از سایت رسمی Microsoft دانلود کرده به صورت دستی اقدام به نصب آن کنیم و سپس نیاز است تا تنظیمات لازم مسیر دهی را در قسمت System Environment Variables اعمال کنیم.

برای نصب C++ Build tools اگر از قبل Visual Studio بر روی سیستم ما نصب باشد نیاز به انجام کار اضافه ای نیست. در غیر این صورت، مثل قبل، میتوان به صورت مستقیم این ابزار را از سایت رسمی Microsoft دانلود و نصب کرد.

برای نصب CUDA و cuDNN ابتدا نیاز است اقدام به دانلود و نصب CUDA کنیم. از سایت رسمی NVIDIA میتوان اقدام به دانلود نسخه ی 11.2 این ابزار کرد. پس از نصب، نوبت به دانلود cuDNN میرسد که نسخه ی 8.1 این ابزار را هم میتوان از سایت رسمی NVIDIA دانلود کرد. پس از دانلود بایستی محتوی فایل فشرده ی دانلود شده را در مسیر نصب CUDA که بصورت پیش فرض مسیر C:\Program Files\NVIDIA GPU Computing Toolkit\v11.2 میباشد، کپی کنیم. پس از این نیاز است تا ۳ دایرکتوری به عنوان ۳ مسیر مجزا در قسمت System Environment Variables بسازیم. این ۳ مسیر به این صورت میباشند:

C:\Program Files\NVIDIA GPU Computing Toolkit\v11.2\bin

C:\Program Files\NVIDIA GPU Computing Toolkit\v11.2\libnvvp

C:\Program Files\NVIDIA GPU Computing Toolkit\v11.2\extras\CUPTI\lib64

سپس نیاز است تا با استفاده از دستور `pip install tensorflow-gpu` کتابخانه ی مورد نظر برای بهره وری از پردازش GPU را نصب کنیم. حال CPU و GPU سیستم ما برای پردازش یادگیری مدل ما آماده هستند.

چرا باید از CUDA و cuDNN استفاده کنیم؟

CUDA یک پلت فرم محاسباتی موازی است که امکان استفاده از GPU ها را برای محاسبات موازی با سرعت بالا فراهم می کند، در حالی که cuDNN یک کتابخانه با شتاب GPU است که اجرای بهینه عملیات یادگیری عمیق کلیدی را برای آموزش کارآمد شبکه عصبی و استنتاج فراهم می کند. CUDA و cuDNN کتابخانه های نرم افزاری هستند که توسط NVIDIA برای تسريع وظایف یادگیری عمیق، از جمله آموزش مدل های یادگیری ماشین، توسعه یافته اند. هنگامی که در کنار TensorFlow (یا سایر چارچوب های یادگیری عمیق) استفاده می شوند، چندین هدف را دنبال می کنند:

شتاب پردازنده گرافیکی

CUDA به ما این امکان را می دهد تا از قدرت پردازنده های گرافیکی NVIDIA (واحد پردازش گرافیکی) استفاده کنیم تا محاسبات مورد نیاز برای آموزش شبکه های عصبی عمیق را به میزان قابل توجهی افزایش دهید. GPU ها پردازنده های بسیار موازی هستند و برای عملیات ماتریسی که زیربنای آموزش شبکه های عصبی هستند، مناسب هستند. این باعث می شود زمان آموزش بسیار سریعتر در مقایسه با استفاده از CPU باشد.

بهینه سازی های cuDNN

cuDNN (CUDA Deep Neural Network library) یا همان کتابخانه شبکه عصبی عمیق CUDA یک کتابخانه با شتاب GPU است که به طور خاص برای عملیات شبکه عصبی عمیق طراحی شده است. این پیاده سازی بسیار بهینه از عملیات شبکه عصبی رایج مانند کانولوشن، ادغام و نرمال سازی را ارائه می دهد. این کار با بهره گیری از بهینه سازی های مخصوص GPU، عملکرد آموزشی را بیشتر بهبود می بخشد.

سازگاری

TensorFlow، همراه با سایر چارچوب های یادگیری عمیق مانند PyTorch، می تواند به طور یکپارچه با CUDA و cuDNN ادغام شود. این بدان معنی است که می توانیم کد یادگیری عمیق خود را با استفاده از این چارچوب ها بنویسیم و از شتاب GPU بدون نیاز به نوشتن کدهای سطح پایین مخصوص GPU بهره مند شویم.

مقیاس پذیری

CUDA و cuDNN ما را قادر می سازند تا حجم کار یادگیری عمیق خود را به مدل ها و مجموعه های داده بزرگتر تقسیم کنیم. آموزش شبکه های عصبی بزرگ بر روی CPU ها می تواند بسیار کند باشد، در حالی که GPU های مجهز به CUDA و cuDNN می توانند این وظایف را به طور موثر انجام دهند.

یادگیری عمیق زیرمجموعه ای از یادگیری ماشین است که شامل آموزش شبکه های عصبی عمیق با چندین لایه است (از این رو اصطلاح "عمیق" نامیده می شود). این شبکه ها می توانند الگوها و نمایش های پیچیده ای را از داده ها بیاموزند. CUDA و cuDNN به تسريع آموزش این شبکه های عصبی عمیق با استفاده از قدرت پردازنده های گرافیکی و ارائه اجرای بهینه عملیات شبکه عصبی کمک می کنند. بنابراین، روشن کردن این نکته مهم است که استفاده از CUDA و cuDNN نوع یادگیری را تغییر نمی دهد. این به سادگی عملکرد و سرعت وظایف یادگیری عمیق را بهبود می بخشد و آموزش مدل های بزرگتر و پیچیده تر بر روی سخت افزار موجود را امکان پذیرتر می کند.

CUDA و cuDNN برای ما چگونه کار میکنند؟

همانطور که گفتیم، CUDA ما را قادر می‌سازد از GPU ها برای محاسبات موازی استفاده کنیم، در حالی که cuDNN بلوک های ساختمانی بهینه سازی شده برای عملیات شبکه عصبی عمیق را فراهم می‌کند. آنها با هم، با استفاده از قابلیت های محاسباتی GPU و اجرای موثر عملیات شبکه عصبی، به طور چشمگیری سرعت آموزش مدل های یادگیری عمیق را افزایش می‌دهند. این شتاب برای آموزش شبکه های عصبی پیچیده و عمیق در مدت زمان معقول ضروری است. CUDA و cuDNN بر آموزش مدل با TensorFlow به روش های زیر تأثیر می‌گذارند:

محاسبات تسریع شده

وقتی از CUDA استفاده می‌کنیم، TensorFlow می‌تواند عملیات محاسباتی فشرده، مانند ضرب های ماتریس و کانولوشن ها را به GPU بارگذاری کند. پردازنده های گرافیکی، پردازنده های بسیار موازی هستند که می‌توانند این عملیات را بسیار سریعتر از پردازنده ها انجام دهند. این باعث می‌شود زمان آموزش مدل به طور قابل توجهی سریعتر شود.

عملیات شبکه عصبی بهینه شده

cuDNN به TensorFlow پیاده سازی های بهینه شده عملیات یادگیری عمیق کلیدی مانند لایه های کانولوشن و ادغام را ارائه می‌کند. وقتی TensorFlow از cuDNN استفاده می‌کند، از این پیاده سازی های بسیار کارآمد سود می‌برد که روند آموزش را سرعت بیشتری می‌بخشد.

مقیاس پذیری

CUDA و cuDNN، TensorFlow را قادر می‌سازند تا به مدل های بزرگتر و پیچیده تر مقیاس شوند. مدل های یادگیری عمیق به طور فزاینده ای عمیق و پیچیده می‌شوند، که به قدرت محاسباتی قابل توجهی نیاز دارد. پردازنده های گرافیکی مجهز به CUDA و cuDNN به TensorFlow این امکان را می‌دهند که این بارهای کاری سخت را به طور موثر مدیریت کند.

عملکرد بهبود یافته

ترکیب CUDA و cuDNN منجر به بهبود عملکرد کلی در طول تمرین می‌شود. مدل های یادگیری عمیق شما سریعتر همگرا می‌شوند، به این معنی که می‌توانیم معماری ها و فرآیندهای مختلف را با سرعت بیشتری آزمایش کنید. این شتاب به ویژه هنگام کار با مجموعه داده های بزرگ و معماری های پیچیده شبکه عصبی ارزشمند است.

به طور خلاصه، CUDA و cuDNN عملکرد TensorFlow را با استفاده از قدرت محاسباتی GPU ها و ارائه اجرای بهینه عملیات شبکه عصبی افزایش می‌دهند. این منجر به آموزش مدل سریعتر و کارآمدتر می‌شود و مقابله با وظایف یادگیری عمیق چالش برانگیز را امکان پذیر می‌کند.

مدلی که ما در این پروژه انتخاب کردیم مستقیماً از مخزن های گیت هاب Tensorflow انتخاب شده است.

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

در این صفحه مدل های از پیش آموزش دیده ی بسیاری موجود است که هر کدام مشخصات مربوط به خودشان را دارند، مثل سرعت پردازش هر فریم، دقت یادگیری برای هر فریم و خروجی مدل آموزش دیده که به چه صورت خواهد بود.

ما در اینجا از مدل SSD MobileNet V2 FPNLite 320x320 استفاده کردیم که سرعت پردازش هر فریم در این مدل در حدود ۲۲ میلی ثانیه، میانگین دقت یادگیری هر فریم ۲۲.۲ درصد و نتیجه و خروجی ما از این مدل به صورت Box خواهد بود.

"SSD MobileNet V2 FPNLite 320x320" یک مدل تشخیص اشیاء از پیش آموزش دیده است که کارایی MobileNet V2 را با قابلیت های تشخیص چند مقیاسی FPNLite ترکیب می کند. این برای وظایف تشخیص اشیاء در زمان واقعی طراحی شده است و به ویژه برای برنامه هایی که منابع محاسباتی محدود هستند مناسب است. می توان این مدل از پیش آموزش دیده را در کار تشخیص اشیاء خاص خود تنظیم یا مستقیماً از آن برای برنامه های بینایی رایانه ای مختلف استفاده کرد.

مدل «SSD MobileNet V2 FPNLite 320x320» یک معماری خاص برای تشخیص اشیاء است که اجزای مختلف را برای دستیابی به تشخیص کارآمد و دقیق اشیاء درون تصاویر ترکیب می کند.

معماری

ساختار مدل را می توان به چند جزء اصلی تقسیم کرد:

1. شبکه ستون فقرات یا همان (MobileNet V2) Backbone Network: شبکه ستون فقرات مسئول استخراج نقشه های

ویژگی از تصویر ورودی است. در این مورد، MobileNet V2 به عنوان ستون فقرات عمل می کند. MobileNet V2 یک شبکه عصبی کانولوشن است که برای دستگاه های تلفن همراه و جاسازی شده طراحی شده است. از پیچیدگی های قابل تفکیک عمیق برای کاهش محاسبات و در عین حال حفظ دقت مدل استفاده می کند.

2. شبکه هرمی ویژگی (FPNLite): (Feature Pyramid Network) FPNLite برای ایجاد یک هرم ویژگی از نقشه های

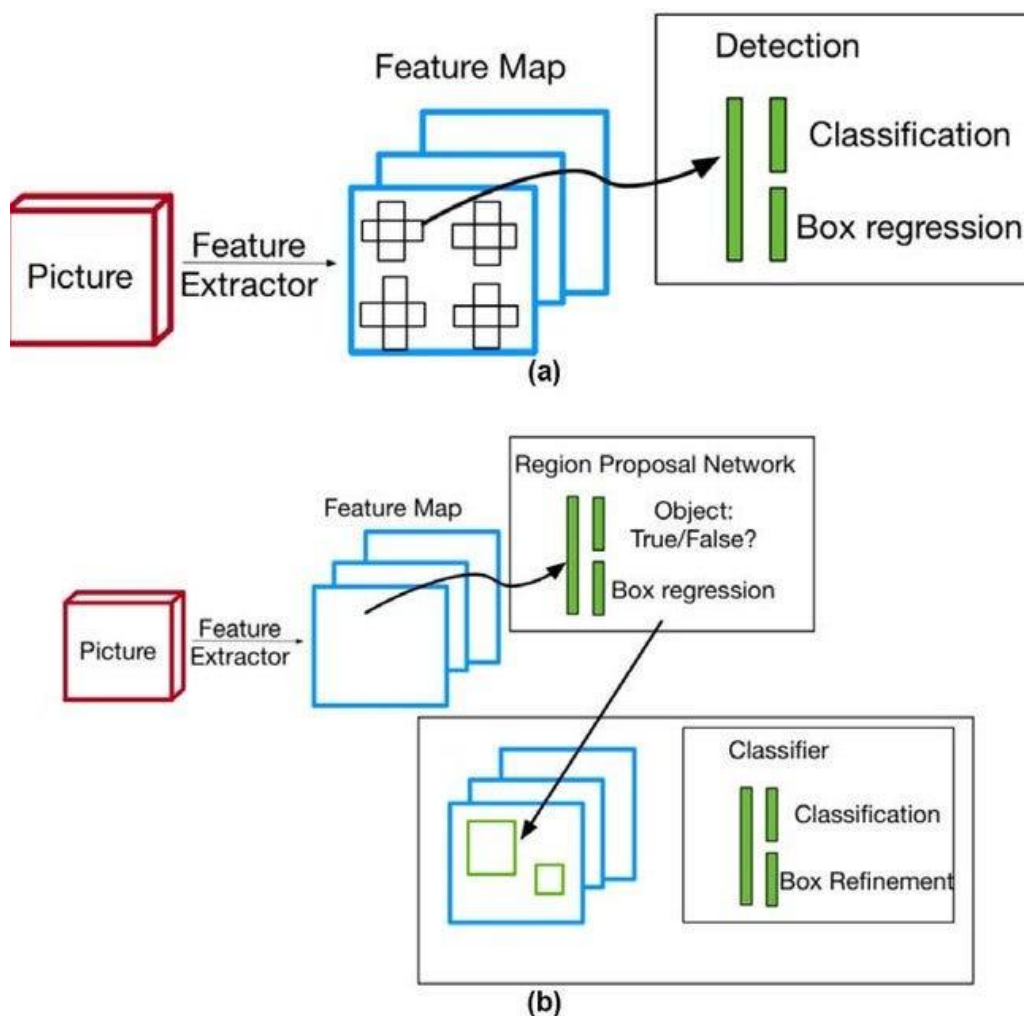
ویژگی استخراج شده توسط شبکه ستون فقرات استفاده می شود. هرم ویژگی مجموعه ای از نقشه های ویژگی در مقیاس های چندقابله است که به تشخیص اجسام در اندازه های مختلف کمک می کند. این توانایی مدل را برای تشخیص اجسام کوچک و بزرگ در یک تصویر افزایش می دهد.

3. SSD Head: در بالای هرم ویژگی، یک سر SSD (Single Shot Detection) وصل شده است. سر SSD از چندین لایه

کانولوشن تشکیل شده است که پیش بینی هایی را برای کلاس های شی و مختصات جعبه مرزی در وضوح های فضایی (مقیاس های) مختلف ایجاد می کند. این به مدل اجازه می دهد تا اشیاء را در مقیاس ها و نسبت های مختلف در تصویر ورودی تشخیص دهد.

4. جعبه های لنگر: برای تسهیل تشخیص اشیاء، مجموعه ای از جعبه های لنگر (همچنین به عنوان جعبه های قبلی شناخته

می شوند) با هر مکان مکانی در نقشه های ویژگی مرتبط است. این جعبه های لنگر دارای اندازه ها و نسبت های از پیش تعریف شده هستند و برای پیش بینی جعبه های محدودکننده شی و احتمالات کلاس مرتبط با آنها استفاده می شوند.



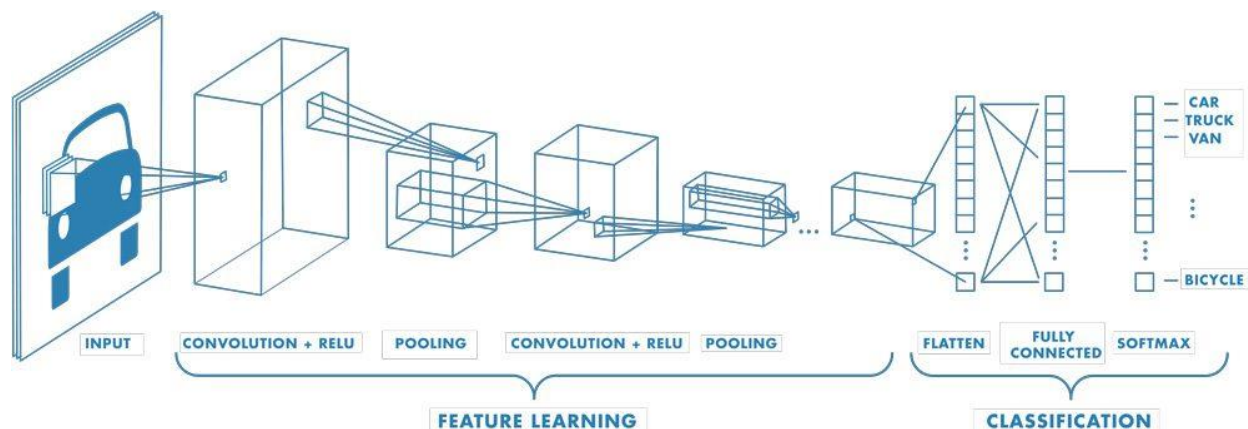
چگونه کار میکند؟

این مدل ابتدا با پردازش تصویر ورودی از طریق شبکه اصلی (MobileNet V2) برای استخراج نقشه های ویژگی کار می کند. سپس این نقشه های ویژگی از طریق FPN Lite برای ایجاد یک هرم ویژگی منتقل می شوند. سپس سر SSD روی هرم ویژگی عمل می کند تا جعبه های محدود و احتمالات کلاس را برای اشیاء در مقیاس ها و مکان های مختلف درون تصویر پیش بینی کند.

برای هر جعبه لنگر، این مدل موارد زیر را پیش بینی میکند:

- **احتمالات کلاس:** احتمال اینکه جعبه لنگر حاوی یک شی از یک کلاس خاص باشد.
- **مرزبندی و جداسازی offset های پاکس:** تنظیم ابعاد و موقعیت جعبه لنگر برای مطابقت با مکان و اندازه واقعی شی.

سپس این پیش بینی ها پس از پردازش می شوند تا تشخیص های کم اعتماد را فیلتر کنند و برای به دست آوردن مجموعه نهایی تشخیص های شی، سرکوب غیر حداکثری را انجام دهند.



مثالی از یک شبکه متشکل از چندین لایه کانولوشن. فیلترها برای هر تصویر آموزشی با وضوح های مختلف اعمال می شوند و خروجی هر تصویر پیچیده به عنوان ورودی لایه بعدی استفاده می شود (منبع Mathworks)

مقابله با Overfitting و Underfitting

برای کاهش Overfitting، تکنیک های رایجی مانند افزایش داده ها، dropout و تنظیم وزن ممکن است در طول تمرین استفاده شود. تقویت داده ها شامل اعمال تبدیل های تصادفی به داده های آموزشی است که به طور موثر تنوع مجموعه داده را افزایش می دهد و به تعمیم بهتر مدل کمک می کند. Dropout یک تکنیک منظم سازی است که از اتکای بیش از حد مدل به هر ویژگی یا نورون منفرد جلوگیری می کند. تکنیک های تنظیم وزن های بزرگ را در مدل جرمیمه می کنند و آن را از تناسب نویز در داده ها منصرف می کنند.

علاوه بر این، انتخاب دقیق فرامترها، مانند نرخ یادگیری و اندازه کلاستر ها، برای آموزش یک مدل به خوبی تعمیم یافته و جلوگیری از Overfitting ضروری است.

1. افزایش داده ها: افزایش داده ها شامل اعمال تبدیل های تصادفی به داده های آموزشی برای ایجاد نمونه های آموزشی اضافی است. برای کارهای تشخیص اشیا مانند آنچه توسط این مدل انجام می شود، تکنیک های تقویت داده ها ممکن است شامل چرخش های تصادفی، ترجمه، مقیاس گذاری، چرخش و تغییر در روشنایی یا کنتراست باشد. افزایش داده ها تنوع داده های آموزشی را افزایش می دهد و به تعمیم بهتر مدل کمک می کند و Overfitting را کاهش می دهد.

2. Dropout: Dropout یک تکنیک منظم سازی است که شامل صفر کردن تصادفی کسری از نورون ها در طول تمرین است. این از وابستگی بیش از حد مدل به هر ویژگی یا نورون خاصی جلوگیری می کند و Overfitting را کاهش می دهد. در حالی که dropout معمولاً در لایه های کاملاً متصل استفاده می شود، ممکن است برای لایه های کانولوشنی خاصی نیز اعمال شود.

3. تنظیم وزن: روش های تنظیم وزن، مانند تنظیم L1 و L2، شرایط جرمیمه ای را به تابع loss اضافه می کنند که مقادیر وزن کمتری را تشویق می کند. این امر مدل را از تطبیق نویز در داده ها منع می کند و می تواند به جلوگیری از Overfitting کمک کند. در مدل «SSD MobileNet V2 FPNLite 320x320»، تنظیم وزن ممکن است برای لایه ها یا اجزای خاص مدل اعمال شود.

4. توقف زودهنگام: آموزش معمولاً با استفاده از یک مجموعه داده اعتبارسنجی نظارت می شود و زمانی که عملکرد مدل در مجموعه داده اعتبارسنجی شروع به کاهش کرد، آموزش می تواند زودتر متوقف شود. این به جلوگیری از ادامه یادگیری مدل از نویز در داده های آموزشی کمک می کند.

5. **نرمال سازی دسته ای:** نرمال سازی دسته ای تکنیکی است که فعال سازی ها را در یک لایه در طول آموزش عادی می کند. این می تواند با جلوگیری از بزرگ یا خیلی کوچک شدن فعالیت ها، به تثبیت و تسریع تمرین و کاهش Overfitting کمک کند.

6. **زمان بندی نرخ یادگیری:** زمان بندی نرخ یادگیری شامل تنظیم نرخ یادگیری در طول آموزش است. این می تواند به همگرایی بیشتر مدل کمک کرده و از Overshoot شدن یک راه حل بهینه، که به خودی خود میتواند منجر به Overfitting گردد، جلوگیری کند.

7. **Cross Validation:** در صورت امکان، این تکنیک می تواند به ارزیابی عملکرد مدل در زیر مجموعه های متعدد داده ها کمک کند و بینش هایی را در مورد مسائل مربوط به overfitting و underfitting ارائه دهد.

8. **Transfer Learning:** مدل های از پیش آموزش دیده، مانند "SSD MobileNet V2 FPNLite 320x320" اغلب از Transfer Learning سود می برند. می توان با وزن هایی که روی یک مجموعه داده بزرگ از قبل آموزش داده شده اند شروع کرد و مدل را روی مجموعه داده های خاص خود تنظیم کرد. Transfer Learning از دانش به دست آمده از مدل از پیش آموزش دیده استفاده می کند و Overfitting را کاهش می دهد، به ویژه زمانی که مجموعه داده ی هدف، کوچک است.

ترکیب خاص این تکنیک ها و پارامترهای آنها به مجموعه داده، معماری مدل و فرآیند آموزش بستگی دارد. پرداختن به بیش برآزش فرآیندی تکراری است که اغلب شامل آزمایش استراتژی های مختلف و نظارت بر عملکرد مدل بر روی داده های اعتبارسنجی برای دستیابی به بهترین مبادله بین سوگیری و واریانس است.

سایر ویژگی های مدل

1. **سرعت و کارایی:** یکی از ویژگی های کلیدی این مدل سرعت و کارایی آن است. این برای انجام سریع وظایف تشخیص اشیاء طراحی شده است و آن را برای برنامه های زمان واقعی یا تقریباً واقعی مناسب می کند. استفاده از MobileNet V2 به عنوان شبکه اصلی به کارایی آن کمک می کند. معماری های MobileNet به دلیل نیازهای کم محاسباتی خود در عین حفظ دقت معقول شناخته شده اند.

2. **تشخیص اشیاء:** این مدل به طور خاص برای تشخیص اشیاء طراحی شده است، به این معنی که می تواند اشیاء را در تصاویر مکان یابی و طبقه بندی کند. این می تواند چندین شی از کلاس های مختلف را در یک تصویر شناسایی کند، مختصات جعبه مرزی و احتمالات کلاس را برای هر شی شناسایی شده ارائه دهد.

3. **مقیاس مستقل:** به لطف شبکه هر می ویژگی (FPNLite)، این مدل مقیاس ناپذیر است. این می تواند اشیاء در اندازه ها و مقیاس های مختلف را در یک تصویر تشخیص دهد. این برای دست زدن به اشیاء با اندازه های مختلف در سناریوهای مختلف دنیای واقعی بسیار مهم است.

4. **وزن های از پیش آموزش دیده:** به طور معمول، این مدل با وزن های از پیش آموزش دیده روی یک مجموعه داده بزرگ، مانند COCO (Common Objects in Context) ارائه می شود. این بدان معنی است که یاد گرفته است طیف گسترده ای از اشیاء را تشخیص دهد و می تواند برای کارهای تشخیص اشیاء خاص با مقادیر نسبتاً کمی از داده های برچسب گذاری شده به خوبی تنظیم شود.

5. **اندازه و ابعاد ورودی:** این مدل به گونه ای پیکربندی شده است که تصاویری با اندازه 320x320 پیکسل را به عنوان ورودی در طول آموزش و استنتاج بپذیرد. انتخاب اندازه ورودی را می توان بسته به میزان تطابق بین سرعت و دقت مورد نیاز برنامه شما تنظیم کرد.

6. **(NMS) Non-Maximum Suppression:** پس از ایجاد پیش‌بینی‌ها برای باکس‌های مشخص کننده ی شی، مدل معمولاً NMS را به عنوان یک مرحله پس از پردازش اعمال می‌کند. NMS برای حذف تشخیص‌های تکراری یا با همپوشانی زیاد استفاده می‌شود، و تضمین می‌کند که فقط مطمئن‌ترین و غیرضروری‌ترین تشخیص‌ها حفظ می‌شوند.

7. **انعطاف پذیری:** در حالی که مدل‌های از پیش آموزش‌دیده برای بسیاری از وظایف ارزشمند هستند، می‌توان آن‌ها را نیز در مجموعه داده‌های خاص دامنه تنظیم کرد. معماری این مدل به ما این امکان را می‌دهد تا با آموزش آن بر روی مجموعه داده‌های خود، آن را با نیازهای تشخیص شی خاص خود تطبیق دهیم.

آموزش دادن مدل

در این بخش به ترتیب به مقدمات آموزش مدل، فرایند آموزش مدل و در آخر، خروجی و نتیجه ی کار مدل میپردازیم. همچنین بعد از اتمام یادگیری مدل، آن را Evaluate کرده و با Tensorflow-board به بررسی عملکرد مدل در طی فرایند یادگیری میپردازیم.

```
CUSTOM_MODEL_NAME = 'my_ssd_mobnet_forUni'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
```

- **CUSTOM_MODEL_NAME:** این متغیر ثابت نشان دهنده نامی است که می خواهیم به مدل تشخیص شی سفارشی خود بدهیم. همانطور که قبلاً به آن اشاره شد، ما یک مدل از پیش آموزش دیده را برای کار خاص خود تنظیم و آن را بر اساس آن نام گذاری کردیم.

- **PRETRAINED_MODEL_NAME:** این متغیر ثابت نشان دهنده نام یک مدل تشخیص شی از قبل آموزش دیده است که قصد داریم از آن به عنوان نقطه شروع برای پروژه خود استفاده کنیم. مدل مشخص شده در اینجا "ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8" است. این مدل بخشی از Tensorflow Zoo Pre-trained Models بوده و بر اساس معماری MobileNet V2 برای تشخیص اشیا است.

- **PRETRAINED_MODEL_URL:** این متغیر ثابت حاوی URL برای دانلود فایل مدل از پیش آموزش دیده است. URL ارائه شده به مدل از پیش آموزش دیده MobileNet V2 با پیکربندی و وزن خاص اشاره می کند.

- **TF_RECORD_SCRIPT_NAME:** این متغیر نام یک فایل اسکریپت پایتون را مشخص می کند که برای تولید فایل های TFRecord استفاده می شود. TFRecord یک فرمت متداول برای ذخیره سازی داده ها است که برای آموزش مدل های یادگیری ماشین از جمله مدل های تشخیص اشیا استفاده می شود. این اسکریپت مسئول تبدیل مجموعه داده ما به قالبی است که می تواند به طور موثر برای آموزش استفاده شود.

- **LABEL_MAP_NAME**: این ثابت نام یک فایل نقشه برچسب (معمولاً با فرمت Protobuf، .pbtxt) را مشخص می کند که حاوی نگاشت بین برچسب های کلاس و شناسه های عدد صحیح مربوط به آنها است. برای برچسب گذاری اشیاء شناسایی شده توسط مدل استفاده می شود.

```
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
```

این قطعه کد تضمین می کند که کدهای مخصوص به تشخیص اشیاء لازم از مخزن مدل های TensorFlow در مسیر مشخص شده وجود دارد. اگر موجود نباشد، مخزن را شبیه سازی می کند تا آن را برای پروژه شما در دسترس قرار دهد. این یک روش معمول برای به دست آوردن آخرین کد و منابع تشخیص اشیاء هنگام راه اندازی یک پروژه تشخیص شی با استفاده از TensorFlow است. اگر دایرکتوری «research/object_detection» در «Tensorflow/models» یافت نشود، کد مخزن مدل های TensorFlow را از GitHub در فهرست «Tensorflow/models» کلون می کند تا اطمینان حاصل شود که کد مربوط به تشخیص شی برای پروژه ما در دسترس است. این یک روش معمول برای تنظیم کد و منابع لازم برای API تشخیص اشیاء TensorFlow است.

```
if os.name == 'posix':
    !wget {PRETRAINED_MODEL_URL}
    !mv {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
if os.name == 'nt':
    wget.download(PRETRAINED_MODEL_URL)
    !move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
```

در این قطعه کد، از این مسیر برای دانلود و استخراج فایل های مدل از پیش آموزش دیده استفاده می کند. به طور خاص، نقطه بازرسی مدل دانلود شده (فایل tar.gz) را به این دایرکتوری منتقل می کند و سپس محتویات بایگانی را در فهرست "Tensorflow/workspace/pre-trained-models" استخراج می کند.

این یک روش معمول هنگام سازماندهی یک پروژه TensorFlow است، زیرا فایل های مدل از پیش آموزش دیده را جدا از سایر فایل ها و منابع پروژه برای مدیریت و دسترسی آسان تر نگه می دارد.

```
labels = [{ 'name': 'ThumbsUp', 'id': 1}, { 'name': 'ThumbsDown', 'id': 2}, { 'name': 'ThankYou', 'id': 3}, { 'name': 'LiveLong', 'id': 4},
           { 'name': 'FingerUp', 'id': 5}, { 'name': 'FingerDown', 'id': 6}, { 'name': 'Stop', 'id': 7}, { 'name': 'RockOn', 'id': 8},
           { 'name': 'Victory', 'id': 9}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname: {}\n'.format(label['name']))
        f.write('\tid: {}\n'.format(label['id']))
        f.write('\n')
```

labels در واقع یک لیستی از دیکشنری هایی است که در آن هر دیکشنری نشان دهنده برچسبی برای تشخیص شی است. هر دیکشنری شامل دو جفت کلید-مقدار است:

- **name**: نام برچسب یا لیبل
- **id**: یک شناسه عدد صحیح که به برچسب اختصاص داده شده است. این شناسه ها برای شناسایی و ارجاع منحصر به فرد برچسب ها در مدل تشخیص شی استفاده می شوند.

این قطعه کد برای ایجاد یک فایل نقشه برچسب در قالب متن Protobuf استفاده می شود که نام برچسب های قابل خواندن توسط انسان را با شناسه های عدد صحیح منحصر به فرد مرتبط می کند. این نقشه برچسب معمولاً در طول آموزش و ارزیابی مدل های تشخیص اشیا برای نگاشت برچسب های کلاس به شناسه های عددی مربوطه استفاده می شود. این یک جزء ضروری برای پیکربندی وظایف تشخیص اشیا و اطمینان از اینکه مدل می تواند اشیا را بر اساس این برچسب ها شناسایی و طبقه بندی کند، است.

```
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}
```

این کد یک اسکریپت پایتون را اجرا می کند که مجموعه داده های تصویری train و test ما را به فرمت TFRecord تبدیل می کند، که یک فرمت داده رایج است که برای آموزش مدل های یادگیری ماشین با TensorFlow استفاده می شود. فایل های TFRecord حاوی داده های تصویری به همراه برچسب های مرتبط تعریف شده در نقشه برچسب هستند که آن را برای آموزش و ارزیابی مدل های تشخیص شی مناسب می سازد.

```
TRAINING_SCRIPT = os.path.join(paths['API_MODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
```

```
command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=2000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'], files['PIPELINE_CONFIG'])
```

متغیر **command** اکنون حاوی یک دستور کامل برای اجرای فرآیند آموزش مدل تشخیص شی ما است. هنگامی که این دستور را اجرا می کنیم، اسکریپت آموزشی مشخص شده (**model_main_tf2.py**) را با کانفیگ ها و تنظیمات ارائه شده فراخوانی و آموزش مدل ما را آغاز می کند.

این اسکریپت چک پوینت های مدل آموزش دیده (از جمله وزن ها و پارامترهای مدل) را در دایرکتوری مشخص شده توسط CHECKPOINT_PATH ذخیره می کند. این دایرکتوری در مسیر «Tensorflow/workspace/models» قرار خواهد گرفت و چک پوینت ها با مدل سفارشی به نام «my_ssd_mobnet_forUni» مرتبط خواهند شد.

حال میتوان یا به صورت مستقیم دستور مربوط به شروع آموزش مدل که در متغیر command ریخته شده است را از داخل Jupyter Notebook با اجرای دستور {command}! اجرا کرد، یا اینکه با استفاده از print(command) محتویات داخل این متغیر را چاپ کرد و به صورت دستی در ترمینال اجرا کرد که روش دوم پیشنهاد میشود. به دلیل این که همزمان میتوان خروجی هایی را مشاهده کرد و از روند فرایند یادگیری مطلع شد. در این جا ما نیز برای نمایش خروجی از روش دوم استفاده کرده و دستور CMD اجرا میکنیم.

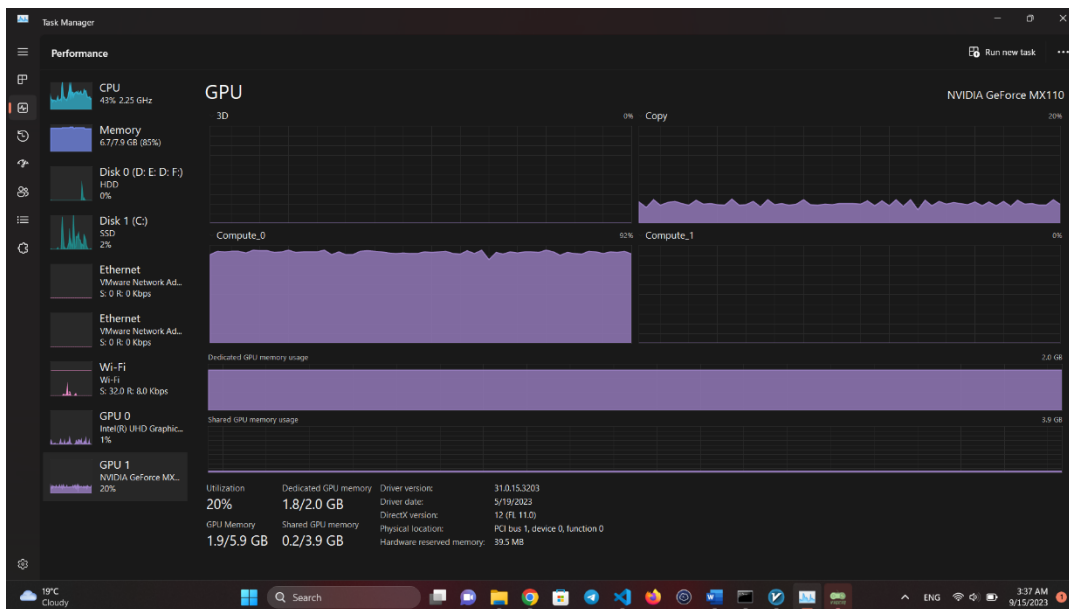
```
print(command)

python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet_forUni --pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet_forUni\pipeline.config --num_train_steps=2000
```

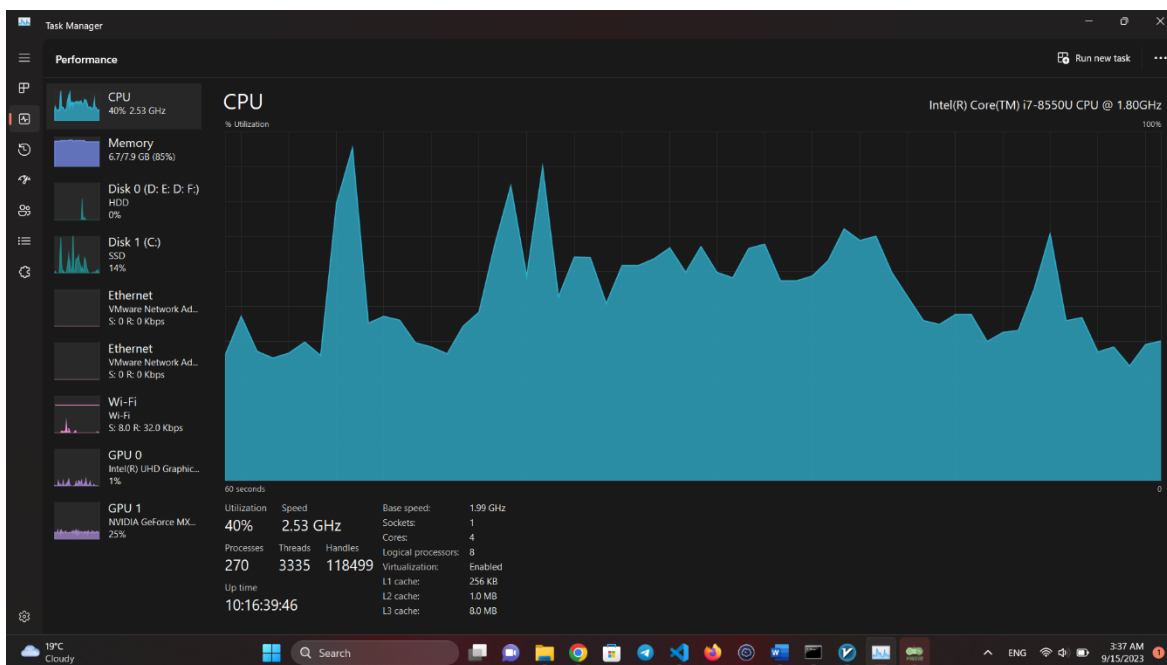
در این جا ما مشخص کردیم که تعداد مراحل یادگیری ۲۰۰۰ باشد.

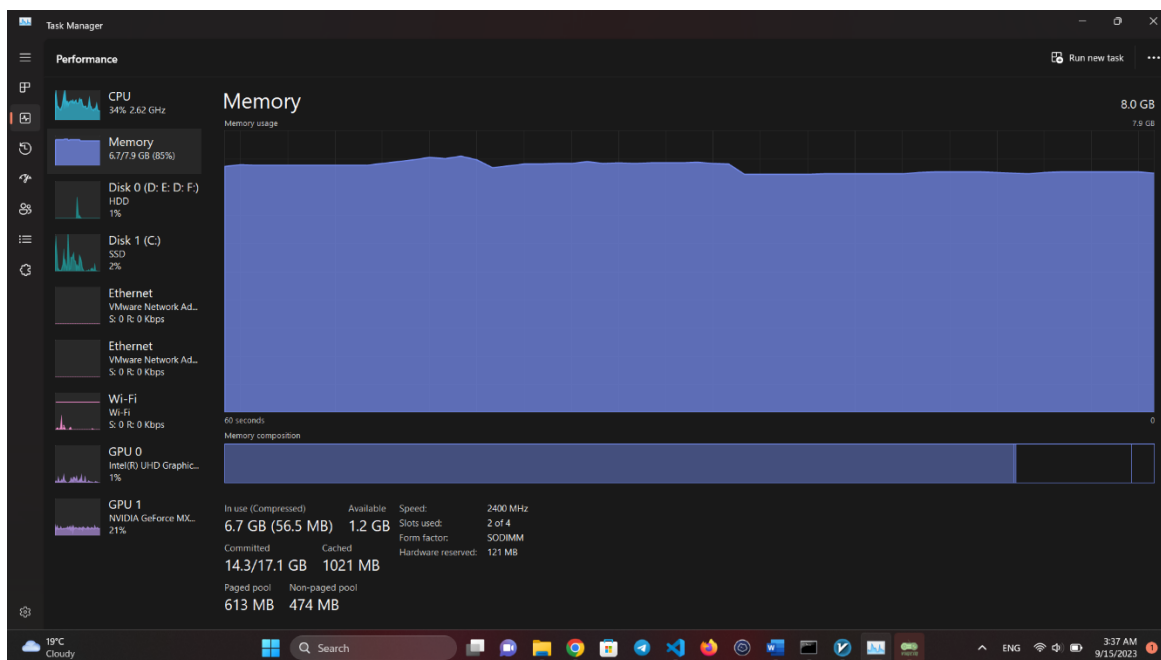
```
Administrator: Command Prompt - python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet_forUni --pipeline_config_path=Tensorflow\workspace\models\my_ssd_mobnet_forUni\pipeline.config --num_train_steps=2000
INFO:tensorflow:Step 100 per-step time 0.990s
[0915 03:35:55.384510 17332 model_lib_v2.py:705] Step 100 per-step time 0.990s
INFO:tensorflow:{"loss/classification_loss": 0.4484636,
"loss/localization_loss": 0.35581604,
"loss/regularization_loss": 0.15427555,
"loss/total_loss": 0.9505552,
"learning_rate": 0.0319994}
[0915 03:35:55.384510 17332 model_lib_v2.py:708] {"loss/classification_loss": 0.4484636,
"loss/localization_loss": 0.35581604,
"loss/regularization_loss": 0.15427555,
"loss/total_loss": 0.9505552,
"learning_rate": 0.0319994}
INFO:tensorflow:Step 200 per-step time 0.626s
[0915 03:36:57.875126 17332 model_lib_v2.py:705] Step 200 per-step time 0.626s
INFO:tensorflow:{"loss/classification_loss": 0.4746014,
"loss/localization_loss": 0.2446419,
"loss/regularization_loss": 0.1545817,
"loss/total_loss": 0.87382495,
"learning_rate": 0.0373328}
[0915 03:36:57.875126 17332 model_lib_v2.py:708] {"loss/classification_loss": 0.4746014,
"loss/localization_loss": 0.2446419,
"loss/regularization_loss": 0.1545817,
"loss/total_loss": 0.87382495,
"learning_rate": 0.0373328}
INFO:tensorflow:Step 300 per-step time 0.628s
[0915 03:38:00.642450 17332 model_lib_v2.py:705] Step 300 per-step time 0.628s
INFO:tensorflow:{"loss/classification_loss": 0.2890763,
"loss/localization_loss": 0.1871256,
"loss/regularization_loss": 0.15471609,
"loss/total_loss": 0.63091797,
"learning_rate": 0.0426662}
[0915 03:38:00.642450 17332 model_lib_v2.py:708] {"loss/classification_loss": 0.2890763,
"loss/localization_loss": 0.1871256,
"loss/regularization_loss": 0.15471609,
"loss/total_loss": 0.63091797,
"learning_rate": 0.0426662}
INFO:tensorflow:Step 400 per-step time 0.604s
[0915 03:39:01.022640 17332 model_lib_v2.py:705] Step 400 per-step time 0.604s
INFO:tensorflow:{"loss/classification_loss": 0.3113395,
"loss/localization_loss": 0.16904348,
"loss/regularization_loss": 0.15475799,
"loss/total_loss": 0.6351354,
"learning_rate": 0.047999598}
[0915 03:39:01.022640 17332 model_lib_v2.py:708] {"loss/classification_loss": 0.3113395,
"loss/localization_loss": 0.16904348,
"loss/regularization_loss": 0.15475799,
"loss/total_loss": 0.6351354,
"learning_rate": 0.047999598}
```

این تصویر نمونه هایی از خروجی مورد نظر ما در طی فرایند یادگیری است. همانطور که مشخص است جزئیاتی مانند نرخ یادگیری، زمان یادگیری، مرحله ی یادگیری قابل مشاهده است. این خروجی بعد از هر ۱۰۰ مرحله یادگیری بروزرسانی میشود. انتظار میرود جزئیاتی مانند نرخ یادگیری و میزان loss در هر محله بهبود یابد.



در شکل بالا میزان درگیری و پردازش GPU قابل مشاهده است. واضح است که هرچه GPU از قدرت پردازش بیشتری برخوردار باشد، میزان کارایی آن نیز بیشتر و بهتر خواهد بود زیرا حافظه ی بیشتری از GPU را میتوان به فرایند یادگیری اختصاص داد. این جریان برای CPU هم صدق میکند. مشخصات و میزان پردازش CPU را در شکل زیر میتوان مشاهده کرد.





چند نکته ی تکمیلی:

- ویژگی های CPU و GPU نقش حیاتی در آموزش مدل ایفا می کنند. یک GPU با ظرفیت حافظه بیشتر (VRAM) می تواند دسته های بزرگتری از داده ها را مدیریت کند که می تواند آموزش را تسریع کند. برعکس، اگر حافظه GPU برای مدل و داده های ما کافی نباشد، ممکن است با خطاهای تخصیص حافظه مواجه شویم. کاهش اندازه دسته یا انتخاب مدلی با حافظه کارآمدتر ممکن است راه حل های ضروری باشد. علاوه بر این، مدل و معماری GPU بر عملکرد تأثیر می گذارد و GPU های قدرتمندتر محاسبات را سریعتر انجام می دهند.
- CPU مسئول پیش پردازش داده ها و تغذیه داده ها به GPU است. یک CPU سریعتر می تواند زمان آماده سازی داده ها را کاهش دهد، اما ممکن است مستقیماً بر سرعت آموزش مدل تأثیر نگذارد. راه اندازی های چند GPU می توانند بارهای کاری را برای آموزش سریعتر توزیع کنند، اما نیاز به مدیریت دقیق حافظه دارند. پیچیدگی مدل و اندازه دسته ای بر استفاده از حافظه تأثیر می گذارد و نظارت بر حافظه GPU در طول آموزش برای شناسایی عملیات های فشرده منابع ضروری است. در نهایت، درک و بهینه سازی ویژگی های CPU و GPU برای آموزش مدل کارآمد و بدون خطا بسیار مهم است.
- مقدار RAM موجود در سیستم ما نیز نقش مهمی در آموزش مدل دارد. در حالی که GPU بیشتر کارهای محاسباتی سنگین را در طول آموزش انجام می دهد، RAM برای ذخیره داده ها، پارامترهای مدل و نتایج متوسط ضروری است. در مواردی که مجموعه داده بزرگ است یا مدل پیچیده است، کمبود RAM می تواند منجر به مشکلاتی از جمله آموزش کند یا خطاهای خارج از حافظه شود.
- برای رفع محدودیت های RAM، می توان استراتژی هایی مانند تولیدکننده های داده را در نظر گرفت که دسته های کوچک تری از داده ها را در یک زمان در حافظه بارگذاری می کنند و ردپای کلی حافظه را کاهش می دهند. علاوه بر این، بهینه سازی pipeline پیش پردازش داده ها و استفاده از تکنیک های کارآمد بارگذاری داده ها می تواند به مدیریت استفاده از RAM کمک کند. ایجاد تعادل بین GPU VRAM و RAM سیستم برای اطمینان از آموزش روان و کارآمد مدل بسیار مهم است. ظرفیت RAM کافی می تواند به جلوگیری از تنگناهای مرتبط با حافظه کمک کند و تجربه کلی تمرین را افزایش دهد.

ارزیابی مدل

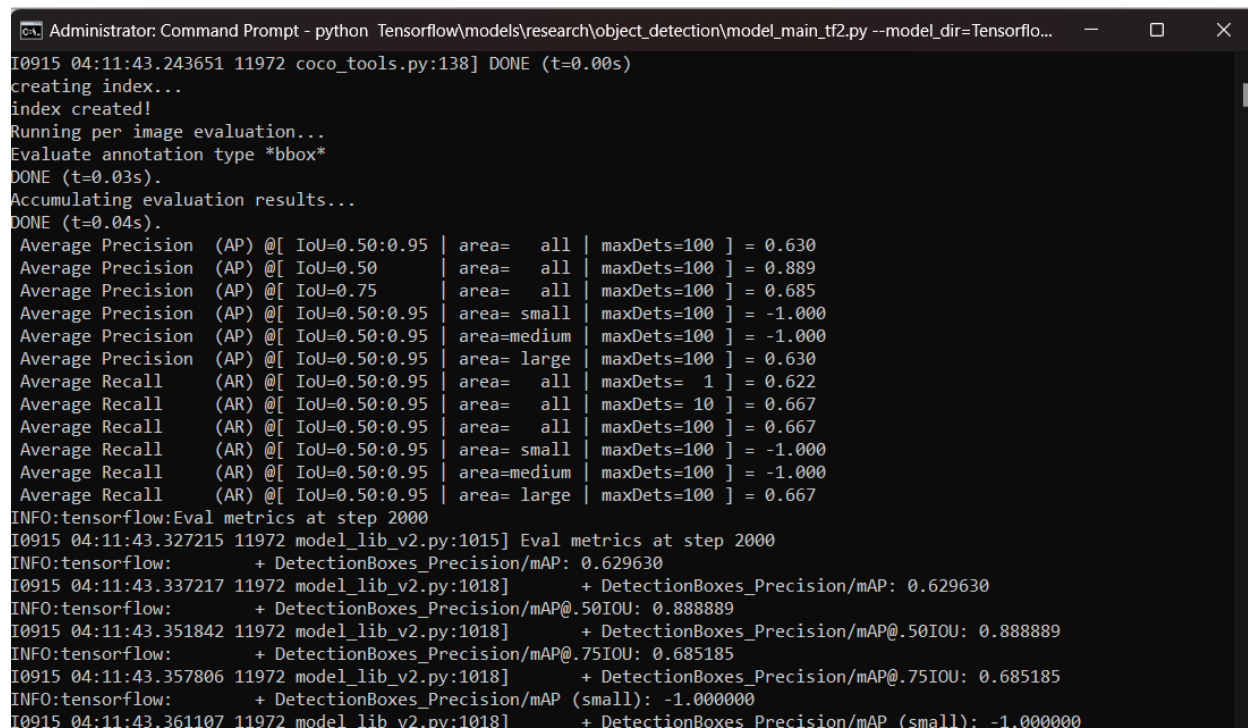
پس از اینکه فرایند آموزش مدل به پایان رسید، اکنون نوبت به ارزیابی مدل میرسد. در اینجا ما جزئیات ارزیابی مدل آموزش دیده را یکبار در ترمینال و بار دیگر با استفاده از ابزار Tensorboard به نمایش خواهیم گذاشت. علت استفاده از Tensorboard برای دیدن نمودار های مرتبط به فرایند یادگیری است تا دید بهتری نسبت به این فرایند به ما بدهد.

```
command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],
                                                                                       files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])
```

```
print(command)
```

```
python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet_forUni --pipeline_config_path=
Tensorflow\workspace\models\my_ssd_mobnet_forUni\pipeline.config --checkpoint_dir=Tensorflow\workspace\models\my_ssd_mobnet_forUni
```

این دستور برای شروع ارزیابی مدل تشخیص شی ما استفاده می شود. فایل ها و دایرکتوری های ورودی لازم را مشخص می کند، از جمله اسکرپت برای اجرا، دایرکتوری مدل برای ذخیره چک پوینت ها، فایل پیکربندی pipeline، و دایرکتوری حاوی فایل های چک پوینت. زمانی که این دستور را اجرا کنیم، فرایند ارزیابی را با استفاده از تنظیمات ارائه شده آغاز می کند.

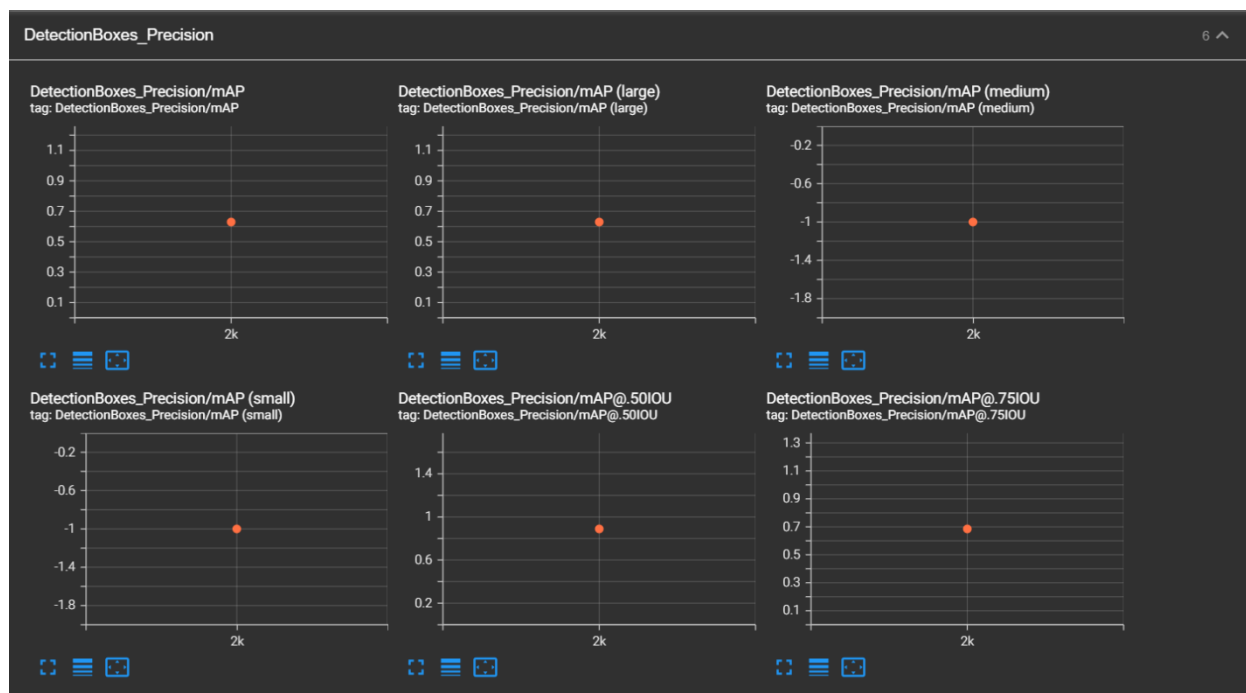


```
Administrator: Command Prompt - python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflo...
I0915 04:11:43.243651 11972 coco_tools.py:138] DONE (t=0.00s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.03s).
Accumulating evaluation results...
DONE (t=0.04s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.630
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.889
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.685
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.630
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.622
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.667
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.667
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.667
INFO:tensorflow:Eval metrics at step 2000
I0915 04:11:43.327215 11972 model_lib_v2.py:1015] Eval metrics at step 2000
INFO:tensorflow: + DetectionBoxes_Precision/mAP: 0.629630
I0915 04:11:43.337217 11972 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP: 0.629630
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.50IOU: 0.888889
I0915 04:11:43.351842 11972 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP@.50IOU: 0.888889
INFO:tensorflow: + DetectionBoxes_Precision/mAP@.75IOU: 0.685185
I0915 04:11:43.357806 11972 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP@.75IOU: 0.685185
INFO:tensorflow: + DetectionBoxes_Precision/mAP (small): -1.000000
I0915 04:11:43.361107 11972 model_lib_v2.py:1018] + DetectionBoxes_Precision/mAP (small): -1.000000
```

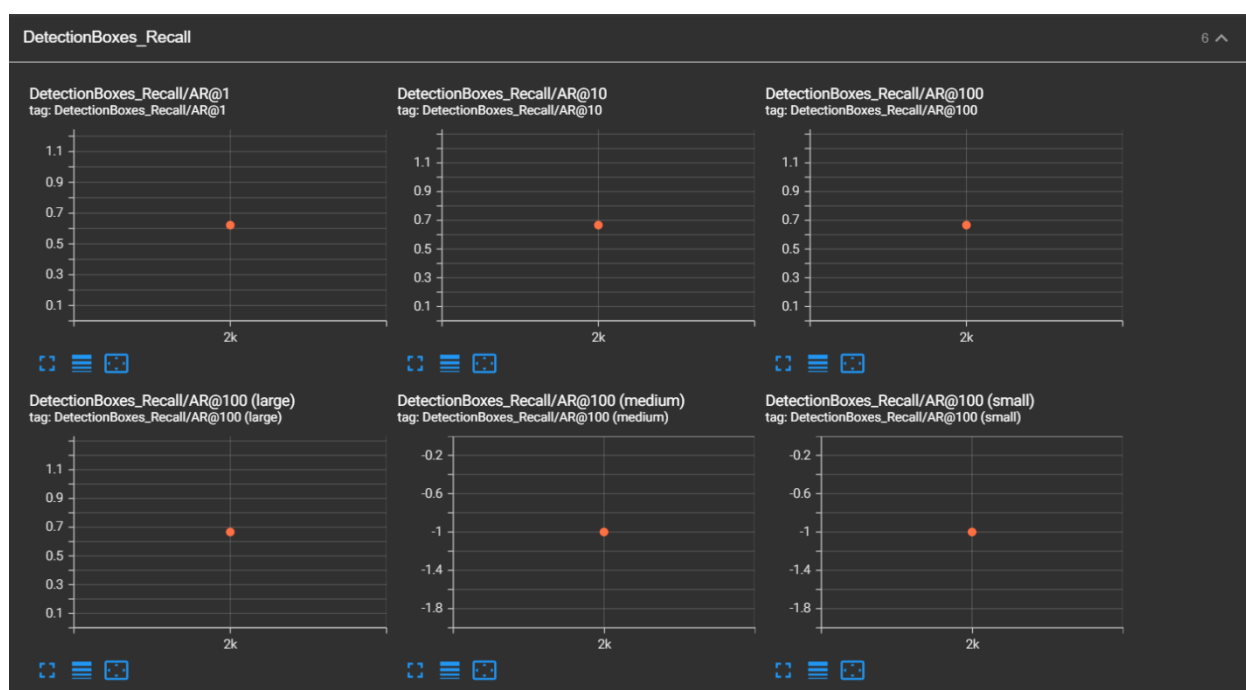
همانطور که مشاهده میشود یک آمار کلی از جزئیات فرایند یادگیری در اختیار ما قرار میدهد از جمله دقت مدل و recall (کلاس های مثبت)

حال یکبار ارزیابی مدل را با استفاده از Tensorboard نمایش میدهیم.

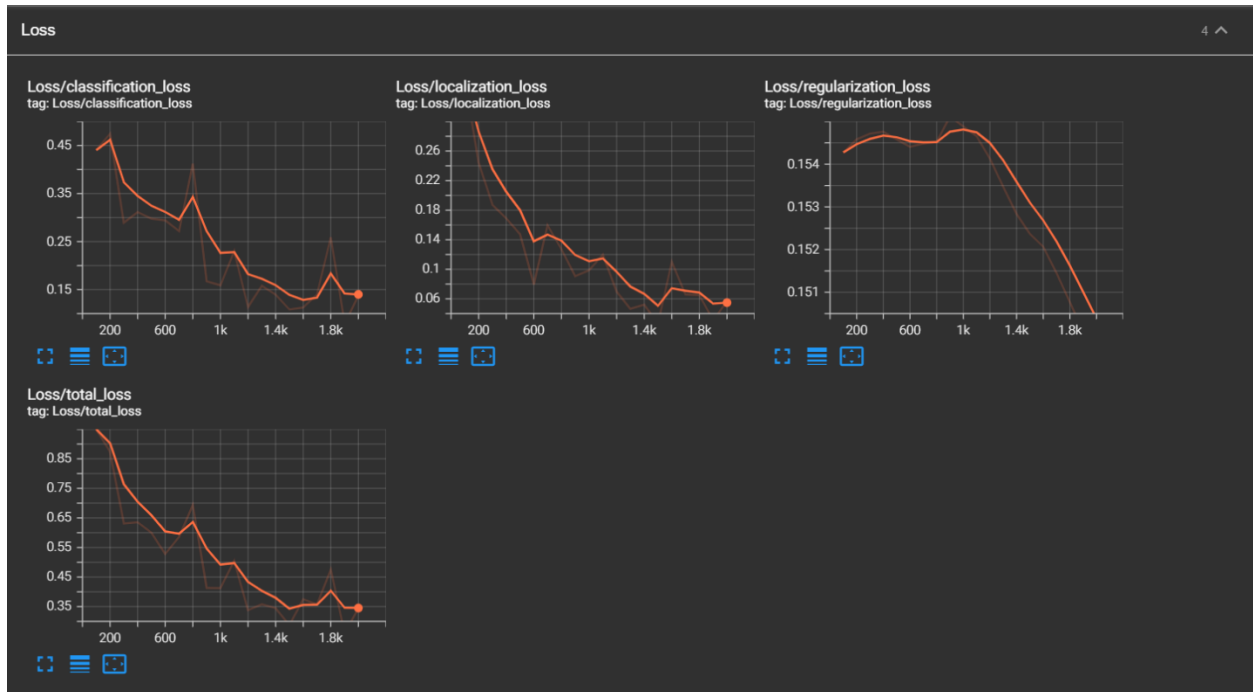
جزئیات ارزیابی نهایی مربوط به precision مدل:



جزئیات ارزیابی نهایی مربوط به recall مدل:



جزئیات آماری مربوط به میزان loss مدل در حین آموزش:



جزئیات ارزیابی نهایی مربوط به میزان loss مدل:



جزئیات آماری مربوط به نرخ یادگیری مدل در حین آموزش:



به طور خلاصه میتوان گفت که این فرایند یادگیری، در ۱۰۰ مرحله ی آخر خود:

- دقت (precision) این مدل تقریباً ۶۳ درصد
- تشخیص درست کلاس های مثبت (recall) این مدل تقریباً ۶۲ درصد
- میزان total loss این مدل بعد از ارزیابی تقریباً ۷۵ درصد
- میزان total loss این مدل در حین آموزش تقریباً ۳۴ درصد
- نرخ یادگیری این مدل در حین آموزش تقریباً ۸ درصد

با توجه به این آمار میتوان گفت این فرایند آن طور که باید نمیتواند مورد قبول واقع شود و مدل خوب و بهینه ای برای ما باشد. ما میتوانیم با اختصاص دادن داده های بیشتر، لیبیل گذاری بهتر و انتخاب یک مدل از پیش آموزش دیده ی دیگر فرایند آموزش را از سر گرفته و به آمار کلی مدل بهبود ببخشیم.

بارگذاری آخرین چک پوینت مدل آموزش دیده و تشخیص اشیاء

تشخیص اشیا میتواند به دو صورت انجام بگیرد. میتوان به صورت استاتیک آدرس یکی از داده های test را به مدل داد تا مدل نتیجه ی تشخیص اشیا را برای ما چاپ کند، یا می توان به صورت real-time با استفاده با وب کم این کار را انجام داد. در این جا ما از هر دو روش استفاده خواهیم کرد.

در ابتدا کافیسیت تا با رفتن به دایرکتوری مدل آموزش دیده، که در این پروژه آن را با نام my_ssd_mobnet_forUni ساختیم، رفته و شماره ی آخرین چک پوینت آموزشی را مشاهده کنیم(در این پروژه آخرین چک پوینت آموزشی با نام ckpt-3 ذخیره شده است). سپس کافیسیت تا آخرین چک پوینت را بارگذاری کنیم.

```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-3')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

آدرس دهی یک داده ی test بصورت استاتیک:

```
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'victory.6631cd89-528d-11ee-b15d-dcf505d95965.jpg')
```

شروع عملیات تشخیص شی مدل آموزش داده:

```
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
               for key, value in detections.items()}
detections['num_detections'] = num_detections

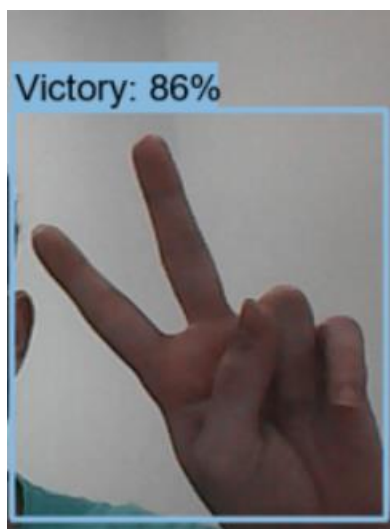
# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_detections,
    detections['detection_boxes'],
    detections['detection_classes'] + label_id_offset,
    detections['detection_scores'],
    category_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=5,
    min_score_thresh=.8,
    agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```

خروجی:



تشخیص اشیا بصورت real-time با استفاده از وب کم:

```
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

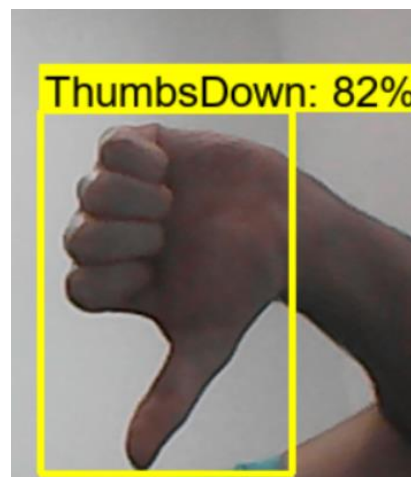
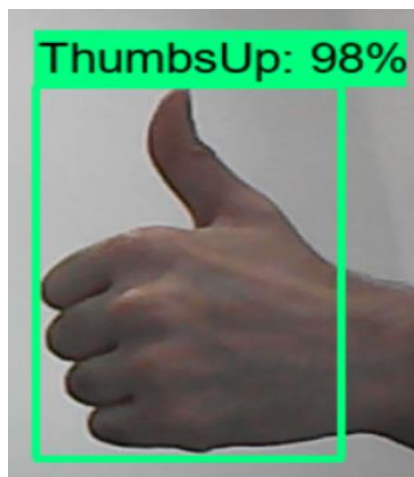
    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes'] + label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=5,
        min_score_thresh=.8,
        agnostic_mode=False)

    cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```



جمع بندی و نتیجه گیری

در نتیجه، این پروژه حول محور تشخیص اشیا با استفاده از TensorFlow، استفاده از قدرت CUDA و cuDNN برای یادگیری عمیق است. در حالی که ما به قلمرو یادگیری عمیق نفوذ کردیم، تصمیم گرفتیم تا جایی که ممکن است عمیق نشویم و پیچیدگی هایی را که ممکن است به وجود بیاید را تشخیص دهیم. مهم است که اذعان کنیم که فرآیندهای آموزش و آزمایش پروژه بر روی ماشین محلی ما، در یک محیط کاملاً ایزوله شده، اجرا شده است.

اگرچه عملکرد مدل کاملاً انتظارات اولیه ما را برآورده نکرد، مهم است که توجه داشته باشیم که این نتیجه را می توان با استفاده از داده های بیشتر و بهتر (با کیفیت) و گسترش مراحل آموزشی به طور قابل توجهی افزایش داد. علاوه بر این، همانطور که گفته می شود، "سخت افزار بهتر نتایج بهتری را به همراه دارد" و بهبود عملکرد سیستم می تواند منجر به فرآیندهای آموزشی کارآمدتر شود.

یکی از جنبه های کلیدی که باید در نظر گرفت این است که ما از یک مدل از پیش آموزش دیده TensorFlow Zoo استفاده کردیم که ما را از چالش های Overfitting یا Underfitting نجات داد. در حالی که این کار ما را ساده کرد، همچنین یادآوری می کند که اصلاح و بهینه سازی بیشتر امکان پذیر است، و دنبال کردن مدل های متناسب تر و قوی تر یک راه امیدوارکننده برای اکتشافات آینده است. به طور خلاصه، این پروژه به عنوان پله ای برای درک جامع تر از تشخیص اشیا، با فضای کافی برای رشد و ارتقاء در پیگیری نتایج همیشه در حال بهبود عمل می کند.

پایان