Persian car plate detection & OCR

این پروژه از سه مرحله اصلی تشکیل شده است :

1) **آموزش مدل yolov7 بر روی پلاک ایرانی**
2) **ساختن یک مدل ocrجدید برای تشخیص اعداد و حروف فارسی**
3) **سگمنت کردن اعداد و حروف روی پلاک**

آموزش مدل yolov7 بر بروی پلاک ایرانی :

مراحل fine tune کردن مدل yolov7 با کمک ویدیو زیر انجام میشود :

لینک ویدیو :

https://www.youtube.com/watch?v=bgAUHS1Adzo&list=PL4sqgbpSjuJjZz_YyNx5e2jPrR9kun2l8

این مراحل عبارت است از :

1. پیدا کردن دیتاستی از پلاک های ایرانی و دیتاستی از پلاک های مشابه ایرانی :
   لینک دیتا ست ایرانی :
   https://www.kaggle.com/datasets/skhalili/iraniancarnumberplate
   لینک دیتاست مشابه ایرانی :
   https://www.kaggle.com/datasets/andrewmvd/car-plate-detection
2. ادغام و انجام پیش بردازش های لازم روی دیتاست ها با استفاده از roboflow
3. کلون گرفتن از پروژه yolov7 و استفاده از api ها برای آموزش که این مراحل در این کولب انجام شده است :لینک کولب :
   https://colab.research.google.com/drive/1VNvmJlSCaLX3qrP7Bsl4_OvzJk847mqh?usp=sharing
4. در آخر دانلود بهترین وزن های مدل آموزش دیده

قسمت چالشی این پروژه مرحله ocr پلاک تشخیص داده شده است . زیرا در ابتدا از لایبرری easy ocr استفاده میکنیم اما مشاهده میشود که عملکرد خوبی بر زبان فارسی ندارند . از این رو خود باید با استفاده از دیتاست های فارسی ، مدل ocr جدیدی آموزش دهیم .

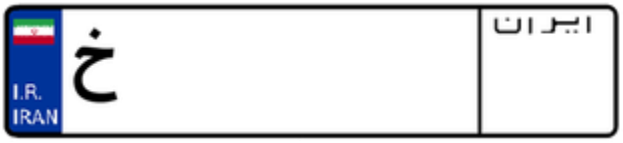
ساختن یک مدل ocr جدید برای تشخیص اعداد و حروف فارسی :

در این مرحله 3 رویکرد برای ساختن مدل اجرا میشود :

1. آموزش مدل بر روی تصاویر کراپ شده پلاک :

ابتدا تصور میشد که هر عدد و هر حرف با رعایت فاصله یکسانی در پلاک نمایش داده میشوند . به همین دلیل در اولین رویکرد ، مدلی را روی تصاویر حرف و عدد کراپ شده پلاک آموزش داده شد .

دیتاست : برای به دست آوردن دیتا ست ، از پروژه ای استفاده شد که پلاک فارسی به همراه با لیبل آن تولید میکند . https://github.com/amirmgh1375/iranian-license-plate-recognition :



با انجام تغییراتی در کد این پروژه پلاک های تک حرف یا تک عددی ساخته شد .
در مرحه بعد ، قسمت حرف یا عدد این پلاک برای مثال از پیکسل 20 تا 40 جدا میشد و به عنوان دیتای ما ذخیره میشد.
به این ترتیب جمعا 900 دیتا ساخته شد و مدل cnn خیلی ساده ای روی آن آموزش داده شد .
نتیجه روی پلاک واقعی خیلی وحشتناک بد بود ، زیرا حروف و اعداد در پلاک ها با فاصله یکسانی قرار نگرفته اند و با کراپ کردن پلاک با فواصل پیکسلی یکسان (مثلا هر 20 یا 30 پیکسل) نمیتوان به تک حرف و عدد دسترسی داشت ، و هر تصویر کراپ شده شامل دو حرف میشد .
به همین دلیل ، براس ساختن مدل از روش دیگری استفاده میشود .

2. آموزش مدل بر بری دیتاستی از اعداد و حروف فارسی :

لینک دیتاست : https://www.kaggle.com/datasets/mehdisahraei/persian-alpha

	class_names	labels_nums
0	ye	0
1	p	1
2	n	2
3	d	3
4	b	4
5	m	5
6	kh	6
7	alef	7
8	he	8

این دیتاست شامل 43 کلاس (32 حرف فارسی و 9 عدد و کلاس 5 بخاطر تغییر نوشتن آن 2 بار آورده شده است)است که هر کلاس شامل 100 تصویر است .
دیتا ست را با ضرایب 80 و 10 و 10 ، به دادگان test , validation , train تقسیم میکنیم.
در این مرحله ، عکس های دیتاست را که ابعاد بزرگ و مختلفی دارد، به ابعاد 32 در 32 ، تبدیل میکنیم و همچنین آن هارا grayscale میکنیم (چون این دیتاست به صورت دیفالت سفید مشکی است) که برای مدل ما مناسب باشند .
همچنین به هر حرف یا عدد ، عددی اختصاص میدهیم که برای آموزش مدل دچار مشکل نشویم .
برای راحتی کار این حروف و اعداد معادل هم را در

استفاده میکنیم که برای decodeو encode کردن y، به راحتی ازدیتا فریم استفاده کنیم . نمونه :

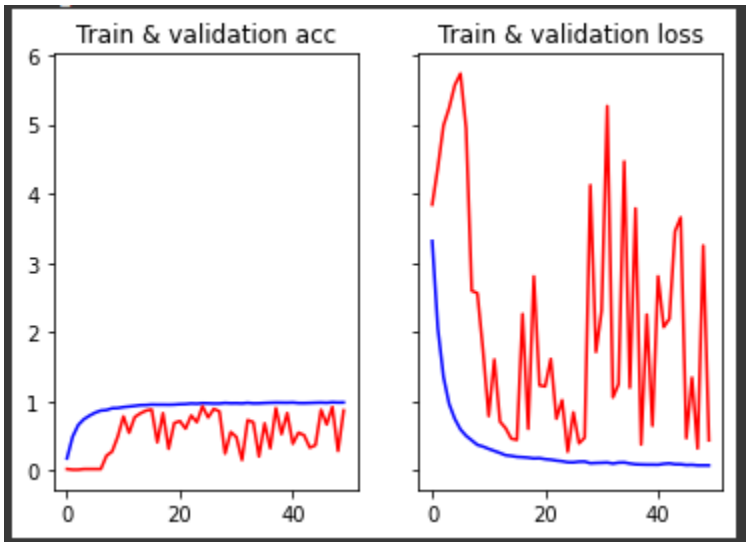
مدل :
برای معماری پایه این مدل ، از معماری مدل LeNet-5 استفاده میکنیم که این یک مدل ساده متشکل از یک لایه کانولوشن با یک لایه max-pooling دو بار به دنبال آن دو لایه کاملاً متصل با خروجی softmax در پایان است. برای اینکه مدل عملکرد قوی تری داشته باشد
خود لایه های BatchNormalization و drop out را به مدل اضافه میکنیم . این مدل را با آپتیمایزر adam و لاس categorical_crossentropy در 50 ایپاک و بچ سایز 64 آموزش میدهیم .

Layer (type)	Output Shape	Param #
convolution_1 (Conv2D)	(None, 28, 28, 32)	832
convolution_2 (Conv2D)	(None, 24, 24, 32)	25600
batchnorm_1 (BatchNormalization)	(None, 24, 24, 32)	128
activation_5 (Activation)	(None, 24, 24, 32)	0
max_pool_1 (MaxPooling2D)	(None, 12, 12, 32)	0
dropout_1 (Dropout)	(None, 12, 12, 32)	0
convolution_3 (Conv2D)	(None, 10, 10, 64)	18496
convolution_4 (Conv2D)	(None, 8, 8, 64)	36864
batchnorm_2 (BatchNormalization)	(None, 8, 8, 64)	256
activation_6 (Activation)	(None, 8, 8, 64)	0
max_pool_2 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
fully_connected_1 (Dense)	(None, 256)	262144
batchnorm_3 (BatchNormalization)	(None, 256)	1024
activation_7 (Activation)	(None, 256)	0
fully_connected_2 (Dense)	(None, 128)	32768
batchnorm_4 (BatchNormalization)	(None, 128)	512
activation_8 (Activation)	(None, 128)	0
fully_connected_3 (Dense)	(None, 84)	10752
batchnorm_5 (BatchNormalization)	(None, 84)	336
activation_9 (Activation)	(None, 84)	0
dropout_3 (Dropout)	(None, 84)	0
output (Dense)	(None, 43)	3655

عملکرد این مدل بسیار نامطلوب بود . با اینکه دقت مدل روی دادگان train در هر ایپاک افزایش پیدا میکرد ، اما دقت روی دادگان validation هرچند ایپاک دوباره بسیار پایین میرفت .در عکس زیر عملکرد مدل در چند ایپاک آخر نشان داده شده است که بسیار روی validationبد است .

```
Epoch 43/50
54/54 [==============================] - 20s 381ms/step - loss: 0.1027 - accuracy: 0.9738 - val_loss: 2.1907 - val_accuracy: 0.5093
Epoch 44/50
54/54 [==============================] - 19s 347ms/step - loss: 0.0903 - accuracy: 0.9762 - val_loss: 3.4569 - val_accuracy: 0.3302
Epoch 45/50
54/54 [==============================] - 18s 330ms/step - loss: 0.0899 - accuracy: 0.9794 - val_loss: 3.6575 - val_accuracy: 0.3651
Epoch 46/50
54/54 [==============================] - 18s 325ms/step - loss: 0.0794 - accuracy: 0.9817 - val_loss: 0.4657 - val_accuracy: 0.8767
Epoch 47/50
54/54 [==============================] - 19s 354ms/step - loss: 0.0820 - accuracy: 0.9785 - val_loss: 1.3436 - val_accuracy: 0.6651
Epoch 48/50
54/54 [==============================] - 18s 329ms/step - loss: 0.0742 - accuracy: 0.9858 - val_loss: 0.3178 - val_accuracy: 0.9186
Epoch 49/50
54/54 [==============================] - 18s 327ms/step - loss: 0.0741 - accuracy: 0.9820 - val_loss: 3.2516 - val_accuracy: 0.2814
Epoch 50/50
54/54 [==============================] - 18s 330ms/step - loss: 0.0736 - accuracy: 0.9837 - val_loss: 0.4368 - val_accuracy: 0.8674
```

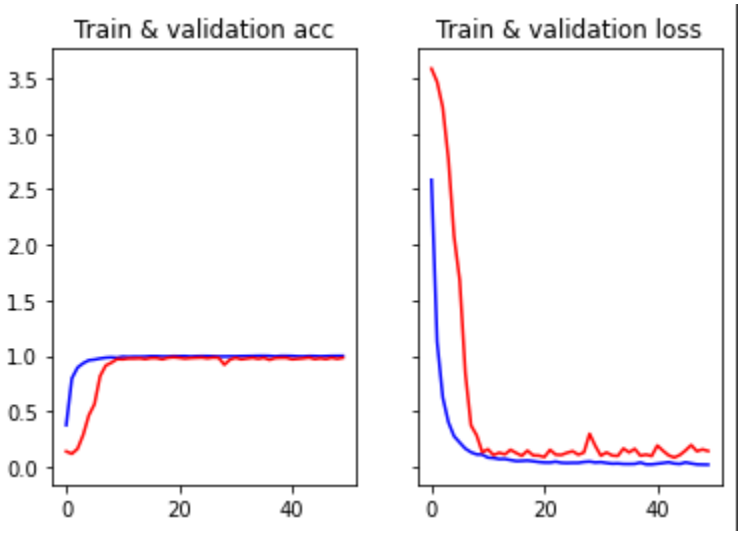
حتی در نمودار ها نویز بسیار و غیر قابل اعتماد بودن مدل مشخص است .



این مدل روی دیتاست تست هم عملکرد بسیار بدی داشت و روی پلاک واقعی هیچ حرف و عددی را نتوانست درست پیش بینی کند .

عملکرد خیلی بد مدل به این علت بود که عکس های واقعی ما ابعاد بزرگی داشتند و آنهارا به 32 در 32 ریسایز کردیم که باعث شد کیفیت آن بسیار بد شود . برای حل این مشکل ، تصمیم گرفتم به جای کوچک کردن عکس ، ابتدا ، ناحیه حرف یا عدد را در عکس تشخیص بدهم (با استفاده از روش کانتور و باندینگ باکس که در ادامه توضیح میدهم)، آن قسمت را از عکس کراپ کنم و سپس قسمت جدا شده را به عنوان x استفاده و ریساز کنم. با این کار کیفیت عکس حفظ شد و دقت عملکرد مدل بسیار بالا رفت .

```
54/54 [==============================] - 21s 387ms/step - loss: 0.0267 - accuracy: 0.9974 - val_loss: 0.1908 - val_accuracy: 0.9674
Epoch 42/50
54/54 [==============================] - 19s 359ms/step - loss: 0.0326 - accuracy: 0.9942 - val_loss: 0.1436 - val_accuracy: 0.9721
Epoch 43/50
54/54 [==============================] - 21s 395ms/step - loss: 0.0398 - accuracy: 0.9933 - val_loss: 0.1022 - val_accuracy: 0.9767
Epoch 44/50
54/54 [==============================] - 19s 357ms/step - loss: 0.0284 - accuracy: 0.9974 - val_loss: 0.0822 - val_accuracy: 0.9837
Epoch 45/50
54/54 [==============================] - 21s 390ms/step - loss: 0.0244 - accuracy: 0.9965 - val_loss: 0.1054 - val_accuracy: 0.9698
Epoch 46/50
54/54 [==============================] - 19s 362ms/step - loss: 0.0387 - accuracy: 0.9939 - val_loss: 0.1465 - val_accuracy: 0.9767
Epoch 47/50
54/54 [==============================] - 20s 379ms/step - loss: 0.0289 - accuracy: 0.9959 - val_loss: 0.1946 - val_accuracy: 0.9698
Epoch 48/50
54/54 [==============================] - 21s 388ms/step - loss: 0.0204 - accuracy: 0.9980 - val_loss: 0.1389 - val_accuracy: 0.9814
Epoch 49/50
54/54 [==============================] - 22s 398ms/step - loss: 0.0187 - accuracy: 0.9980 - val_loss: 0.1553 - val_accuracy: 0.9721
Epoch 50/50
54/54 [==============================] - 19s 358ms/step - loss: 0.0177 - accuracy: 0.9985 - val_loss: 0.1408 - val_accuracy: 0.9814
```



دقت مدل بر روی دادگان validation:
0.98
و برروی دادگان تست :
0.99

با توجه به نمودار هاهم متوجه میشویم که مشکل غیرقابل اعتماد بودن و ن.سانی بودن نتیجه ولیدیشن حل شده است .

دقت مدل را روی دادگان تست :

با توجه به عملکرد خوب مدل ، از همین مدل

برای خواندن اعداد و حروف پلاک استفاده میشود .

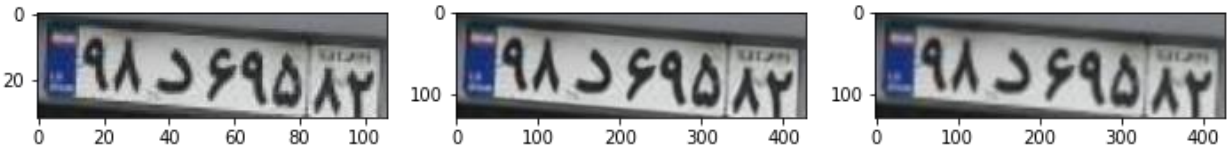
	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.86	0.92	0.89	13
2	1.00	1.00	1.00	9
3	1.00	1.00	1.00	12
4	1.00	1.00	1.00	6
5	1.00	1.00	1.00	9
6	1.00	1.00	1.00	9
7	1.00	1.00	1.00	12
8	1.00	0.88	0.93	8
9	1.00	1.00	1.00	13
10	1.00	1.00	1.00	9
11	0.93	0.93	0.93	14
12	1.00	1.00	1.00	8
13	1.00	1.00	1.00	11
14	1.00	1.00	1.00	15
15	1.00	1.00	1.00	8
16	1.00	1.00	1.00	9
17	1.00	0.94	0.97	16
18	1.00	1.00	1.00	8
19	1.00	1.00	1.00	5
20	1.00	1.00	1.00	8
21	1.00	1.00	1.00	13
22	0.92	0.69	0.79	16
23	0.67	1.00	0.80	4
24	1.00	1.00	1.00	13
25	1.00	1.00	1.00	9
26	1.00	1.00	1.00	8
27	1.00	1.00	1.00	8
28	1.00	1.00	1.00	10
29	1.00	1.00	1.00	6
30	1.00	0.90	0.95	10
31	1.00	1.00	1.00	11
32	1.00	1.00	1.00	12
33	0.86	1.00	0.92	6
34	1.00	1.00	1.00	11
35	0.90	1.00	0.95	9
36	1.00	1.00	1.00	13
37	1.00	1.00	1.00	9
38	1.00	1.00	1.00	11
39	0.75	0.90	0.82	10
40	1.00	1.00	1.00	12
41	1.00	1.00	1.00	8
42	1.00	1.00	1.00	9
accuracy			0.97	430
macro avg	0.97	0.98	0.98	430
weighted avg	0.98	0.97	0.97	430

سگمنت کردن اعداد و حروف روی پلاک :

برای تشخیص اعداد روی پلاک ، اعمالی را با کتابخانه opencv روی عکس انجام میدهیم .

قبل شروع این موضوع 2 مرحله را روی عکس پیاده میکنیم :

1) با استفاده از مدل های آماده opencv که باعث افزایش کیفیت عکس میشوند ، کیفیت پلاک های تشخیص داده شده را ، افزایش میدهیم . من از مدل EDSR_x4 استفاده کردم . در تصویر زیر اولین عکس از سمت چپ عکس اصلی، دومین عکس ، عکس بهبود یافته با مدل EDSR_x4 و عکس آخر تصویری است که با روش عادی resize شده است . میبینیم که اعداد ددر عکس دوم کیفیت بهتری دارند و مشخص تر هستند.
لینک دانلود مدل :
https://github.com/Saafke/EDSR_Tensorflow/tree/master/models

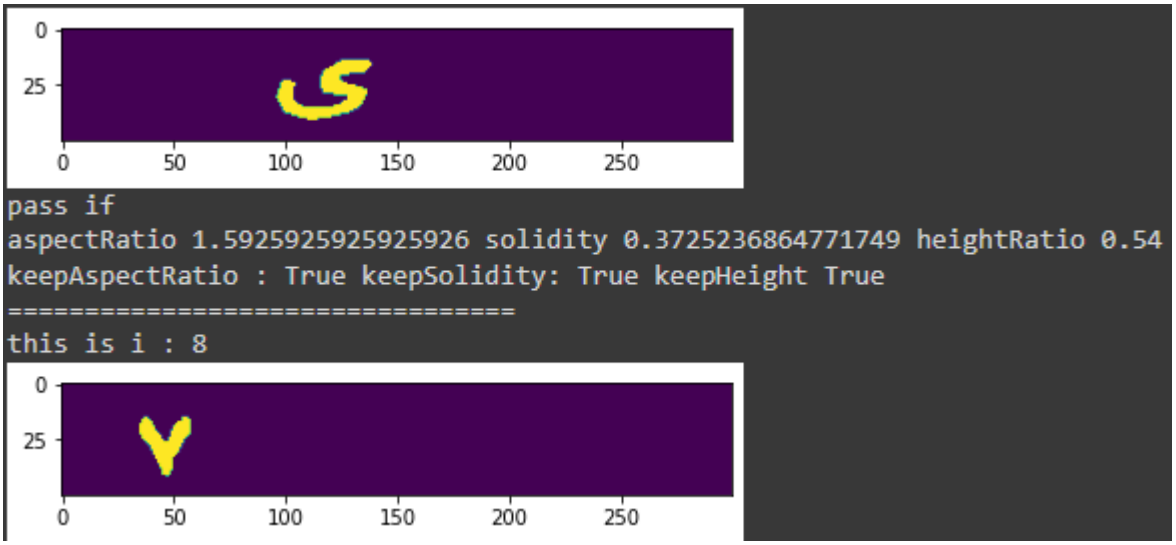


2) در دومین مرحله بررسی میشود که آیا پلاک مستقیم از یا کج ، بعد از تشخیص پلاک مستقیم میشود :

پس اعمال این تغییرات ، عملاتی برای تشخیص اعداد روی تصویر انجام میشود :

1) تصویر با کیفیت و مستقیم شده را به ابعاد (300 و 50) تغییر میدهیم . زیرا بعضی از پلاک ها بسیار کوچک هستند و عمل تشخیص رو آن صورت نمیگیرد .
2) تصویر را به rgb تبدیل میکنیم و عمل adaptiveThreshold را روی آن اعمال میکنیم . این کار باعث میشود اعداد یا نویز ها به صورت کلی از بکگراند متمایز شوند .
3) با استفاده از measure.label ، قسمت های مختلف متمایز شده را به دست می آوریم.
4) با استفاده از cv2. findContours سعی میکنیم قسمت های عدد را جدا کنیم .
5) با حساب aspect_ratio ، solidity و height_ratio و اعمال شرط هایی درباره بازه این ویژگی ها ، تشخیص میدهیم که آیا کانتور عدد یا حرف است ، یا خط و خطوط و نویز .

6) بعد از تشخیص درست بودن کانتور ، با cv2. boundingRect، ابعاد مناطق مد نظر را به دست میاوریم.



نمومنه تشخیص اعداد یا حروف

بعد از تشخیص باندیگ باکس ها ، این نقاط را از روی عکس اصلی جدا میکنیم و برای مدل ocr میفرستیم تا تشخیص انجام شود .

نمونه تشخیص مدل :

نتیجه گیری:

با اینکه این پروژه پیشرفت هایی داشته است ، اما هنوز مشکلاتی دارد :

1) با اینکه نتایج مدل بر دادگان تست بسیار خوب بود ، ولی همچنان عملکرد راضی کننده ای بر روی بعضی از کلاس ها ندارد . مثلا اکثر مواقع عدد 3 را 2 و عدد 6 را حرف ق یا ض تشخیص میدهد . همچنین اگر پلاک خیلی کج باشد ، اصلا حروف و اعداد را به خوبی تشخیص نمیدهد.
برای حل این مشکلات باید دیتاستی بزرگ تر پیدا کنیم و بهتر است دیتاستی به طور اختصاصی روی پلاک پیدا کنیم و آموزش دهیم .

2) اعداد مرزی ای که برای تشخیص عدد یا حرف بودن کانتور ها استفاده میکنیم ، با اینکه اکثر مواقع درست عمل میکند ، اما در بعضی مواقع این مرز ها باعث میشود که یعضی علایم و نویز هاهم به عنوان عدد یا حرف تشخیص داده بشوند .
و همچنین اگر تصویر پلاک ما تیره یا بیکیفیت باشد ، در مرحله threshold ، لایه اصلی اعداد از بین میرود . با بررسی های بیشتر میتوان راه های بهتری برای پیدا کردن نقاط مدنظر پیداکرد .