دانشکده مهندسی کامپیوتر

بینایی ماشین

مستند پروژه پایانی

عنوان . . . . . تشخیص پلاک مخدوش

مدرس . . . . . دکتر محمدی

اعضای گروه . . . . . سپهر باباپور - زهرا حسینی

تاریخ انتشار . . . . . ۲۱ بهمن ۱۳۹۹

فهرست مطالب

۱ مقدمه

۱.۱ چرا تشخیص پلاک مخدوش؟

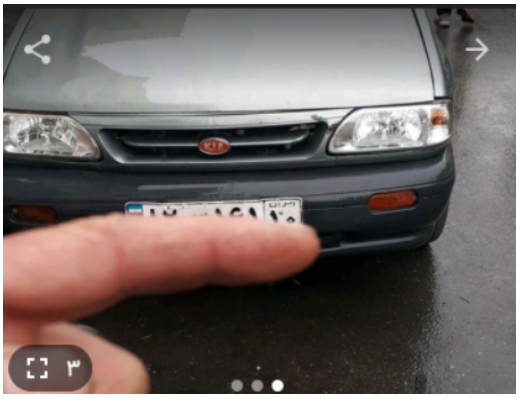
شماره پلاک خودرو یکی از مناسب‌ترین اقلام اطلاعاتی جهت احراز هویت خودروها می‌باشد، زیرا هر خودرو شماره‌ایی منحصر به فرد، مانند اثر انگشت برای انسان، دارد. این امر باعث شده تا از شناسایی پلاک خودروها، استفاده‌های متعددی در حوزه‌های مختلفی مانند راهنمایی رانندگی، فروشگاه‌های آنلاین خودرو، پارکینگ های عمومی و ... بشود.
اگر شما گواهینامه رانندگی داشته باشید، حتما با دوربین‌های ترافیکی آشنا هستید. تخلفات گوناگونی مانند تخلف سرعت لحظه‌ای، تخلف عبور از چراغ قرمز، تخلف پارک در محل غیرمجاز، تخلف عبور از مسیر خودروهای عمومی، تخلف رانندگی در معابر ورود ممنوع، تخلف سبقت غیرمجاز و ... وجود دارند، که برای ثبت هر تخلف سامانه مناسب آن می‌بایست نصب گردد. تکنولوژی‌های مختلفی برای این کار وجود دارد که در این مطلب به بررسی پردازش تصویر در این کاربرد می‌پردازیم.
در بسیاری از معابر شهری و چهارراه‌ها و همچنین معابر بین شهری از دوربین‌هایی برای کنترل و ثبت جرایم رانندگی استفاده می‌شود. در این کاربرد تصویر خودرو متخلف ثبت شده و پس از پردازش های لازم پلاک خودرو از تصویر استخراج می‌شود و تخلف انجام شده به حساب کاربری سامانه‌ایی که مالک خودرو در آن قرار دارد، افزوده می‌شود.

شکل ۱: تشخیص محدوده پلاک، سرعت و رنگ ماشین توسط دوربین‌های ترافیکی.

از دیگر موارد استفاده از پلاک، ثبت تصویر در هنگام معاینه فنی خودروها است. در فرایند معاینه فنی، قبل و بعد از ورود خودرو به مکان معاینه فنی تصویری از خودرو ثبت می‌شود. این ثبت تصویر برای اطمینان حاصل شدن از صحت انجام معاینه فنی است. همچنین این سامانه در برخی از موارد به سامانه ترافیکی نیز متصل است یعنی اگر خودرویی در معابر باشد که معاینه فنی نداشته باشد از پلاک خودرو تشخیص داده می‌شود که این خودرو فاقد معاینه فنی می‌باشد و تخلفی برای آن خودرو ثبت خواهد شد.
از آنجا که وجود مانع بر سر راه خودروها در عوارضی باعث کند شدن حرکت، ایجاد ترافیک، و به تبع آن آلودگی محیط

زیست میشود، راههای مختلفی برای حذف موانع موجود در عوارضی ها پیشنهاد شده است. یکی از این راهها استفاده از سامانه شناسایی پلاک خودرو میباشد. در این راه حل، خودروها بدون نیاز به توقف از عوارضیها عبور میکنند و سامانه ی شناسایی پلاک خودرو، شماره پلاک آنها را ثبت میکند و براساس شماره پلاک، عوارض مربوطه محاسبه میشود. در نهایت راننده ملزم به پرداخت عوارض در زمان مشخصی خواهد بود.

کاربرد جالب دیگری که برای ما تازگی داشت، تشخیص پلاک در فروشگاه‌های آنلاینی مانند دیوار است که مالک خودرو اقدام به فروش خودروی خود از طریق این سایت میکند. در این هنگام، فروشنده لازم است تصاویری از خودروی خود ثبت کند. نمایش پلاک خودرو می‌تواند نگرانی‌هایی را برای کاربران بابت نقض حریم شخصی‌شان به وجود بیاورد. پیش از هوشمندسازی این بخش از سایت، کاربران تصاویر نامطلوبی از خودرو خود ثبت می‌کردند تا پلاک آن مشخص نباشد یا اگر پلاک در کادر قرار داشت، آن را به شیوه‌های مختلفی از جمله رنگ‌آمیزی، قرار دادن شئ و ... آن را مخدوش می‌کردند تا اعداد آن مشخص نباشد. برای رفع این مشکل و بالا بردن کیفیت آگهی‌ها، سایت دیوار اقدام به مخدوش کردن خودکار پلاک خودروها کرد. در این فرایند ابتدا محدوده پلاک خودروها به صورت خودکار تشخص داده می‌شود و سپس برچسبی بر روی آن محدوده زده می‌شود تا دیگر پلاک قابل تشخیص نباشد. در زیر تصویری از خروجی سیستم دیوار را مشاهده می‌کنید:



شکل ۲: تصویر مخدوش کردن پلاک توسط کاربر

شکل ۳: تصویر مخدوش کردن پلاک توسط الگوریتم دیوار

از دیگر کاربردهای این سامانه‌ها می‌توان به درب ورود و خروج پارکینگ مجتمع‌های اداری یا تجاری، کنترل امنیت مرزها و سیستم حمل و نقل هوشمند اشاره کرد.

در پروژه درس مبانی بینایی کامپیوتر هدف ما نیز، در گام اول تشخیص پلاک خودرو و سپس تشخیص سالم یا مخدوش بودن آن است. از روش‌های مختلفی برای پیشبرد پروژه استفاده کردیم که اکثرا در کلاس درس در رابطه با آنها صحبت شده بود. در ادامه به بررسی جزئیات پروژه و روش‌های پیاده‌سازی آن می‌پردازیم.

۲ انواع الگوریتم تشخیص پلاک مخدوش

برای حل مسئله تشخیص پلاک مخدوش، الگوریتم‌های متفاوتی مطرح شده‌اند و مقالات و پروژه های متنوعی وجود دارند که بنا به کاربرد و محدودیت‌های مسئله از روش‌های مختلفی استفاده کرده‌اند. در ادامه به بررسی برخی از آن‌ها می‌پردازیم.

۱.۲ روش‌های کلاسیک پردازش تصویر

این روش‌ها الگوریتم‌های ساده‌ای‌اند که در بسیاری از مسائل پردازش تصویر کاربرد دارند. در اغلب این روش‌ها، یکی از روش‌های Edge Detection نظیر Canny یا Sobel، لبه‌های اشیا موجود در تصویر را مشخص می‌کنند. خروجی این تکنیک در الگوریتم Canny، تصویری باینری خواهد بود که تنها پیکسل‌های مربوط به لبه اشیا سفید است. با در دست داشتن لبه‌ها می‌توانیم الگوریتم‌های زیر را اجرا کنیم:

شکل ۴: تصویر خروجی الگوریتم Canny

* یافتن خطوط:

با استفاده از این روش می‌توان خطوط موجود در تصویر را به صورت تقریبی پیدا کرد. پس از انجام این کار می‌توانیم دسته خطوطی را که تشکیل چهارضلعی می‌دهند را پیدا کنیم و سپس تعیین کنیم که این چهارضلعی پلاک است یا خیر. می‌توان از الگوریتم‌هایی مانند Hough Transform برای یافتن خطوط استفاده کرد.

* یافتن نقاط گوشه:

با توجه به اینکه نقاط گوشه در تصاویر این مسئله بسیار زیاد هستند، این روش به تنهایی برای تشخیص پلاک مخدوش مناسب نیست. برای این موضوع، در ابتدا باید محل تقریبی پلاک در تصویر را مشخص کنیم و سپس به وسیله الگوریتم‌هایی مانند Harris Corner Detection گوشه ها را مشخص کنیم.

* یافتن نقاط گوشه:

می‌توان کانتورهای موجود در تصویر را پیدا کرد و سپس کانتورهایی که شبیه به مستطیل هستند را بررسی کرد.

* یافتن کاراکترهای پلاک:

اگر تصویر دارای کیفیت مناسبی باشد، می‌توان کانتور تمام شئ‌ها را پیدا کرد و از میان آن‌ها، عدد یا حرف را تشخیص دهیم. برای این کار کافی‌ست کانتورها را به یک مدل ساده classifier که بر روی اعداد و حروف فارسی آموزش داده شده بدهیم تا نوع هر کانتور مشخص شود. سپس تبیین کنیم کدام دسته از این کانتورها می‌توانند متعلق به یک پلاک باشند.

شکل ۵: تصویر یافتن کانتورهای یک پلاک

در روش‌های کلاسیک، سرعت بسیار بالا است ولی دقت بالایی وجود ندارد. این امر سبب میشود که روش های دیگری همچون شبکه عمیق بررسی شوند.

۲.۲ تشخیص شی (Object Detection)

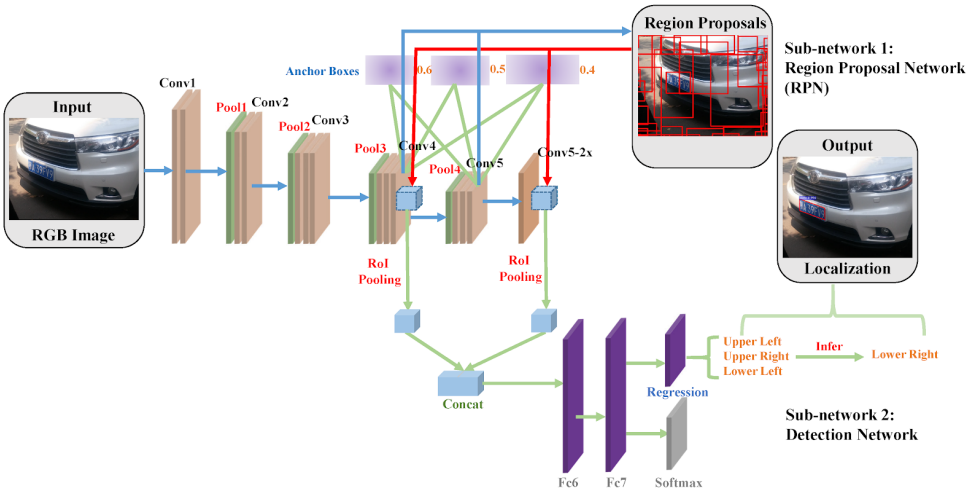
در این دست روش‌ها ،مدل‌های شبکه عصبی برای تشخیص اشیا آموزش می‌بینند.این مجموعه از مدل‌ها با توجه به مجموعه داده‌ای که بر روی آن آموزش داده می‌شوند، وظیفه دارند Bounding Box تمام اشیا مورد نظر موجود در تصویر را تشخیص دهند.

در زیر به تعدادی از روش‌های مرسوم این این الگوریتم می پردازیم:

* به عنوان مثال یکی از پیاده سازی های مرسوم، تشخیص مکان پلاک و سپس خواندن اعداد است. از این روش به این گونه استفاده می‌شود که شبکه ی طراحی شده را بر روی مجموعه داده‌ایی که برچسب آنها مکان پلاک ها است آموزش داده می‌شود. بعد از پیدا کردن مکان پلاک، اعداد و حروف آن را، به وسیله ی شبکه‌ایی که بر روی اعداد و حروف فارسی (یا زبان مورد نظر) آموزش دیده است، پیدا می‌کنند. در نهایت خروجی مطلوب حاصل می‌شود.

* از جمله روش‌های دیگر که کاربرد صنعتی‌تری دارند، استفاده از مدل‌های R-CNN است. این شبکه‌ها دارای چهار بخش اصلی هستند، در بخش اول با استفاده از CNN ها ویژگی‌های اولیه مانند گوشه‌ها و لبه‌ها را استخراج می‌کند. سپس در بخش بعدی و مهم‌ترین بخش که باعث ایجاد تمایز با سایر شبکه‌ها شده است، بخش Region

(Proposal Network) است در این بخش، مدل می‌خواهد تشخیص دهد که چه قسمت‌هایی از تصویر، ارزش و محتوای بیشتری دارد تا با تمرکز بر آن بخش‌ها، اشیا را پیدا کند. این بخش از شبکه در نظر دارد فریم‌های مستطیل شکلی از تصویر را که در آن شئ وجود دارد (کلاس آن شئ مهم نیست)، پیدا کند که به اصطلاح به این فریم‌ها (Region Proposal) گفته می‌شود. خروجی به بخش بعدی می رود تا بهترین (Region Proposal) ها (آن‌هایی که مدل پیش‌بینی کرده احتمال حضور شئ داخل‌شان بیشتر است.) انتخاب شوند و سپس مدل با استفاده از خروجی بخش اول، ویژگی‌های مربوط به هر Region را از آن استحراج می‌کند. در انتها برای Proposal های انتخاب شده مشخص می‌شود که شئ موجود در آن قسمت متعلق به کدام دسته است.



شکل ۶: نمایی از توضیحات فوق در قالب شکل

۳.۲ مدل‌های بخش‌بندی (Object Segmentation)

هدف مدل‌های بخش‌بندی این است که مشخص کنند هر پیکسل در تصویر متعلق به چه کلاسی است. آیا پیکسل متعلق به دسته انسان است؟ یا متعلق به خودرو؟ یا متعلق به هیچ شئ تعریف‌شده‌ای نیست؟ (متعلق به کلاس پس‌زمینه است.) بنابراین در حالت ساده می‌توان به آن به عنوان مسئله طبقه‌بندی هر پیکسل نگاه کرد که با آموزش مدل‌های بخش‌بندی می‌توانیم تمامی پیکسل‌های مربوط به پلاک رو بدست بیاوریم. وقتی که تمام پیکسل‌های مربوطه را داشته باشیم، به راحتی می‌توانیم چهارضلعی پلاک را مشخص کنیم.

۳ پیاده‌سازی

۱.۳ مجموعه داده (Dataset)

مهم‌ترین گام در حل مسئله شناخت صحیح و کامل مسئله است، لذا پیش از هرچیزی ما به بررسی دقیق مجموعه داده پرداختیم که به اختصار توضیحی در زیر ارائه می‌دهیم:

مجموعه تصاویر ورودی داده شده در این پروژه در مجموع دارای ۲۷۲۷ تصویر می‌باشه که به سه دسته مختلف تقسیم میشوند، پلاک سالم، پلاک مخدوش و غیر پلاک.در ادامه به تعریف هر یک از دسته های زیر میپردازیم:

* پلاک سالم: این دسته از تصاویر که در پوشه‌ی شماره صفر قرار دارند، تصاویری هستند که در هر کدام از آنها، یک پلاک خودرو بدون هیچ گونه نقصی به وضوح برای یک کاربر انسانی قابل تشخیص است. تمام ناحیه ی پلاک در این حالت در تصویر ورودی قرار داشته و کلیه اعداد، حروف و علائم مربوط به پلاک قابل تشخیص هستند. در این تصاویر محل، اندازه، زاویه قرارگیری و اثرات محیطی (مانند نورپردازی) مربوط به ناحیه‌ی پلاک در تصاویر مختلف این دسته متفاوت هستند. این موارد چالش‌هایی را همراه داشت که در طول مستند به شرح آنها نیز پرداخته میشود.

  این دسته از تصاویر بخش عمده‌ی مجموعه داده را به خود اختصاص می‌دهند که شامل ۱۸۷۶ تصویر است که حدود ۶۹ درصد از کل مجموعه است.

  نمونه‌ای از این تصاویر را می‌توانید در زیر مشاهده کنید:

شکل ۷: نمونه‌ای از تصاویر کلاس پلاک مخدوش

* پلاک مخدوش: این دسته از تصاویر که در پوشه‌ی شماره یک قرار دارند ،تصاویری هستند که در هر کدام از آنها، پلاک در تصویر موجود است اما تمام یا بخشی از یک یا چند عدد یا حرف از پلاک قابل تشخیص نمی‌باشد. مخدوش بودن پلاک عمدتاً به دلیل .پوشاندن بخشی از پلاک توسط یک شیء خارجی اتفاق میافتد. این دسته از تصاویر ۲۸۳ تصویر را شامل میشوند که در زیر می‌توانید نمونه‌ای از آن‌ها را مشاهده کنید:

شکل ۸: نمونه‌ای از تصاویر کلاس پلاک مخدوش

* غیرپلاک: این دسته از تصاویر که در پوشه‌ی شماره دو قرار دارند، تصاویری هستند که پلاکی در تصویر وجود ندارد.

این دسته از تصاویر نیز شامل ۵۶۸ تصویر هستند که در زیر می‌توانید برخی از آن‌ها را مشاهده کنید:

شکل ۹: نمونه‌ای از تصاویر کلاس پلاک مخدوش

۲.۳ تاریخچه: استفاده از شبکه **ResNet50** و اعمال شیفت برای داده افزایی

قبل از آنکه به توضیح پیاده‌سازی اصلی این پروژه بپردازم، بهتر دانستم به مواردی که در این پیاده‌سازی، پس از استفاده از هوش مصنوعی، برای حل مسئله به کار گرفته شده اشاره کوتاهی بکنم.

در ابتدای این پروژه، پس از بدست آوردن الگوریتمی برای یافتن ناحیه‌های پلاک، از آنجایی که در تمرین ۱۴ از شبکه‌های ResNet50 استفاده کرده بودیم، سعی در دسته‌بندی سه گروه مطلوب مسئله بوسیله این شبکه داشتم. پس از جست و جویی در ارتباط با پارامترهای تابع pre-process این شبکه، به صورت تصادفی به صفحه Keras Applications برخورد کردم. در این صفحه جدولی که در ابتدای صفحه بعد قسمتی از آن را نمایش دادم، باعث ایجاد کنجکاوی در من شد.

همانطور که در جدول مشاهده می‌کنید در بین تمامی مدل‌ها، مدل NASNetLarge دارای بهترین عملکرد در بین مدل‌ها می‌باشد، اما مشکلی که دارد، تعداد زیاد وزن‌های آن و به تبع آن حجم زیاد مدل می‌باشد.

با توجه به این جدول، در این پیاده‌سازی از مدل InceptionResNetV2 استفاده شده است. علت این انتخاب دارا بودن دقتی بالاتر از دو مدل ResNet152V2 و ResNet152 و پارامتری کمتر از این دو مدل می‌باشد. لذا نتیجه گرفتم از این مدل در ادامه این پیاده‌سازی استفاده کنم.

توضیحات بیشتر در ارتباط با این مدل را در قسمت طراحی مدل در بخش اصلی شرح می‌دهم.

لازم به ذکر است عملکرد مدل ResNet50ی که در ابتدای پروژه از آن استفاده شده بود در دو Notebook با عناوین ResNet50_1.ipynb و ResNet50_2.ipynb در دو آدرس زیر موجود می‌باشد:

History/ResNet50_1.ipynb

History/ResNet50_2.ipynb

عملکرد فایل دوم را می‌توانید در شکل ۱۱ موجود در صفحه بعد مشاهده بفرمایید.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

شکل ۱۰: جدول benchmarkهای مدل‌های پیاده‌سازی شده در keras

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219062272/219055592 [==============================] - 3s 0us/step
Epoch 1/10
219/219 [==============================] - 540s 2s/step - loss: 0.6251 - accuracy: 0.7598 - f1_score: 0.5311 - val_loss: 0.2661 - val_accuracy: 0.9191 - val_f1_score: 0.8356
Epoch 2/10
219/219 [==============================] - 519s 2s/step - loss: 0.2404 - accuracy: 0.9138 - f1_score: 0.8386 - val_loss: 0.2638 - val_accuracy: 0.9118 - val_f1_score: 0.8515
Epoch 3/10
219/219 [==============================] - 518s 2s/step - loss: 0.1537 - accuracy: 0.9460 - f1_score: 0.9125 - val_loss: 0.3447 - val_accuracy: 0.9118 - val_f1_score: 0.8109
Epoch 4/10
219/219 [==============================] - 513s 2s/step - loss: 0.1300 - accuracy: 0.9466 - f1_score: 0.9021 - val_loss: 0.1707 - val_accuracy: 0.9504 - val_f1_score: 0.8969
Epoch 5/10
219/219 [==============================] - 502s 2s/step - loss: 0.0819 - accuracy: 0.9732 - f1_score: 0.9466 - val_loss: 0.1394 - val_accuracy: 0.9559 - val_f1_score: 0.9112
Epoch 6/10
219/219 [==============================] - 499s 2s/step - loss: 0.1064 - accuracy: 0.9651 - f1_score: 0.9386 - val_loss: 0.2588 - val_accuracy: 0.9393 - val_f1_score: 0.8554
Epoch 7/10
219/219 [==============================] - 507s 2s/step - loss: 0.0829 - accuracy: 0.9703 - f1_score: 0.9441 - val_loss: 0.3452 - val_accuracy: 0.9136 - val_f1_score: 0.8127
Epoch 8/10
219/219 [==============================] - 504s 2s/step - loss: 0.0671 - accuracy: 0.9801 - f1_score: 0.9644 - val_loss: 0.1665 - val_accuracy: 0.9559 - val_f1_score: 0.9219
Epoch 9/10
219/219 [==============================] - 500s 2s/step - loss: 0.0624 - accuracy: 0.9749 - f1_score: 0.9541 - val_loss: 0.1198 - val_accuracy: 0.9577 - val_f1_score: 0.9215
Epoch 10/10
219/219 [==============================] - 511s 2s/step - loss: 0.0580 - accuracy: 0.9805 - f1_score: 0.9616 - val_loss: 0.1180 - val_accuracy: 0.9577 - val_f1_score: 0.9237
<tensorflow.python.keras.callbacks.History at 0x7f02ac162d68>
```

شکل ۱۱: عملکرد شبکه ResNet50 در ۱۰ epoch

درست است که این بخش نام تاریخچه به خود دارد، اما کاری که در این قسمت می‌خواهم شرح دهم پس از پیاده‌سازی نسخه نهایی انجام شده است و تنها بدلیل عدم موفقیت آن در این بخش قرار دادم.

پس از آنکه در نسخه نهایی بدلیل خارج شدن پلاک از تصویر بر اثر داده‌افزایی شیفت عرضی، این خصوصیت را از داده‌افزایی حذف کردم، با مشورت با استاد به راه‌حلی که در دو آدرس زیر آن را پیاده‌سازی کردم رسیدم.

WithShift/WithShift.ipynb

WithShift/WithShift (predict).ipynb

بدلیل کسب نتایج ضعیف، از پیاده‌سازی predict این راه‌حل منصرف شده‌ام.
در این راه‌حل پیشنهادی که برای رفع مشکل خارج شدن پلاک از تصویر ارائه شد، اضافه کردن پیکسل‌های مشکی به مرزهای تصویر می‌باشد. با این کار، در صورتی که پلاک در ناحیه اطراف مرز تصویر اصلی باشد، با گسترش مرزها احتمال خارج شدن پلاک از تصویر کاهش می‌یابد. اما پیاده‌سازی اصلی که برای این پروژه زده شده بود، دارای تصویر به قدر کافی بزرگی بود و لذا گسترش مرزها منجر به افزایش شدید زمان pre-process می‌شد. در نتیجه در این راه‌حل مجبور به کاهش سایز تصویر اصلی شدم. این کار منجر به کاهش دقت نهایی با وجود داده‌افزایی عرضی و ارتفاعی شد.
نتیجه ۱۰ epoch نخست این پیاده‌سازی را می‌توانید در زیر مشاهده کنید:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5
219062272/219055592 [==============================] - 1s 0us/step
Epoch 1/10
219/219 [==============================] - 429s 2s/step - loss: 0.6828 - accuracy: 0.7244 - f1_score: 0.4648 - val_loss: 0.4734 - val_accuracy: 0.8051 - val_f1_score: 0.5361
Epoch 2/10
219/219 [==============================] - 400s 2s/step - loss: 0.4337 - accuracy: 0.8227 - f1_score: 0.6041 - val_loss: 0.3775 - val_accuracy: 0.8566 - val_f1_score: 0.7162
Epoch 3/10
219/219 [==============================] - 397s 2s/step - loss: 0.3783 - accuracy: 0.8594 - f1_score: 0.6815 - val_loss: 0.4210 - val_accuracy: 0.8621 - val_f1_score: 0.6054
Epoch 4/10
219/219 [==============================] - 403s 2s/step - loss: 0.3466 - accuracy: 0.8613 - f1_score: 0.6985 - val_loss: 0.3633 - val_accuracy: 0.8640 - val_f1_score: 0.7091
Epoch 5/10
219/219 [==============================] - 398s 2s/step - loss: 0.3018 - accuracy: 0.8873 - f1_score: 0.7937 - val_loss: 0.4478 - val_accuracy: 0.8401 - val_f1_score: 0.6238
Epoch 6/10
219/219 [==============================] - 397s 2s/step - loss: 0.3036 - accuracy: 0.8710 - f1_score: 0.7634 - val_loss: 0.3541 - val_accuracy: 0.8805 - val_f1_score: 0.7290
Epoch 7/10
219/219 [==============================] - 394s 2s/step - loss: 0.2318 - accuracy: 0.9065 - f1_score: 0.8238 - val_loss: 0.2773 - val_accuracy: 0.9026 - val_f1_score: 0.8223
Epoch 8/10
219/219 [==============================] - 396s 2s/step - loss: 0.2234 - accuracy: 0.9137 - f1_score: 0.8430 - val_loss: 0.2833 - val_accuracy: 0.9044 - val_f1_score: 0.8283
Epoch 9/10
219/219 [==============================] - 396s 2s/step - loss: 0.2111 - accuracy: 0.9292 - f1_score: 0.8589 - val_loss: 0.2682 - val_accuracy: 0.9044 - val_f1_score: 0.8205
Epoch 10/10
219/219 [==============================] - 396s 2s/step - loss: 0.1915 - accuracy: 0.9255 - f1_score: 0.8564 - val_loss: 0.3036 - val_accuracy: 0.8952 - val_f1_score: 0.7822
```

شکل ۱۲: عملکرد شبکه InceptionResNetV2 با داده‌افزایی شیفت عرضی و طولی در ۱۰ epoch نخست

۳.۳ پیاده‌سازی اصلی: مقدمه

پیاده‌سازی اصلی این پروژه در ۴ Notebook در آدرس‌های زیر موجود می‌باشد:

NoShift/Number Plate Recognition.ipynb

NoShift/Number Plate Recognition (Train).ipynb

NoShift/Number Plate Recognition (Predict).ipynb

NoShift/main.ipynb

فایل اول که به نوعی اصلی‌ترین فایل است، پیاده‌سازی اصلی و ابتدایی پروژه در آن قرار دارد. فایل بعدی برای ادامه فرایند آموزش مدل در فایل قبلی پیاده‌سازی شده است و فایل سوم برای آزمایش نمونه‌های موجود در dataset پیاده‌سازی شده است.
در فایل آخر همانطور که در مستند پروژه آمده بود، تابعی با عنوان test در آخر فایل نوشته شده است که برای آزمایش نهایی پیاده‌سازی مورد استفاده قرار می‌گیرد. حال در ادامه به شرح پروژه که در واقع فایل نخست فوق می‌باشد می‌پردازیم.

۴.۳ پیاده‌سازی اصلی: بخش اول و دوم

در بخش نخست این پیاده‌سازی علاوه‌بر شرح پروژه به صورت مفصل و کاربردهای آن، کتابخانه‌های مورد نیاز پروژه را به پروژه اضافه کردیم.

در بخش بعدی ثوابت مورد نیاز پروژه را نوشتیم. در زیر به توضیح مختصری در مورد هر یک از آن‌ها می‌پردازیم:

* IMG_WIDTH: عرض تصاویری است که مدل با آن آموزش می‌بیند. در واقع تصاویر دیتاست به این عرض تغییر ابعاد داده می‌شوند.
* IMG_HEIGHT: ارتفاع تصاویری است که مدل با آن آموزش می‌بیند. در واقع تصاویر دیتاست به این ارتفاع تغییر ابعاد داده می‌شوند.
* EPOCHS: تعداد Epochهای فرایند آموزش را مشخص می‌کند. این متغیر همچنین برای آنکه در هر گام تنها به تعداد تصاویر مورد نیاز برای آموزش در اختیار مدل قرار گیرد، به تابع flow_from_directory نیز داده می‌شود.
* BATCH_SIZE: اندازه Batch در هر Epoch را مشخص می‌کند.
* STRUCTURING_ELEMENT_i: از این عناصر ساختاری برای بدست آوردن ناحیه پلاک استفاده شده است. توضیح رویدادی که پس از استفاده از هر یک از این عناصر رخ می‌دهد به صورت Comment در کد نوشته شده است.
* N_CLASSES: کلاس‌های مسئله را مشخص می‌کند. در این مسئله تعداد کل کلاس‌ها برابر با ۱۹۶ است.

۵.۳ پیاده‌سازی اصلی: بخش سوم

برای آوردن دیتاست به Notebook موجود بر روی Colab، فایل دیتاست را به صورت یک فایل zip به درایو منتقل کردم تا بتوانم در Notebook آن را به صورت مستقیم دانلود کنم.
برای این کار از پس از آنکه دسترسی این فایل را آزاد کردم، از id فایل موجود در درایو برای دانلود این فایل استفاده کردم.
در نهایت پس از دانلود فایل دیتاست به صورت zip، آن را unzip کردم.

۶.۳ پیاده‌سازی اصلی: بخش چهارم

در این قسمت نمایش نمونه‌های تصادفی از هر کلاس پرداختیم. برای این کار ابتدا اسامی تصاویر هر کلاس را در قالب لیست در آوردیم و سپس سه تصویر تصادفی از بین اسامی انتخاب کردیم و ان‌ها را برای هر کلاس نمایش دادیم.

۷.۳ پیاده‌سازی اصلی: بخش پنجم

به نوعی می‌توان گفت قلب اصلی پیاده‌سازی این پروژه در این بخش جای دارد. در این بخش تابعی با نام find_ROI وظیفه مشخص کردن ناحیه پلاک را برعهده دارد. توضیحات مفصل این تابع در متن کد نوشته شده است. علاوه بر این توضیحات، یک demo برای این تابع در آدرس زیر نوشته شده است:

NoShift/ROI_Demo

پس از یافتن ناحیه پلاک توسط تابع find_ROI، از تابع find_condidate_number_plate برای حذف (مشکی کردن) سایر نواحی به جز پلاک استفاده کردیم. این کار توسط تابع bitwise_and موجود در کتابخانه OpenCV انجام شده است.

در آخر تابعی تحت عنوان my_preprocessing_function نوشته شده است که وظیفه آن فراخوانی همزمان توابع پیش‌پردازش نوشته شده در کد و تابع پیش‌پردازش موجود در مدل InceptionResNetV2 می‌باشد. از این تابع در بخش گردآوردی داده‌ها استفاده شده است.

۸.۳ پیاده‌سازی اصلی: بخش ششم

در این قسمت علاوه‌بر گردآوری و برچسب‌گذاری داده‌ها، به پیاده‌سازی داده‌افزایی در این پروژه پرداختیم. برای این کار از خوصویات زیر استفاده کردیم:

* تصاویر جدید می‌توانستند تا ۲۰ درجه نسبت به تصاویر اصلی چرخیده شوند.
* تصاویر جدید می‌توانستند تا ۱۰ درصد نسبت به تصاویر اصلی بزرگتر شوند.
* تصاویر جدید می‌توانستند نسبت به تصاویر اصلی به صورت افقی معکوس شوند.

در این بخش علاوه بر گردآوری و داده‌افزایی به تقسیم داده بین دو گروه train و validation پرداختیم. برای این کار ۲۰ درصد از کل داده‌ها را به داده‌های validation اختصاص دادیم.

۹.۳ پیاده‌سازی اصلی: بخش هفتم

در این بخش به پیاده‌سازی مدل پرداختیم. تصویری از این مدل را می‌توانید در زیر مشاهده کنید:

```
def build_model_with_InceptionResNetV2():
    # Create InceptionResNetV2 model with imagenet weights.
    model = InceptionResNetV2(include_top=False, weights='imagenet', input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))

    # Make all layers trainable.
    for layer in model.layers:
        layer.trainable = True

    # Add avgPooling layer to reduce parameters and flatten the output.
    avgPooling = GlobalAveragePooling2D()(model.output)

    # Add two dense layers for classification.
    dense1 = Dense(1024, activation='relu')(avgPooling)
    dense2 = Dense(N_CLASSES, activation='softmax')(dense1)

    # Create final model.
    model = Model(model.input, dense2)

    return model
```

شکل ۱۳: تصویری از مدل

۱۰.۳ پیاده‌سازی اصلی: بخش هشتم و نهم

در این دو بخش، به آموزش و ذخیره مدل پرداختیم. تصویری از فرایند یادگیری این مدل در ۱۰ epoch دوم را می‌توانید در زیر مشاهده کنید:

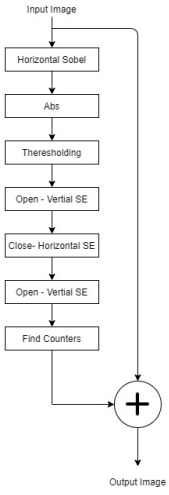
```
Epoch 1/10
219/219 [==============================] - 541s 2s/step - loss: 0.0497 - accuracy: 0.9849 - f1_score: 0.9751 - val_loss: 0.1407 - val_accuracy: 0.9688 - val_f1_score: 0.9375
Epoch 2/10
219/219 [==============================] - 517s 2s/step - loss: 0.0472 - accuracy: 0.9853 - f1_score: 0.9727 - val_loss: 0.1150 - val_accuracy: 0.9706 - val_f1_score: 0.9469
Epoch 3/10
219/219 [==============================] - 517s 2s/step - loss: 0.0467 - accuracy: 0.9872 - f1_score: 0.9793 - val_loss: 0.1458 - val_accuracy: 0.9577 - val_f1_score: 0.9153
Epoch 4/10
219/219 [==============================] - 521s 2s/step - loss: 0.0428 - accuracy: 0.9872 - f1_score: 0.9771 - val_loss: 0.2567 - val_accuracy: 0.9467 - val_f1_score: 0.8847
Epoch 5/10
219/219 [==============================] - 512s 2s/step - loss: 0.0753 - accuracy: 0.9711 - f1_score: 0.9513 - val_loss: 0.1455 - val_accuracy: 0.9706 - val_f1_score: 0.9482
Epoch 6/10
219/219 [==============================] - 525s 2s/step - loss: 0.0340 - accuracy: 0.9908 - f1_score: 0.9846 - val_loss: 0.2480 - val_accuracy: 0.9632 - val_f1_score: 0.9232
Epoch 7/10
219/219 [==============================] - 526s 2s/step - loss: 0.0281 - accuracy: 0.9899 - f1_score: 0.9833 - val_loss: 0.1652 - val_accuracy: 0.9393 - val_f1_score: 0.8980
Epoch 8/10
219/219 [==============================] - 525s 2s/step - loss: 0.0457 - accuracy: 0.9867 - f1_score: 0.9769 - val_loss: 0.1494 - val_accuracy: 0.9669 - val_f1_score: 0.9385
Epoch 9/10
219/219 [==============================] - 525s 2s/step - loss: 0.0409 - accuracy: 0.9872 - f1_score: 0.9790 - val_loss: 0.1599 - val_accuracy: 0.9614 - val_f1_score: 0.9199
Epoch 10/10
219/219 [==============================] - 521s 2s/step - loss: 0.0370 - accuracy: 0.9885 - f1_score: 0.9807 - val_loss: 0.2265 - val_accuracy: 0.9522 - val_f1_score: 0.8957
```

شکل ۱۴: عملکرد شبکه InceptionResNetV2 بدون داده‌افزایی شیفت عرضی و طولی در ۱۰ epoch دوم

۴ سایر ایده های پیاده سازی

۱.۴ ایده اول:

در ابتدا با پیدا کردن مکان پلاک شروع کردیم. روشهای متفاوتی را امتحان کردیم تا به نتیجه ی قابل قبولی رسیدیم،مسیر پیش گرفته شده و روش نهایی را در زیر بررسی میکنیم. از روشی هایی که سرکلاس، جلسه ی ۲۷ ام مطرح شد استفاده کردیم. در notebook ایی با نام Plate-Detection-Using-Sobel-Session-27.ipynb این روش را پیاده سازی کردیم.مراحل طی شده و عملگرهای به کاربرده شده به ترتیت زیر است:



شکل ۱۵: مراحل پیاده سازی

نتیجه ایی که بر روی تصویر موجود در اسلاید های درس بود بسیار خوب بود ولی بر روی سایر عکس های مجموعه

داده نتیجه ی خوبی نمیداد. پس از مشاهده خروجی ها مشخص شد که این روش بسیار وابسته به ویژگی های تصویر از جمله مقیاس، نور، زاویه و ... است.در زیر دو نمونه از خروجی ها را مشاهده میکنید. همانطور که مبینید تصویر زیر خروجی بسیار خوبی دارد و مکان پلاک را به درستی تشخیص داده است.نتیجه ی عملگر آخر بر روی تصویر نیز در ادامه آورده شده است.همانطور که مشخص است مکان های تشخیص داده شده برای پلاک به صورت پیوسته و دقیق هستند.

شکل ۱۶: نحوه عملکرد الگوریتم

ولی در تصویر بعدی که از نمونه خروجی های بد این کد است، مشاهده میشود که پلاک را به طور دقیق تشخیص نداده و آن را به صورت بخش های کوچکی تشخیص داده است.این امر نتیجه ی عمل نکردن درست عملگر ها بر روی تصویر هستند.از دلایل این مشکل وابسته بودن اندازه عنصر ساختاری به تصویر است. همچنین حفره یا نویز ایجاد شده در تصویر نیز تاثیرگزار بر انتخاب نوع عنصر ساختاری و اندازه ی آن است.نتایج به دست آمده منجر شد تا این روش را کنار بگذاریم و روش های دیگری را در پی بگیریم.

شکل ۱۷: نحوه عملکرد الگوریتم -نتیجه نامطلوب

۲.۴ ایده دوم:

روش دیگری که پیش گرفتیم که در قسمت "الگوریتم" اشاراتی به آن شد.از مجموعه داده ی اعداد فارسی هدی استفاده کردیم و شبکه ی نسبتا ساده ایی را به وسیله ی این مجموعه داده آموزش دادیم.در ادامه جزئیات بیشتری از این روش ذکر خواهیم کرد. معرفی مجموعه داده هدی: مجموعه ارقام دستنویس هدی که اولین مجموعه‌ی بزرگ ارقام دستنویس فارسی است، مشتمل بر ۱۰۲۳۵۳ نمونه دستنوشته سیاه سفید است. داده های این مجموعه از حدود ۱۲۰۰۰ فرم ثبت نام آزمون سراسری کارشناسی ارشد سال ۱۳۸۴ و آزمون کاردانی پیوستهی دانشگاه جامع علمی کاربردی سال ۱۳۸۳ استخراج شده است. خصوصیات این مجموعه داده به شرح زیر است: درجه تفکیک نمونه‌ها: ۲۰۰ نقطه بر اینچ

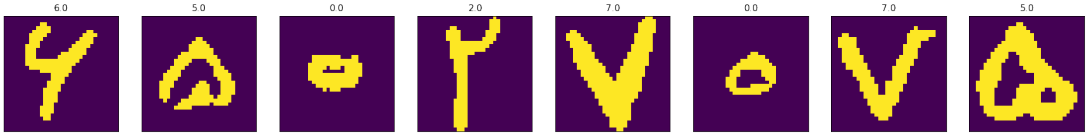
تعداد کل نمونه‌ها: ۱۰۲۳۵۲ نمونه

تعداد نمونه‌های آموزش: ۶۰۰۰ نمونه از هر کلاس

تعداد نمونه‌های آزمایش: ۲۰۰۰ نمونه از هر کلاس

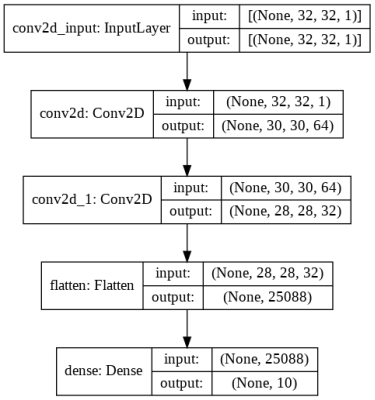
سایر نمونه‌ها: ۲۲۳۵۲ نمونه

در ابتدا کار کردن با این مجموعه داده مقداری پیچیده بود با استفاده از کد کمکی همراه مجموعه داده را بارگذاری کردیم.تصاویر دارای ابعاد مختلفی بودند که به سایز ۳۲*۳۲ تاخیر یافتند. نمونه هایی از این مجموعه داده:



شکل ۱۸: نمونه داده از مجموعه داده هدی

معماری شبکه:شبکه طراحی شده دارای لایه های کانوولشنی است که جزئیات بیشتر آن در تصویر زیر مشخص است.



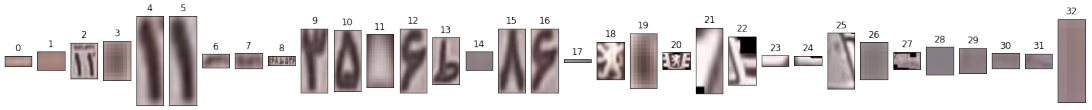
شکل ۱۹: معماری شبکه

پس از آموزش این شبکه از کد استخراج پلاک استفاده کردیم تا مکان پلاک بدست بیاید.به عنوان نمونه تصویر زیر

را انتخاب کردیم که خروجی پلاک آن به صورت زیر بود:

شکل ۲۰: تشخیص پلاک

کانتوری های تصویر حاصله را بدست آوردیم و سپس هر کانتور را جدا کردیم.تصویر زیر نتیجه ی جدا کردن کانتور های عکس پلاک است.



شکل ۲۱: کانتورها

همانطور که مشخص است اعداد به خوبی تشخیص داده اند.بعد ازپردازش هایی برای رفع حفره و نویز و اعمال تغییرات لازم در اندازه ورودی، هر کدام از تصاویر بریده شده را به شبکه دادیم که نتیجه ی زیر حاصل شد. اعداد نوشته شده در بالای تصاویر ،نتیجه ی شبکه از پیش بینی عدد تصویر مربوطه است.

شکل ۲۲: کانتورها

مشاهده میشود که خطای بسیار زیادی دارد. بر روی ورودی ما با اینکه دقت شبکه روی داده های تست نزدیک به ۹۹ درصد بود. به نظرما این روش نیازمند کیفیت مناسب تصاویر و همچنین دانستن محل دقیق پلاک است.همانطور که میبینید از مشکلات شبکه تشخیص نماد پژو به عنوان عدد ۳ است.پس این روش نیز کنار گذاشتیم و به سراغ روش های دیگر رفتیم.

۵ منابع

* Underfitting and Overfitting in Machine Learning

* Transfer learning - Wikipedia
* Approach pre-trained deep learning models with caution
* The Applications and Benefits of a PreTrained Model — Kaggle's DogsVSCats
* The Applications and Benefits of a PreTrained Model — Kaggle's DogsVSCats
* Hoda dataset