

## فهرست گزارش سوالات

۳	CNN – ۱
۳۱	Transfer Learning – ۲
۴۳	Semantic Segmentation – ۳
۵۲	Object Detection – ۴

## CNN – ۱

مجموعه داده ها را توسط کد داده شده ذخیره کردیم. داده های ما شامل  $60000 \times 28 \times 28$  تصویر در مقیاس خاکستری از  $10 \times 10$  دسته مد به همراه یک مجموعه آزمایشی از  $10000 \times 28 \times 28$  تصویر است.

سپس ماتریس های  $28 \times 28$  مان را به یک آرایه به اندازه  $784 \times 1$  تایی reshape کردیم و سپس پیش پردازش های لازم یعنی نرمال سازی و استانداردسازی را روی آنها انجام دادیم، و در نهایت آنها را به داده های با ابعاد اولیه برگرداندیم که همان  $60000 \times 28 \times 28$  است.

در این فرآیند نرمال سازی و استانداردسازی را انجام می دهیم. که نرمال سازی و استانداردسازی باعث می شود که  $x$  های توزیعی از بین بروند. از کتابخانه `sklearn.preprocessing` استفاده می کنیم. هنگامی که از `MinMaxScaler` استفاده می شود، به آن نرمال سازی نیز می گویند و تمام مقادیر را در محدوده بین  $(0,1)$  اسکیل می کند. با فرمول:

$$x = \frac{Value - Min}{Max - Min}$$

و با `StandardScaler` تحت استاندارد سازی قرار می گیرد و مقدار آن بین  $(-3,3)$  قرار می گیرد. با فرمول:

$$x = \frac{x - mean}{std - deviation}$$

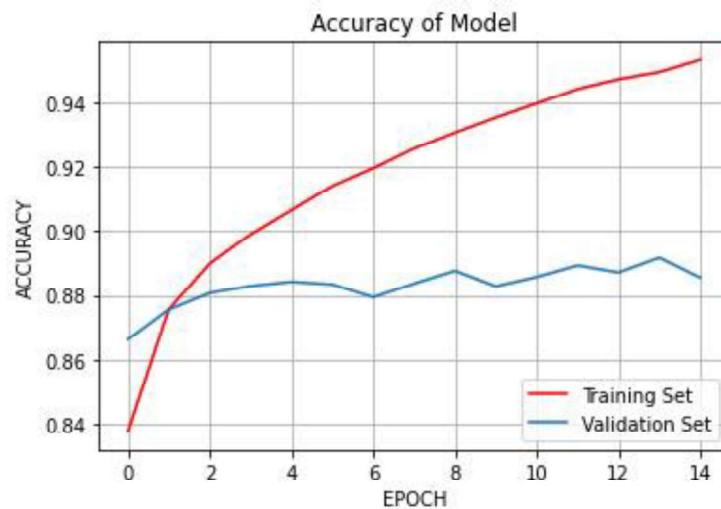
سپس یک شبکه `MLP` با دو لایه مخفی با تعداد نورون های لایه اول  $400$  و لایه دوم  $200$  را آموزش می دهیم. و معماری شبکه ساخته شده به صورت زیر داریم:

تابع فعال ساز  $2$  لایه اول `relu` و لایه خروجی را `softmax` loss function نیز انتخاب کردیم و `sparse_categorical_crossentropy` انتخاب کردیم. و تعداد ایپاک را  $15$  در نظر گرفتیم.

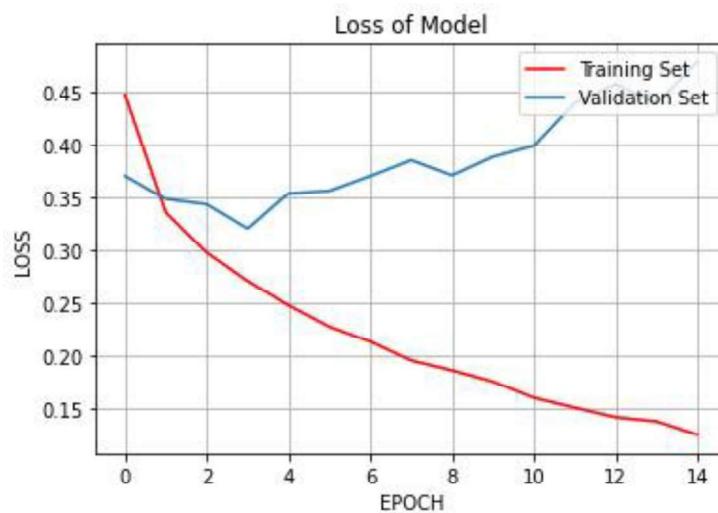
با استفاده از روش `Stochastic mini batch based` در هر قسمت نمودار های خطای خطا و دقت و ماتریس آشفتگی را بدست می آوریم:

**الف)** در شبکه از سه دسته با اندازه های  $32 \times 32 \times 3$  و  $256 \times 256 \times 3$  استفاده می کنیم و نتایج ما بصورت زیر می شود:

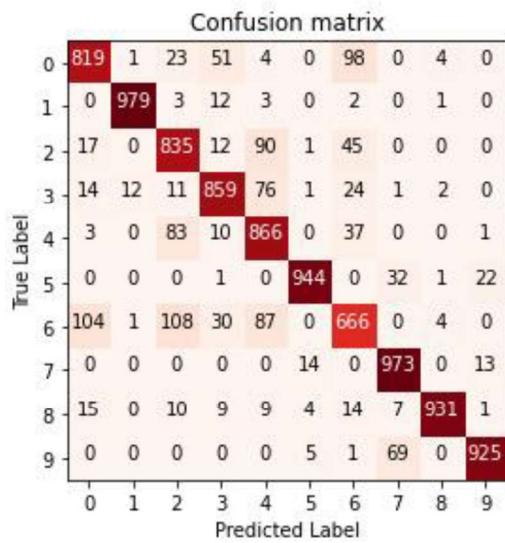
مدل ۱: اندازه دسته: ۳۲



شکل ۱ نمودار تغییرات صحت در هر ایپاک



شکل ۲ نمودار تغییرات خطای هر ایپاک

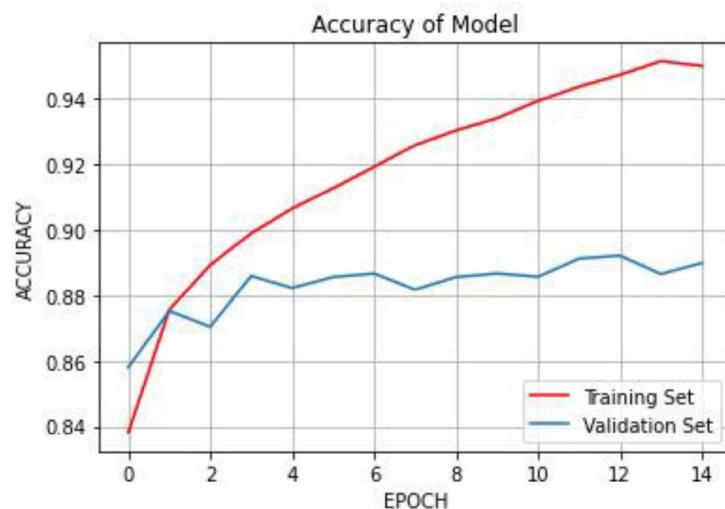


شکل ۳ ماتریس آشفتگی

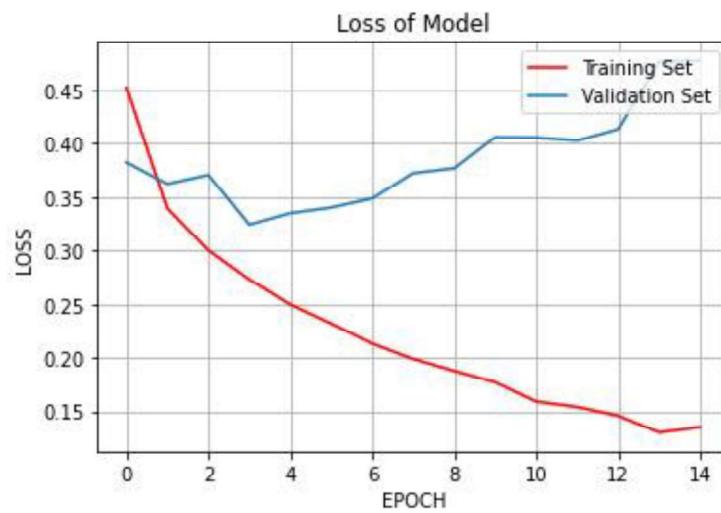
Test Accuracy , Test Loss مقدار Table 1

Test Loss	Test Accuracy
0.5072045922279358	0.8797000050544739

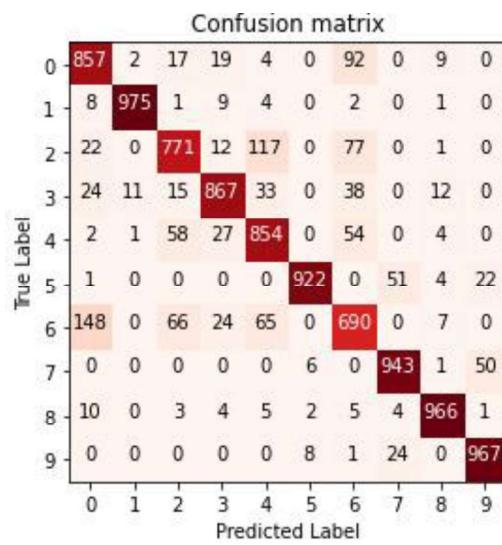
مدل ۲: اندازه دسته: ۶۴



شکل ۴ نمودار تغییرات صحت در هر ایپاک



شکل ۵ نمودار تغییرات دقت در هر اپاک

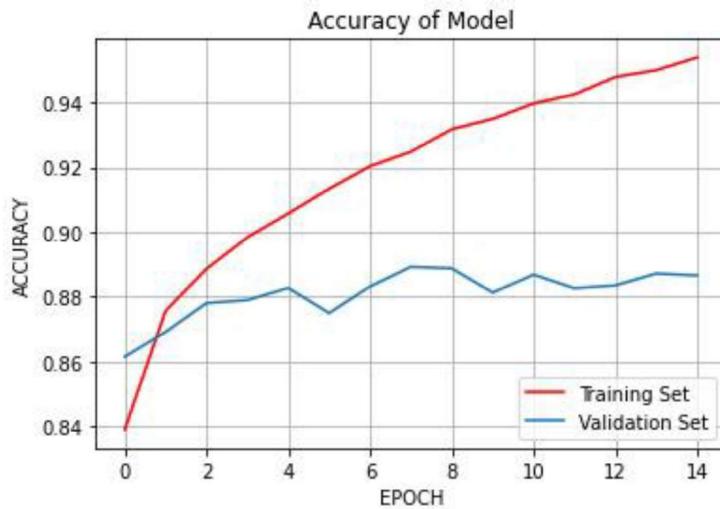


شکل ۶ ماتریس آشتفتگی

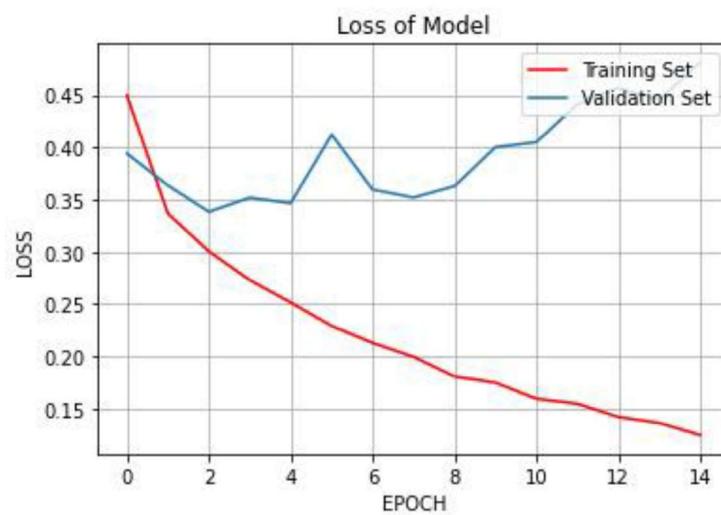
Test Accuracy , Test Loss مقدار Table 2

Test Loss	Test Accuracy
0.5329849123954773	0.8812000155448914

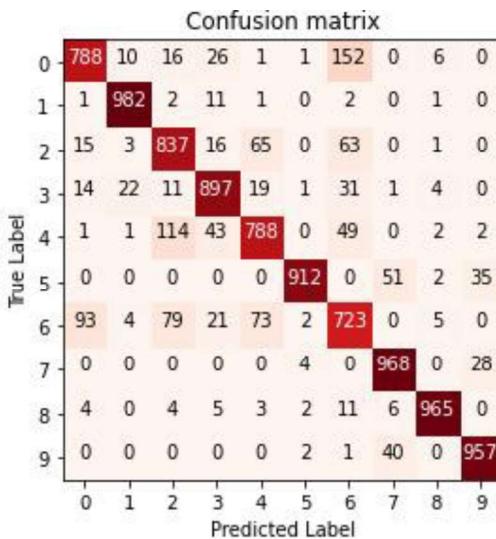
مدل ۳: اندازه دسته: ۲۵۶



شکل ۷ نمودار تغییرات صحت در هر ایپاک



شکل ۸ نمودار تغییرات خطا در هر ایپاک



شکل ۹ ماتریس آشتبگی

Test Accuracy , Test Loss مقادیر Table 3

Test Loss	Test Accuracy
0.5180811285972595	0.8816999793052673

نتایج کلی به شرح زیر است:

Test Accuracy و Test Loss مقادیر برای هر سه مدل Table 4

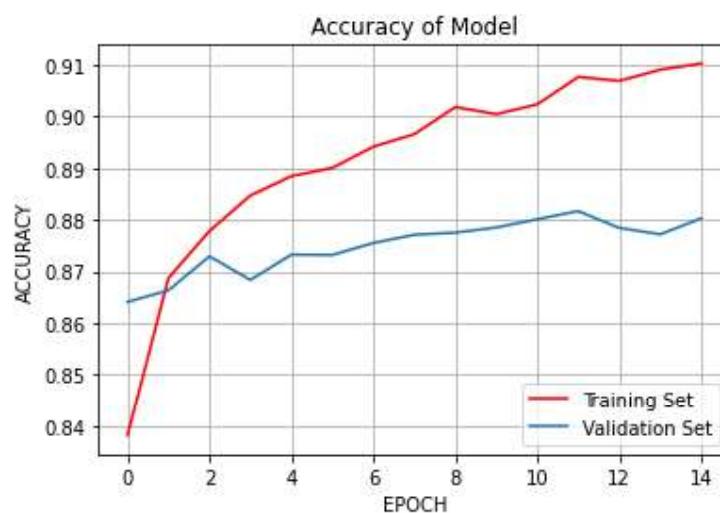
Batch size	Test Loss	Test Accuracy
32	0.5072045922279358	0.8797000050544739
64	0.5329849123954773	0.8812000155448914
256	0.5180811285972595	0.8816999793052673

همان طور که مشاهده می نماییم با افزایش سایز دسته (Batch Size) خطای مدل بر روی دادگان تست کاهش یافته، و صحت مدل بر روی این دادگان افزایش می یابد. البته باید نوجه کرد که هر اندازه سایز دسته افزایش یابد، نمونه های بیشتری در هر ایتریشن (iteration) مورد پردازش قرار می گیرند و طبیعتاً پیچیدگی حافظه افزایش پیدا می کند. لذا با در نظر گرفتن یک تریدآف بین این موضوع و

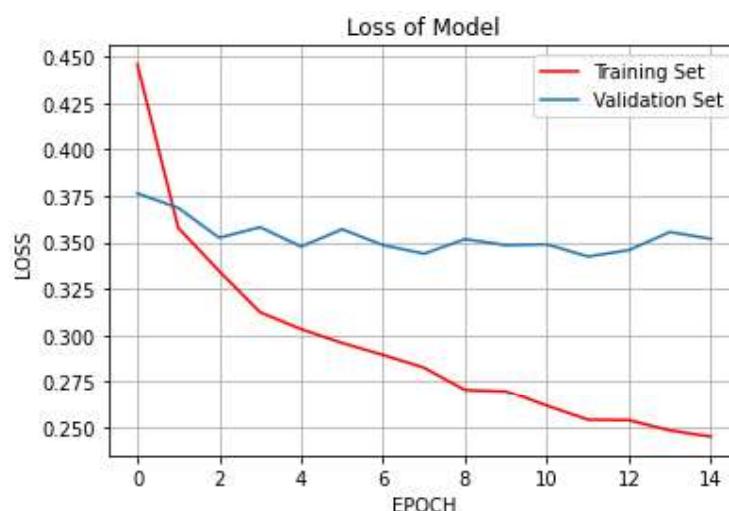
افزایش صحت (و نیز کاهش خطأ) و هم چنین با توجه به تعداد داده های موجود در دیتابست مقدار ۲۵۶ به عنوان سایز دسته عدد مناسبی به نظر می آید. نتیجتاً پاسخ به سوالات بعدی را با در نظر گرفتن سایز دسته فوق پیش می گیریم.

ب) در شبکه تابع فعالساز و خطأ را تغییر می دهیم که نتایج به شرح زیر است:

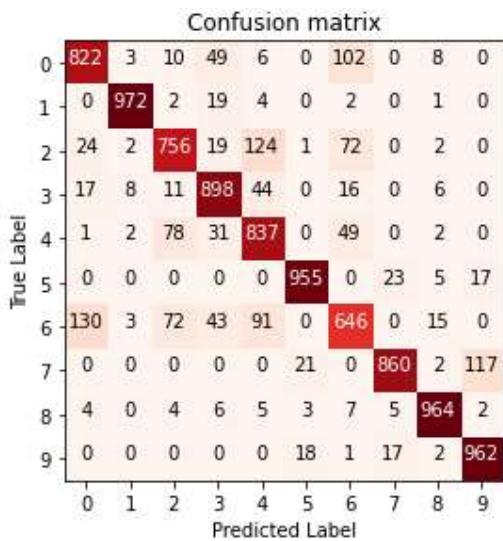
Loss function: categorical crossentropy , Activation function: tanh : مدل ۱ :



شکل ۱۰ نمودار تغییرات صحت در هر ایپاک



شکل ۱۱ نمودار تغییرات خطأ در هر ایپاک

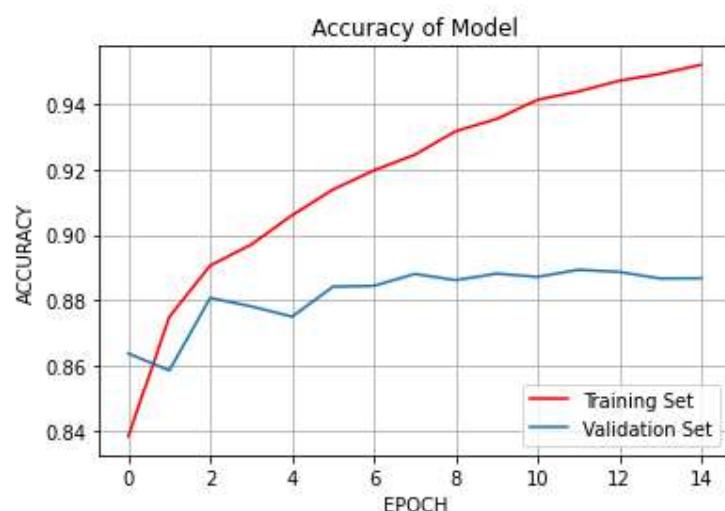


شکل ۱۲ ماتریس آشناگی

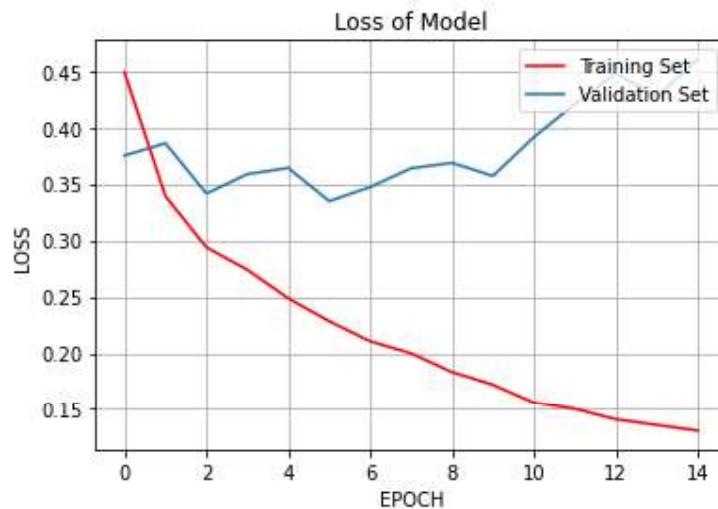
Test Accuracy , Test Loss مقدار Table 5

Test Loss	Test Accuracy
0.3845779001712799	0.8672000169754028

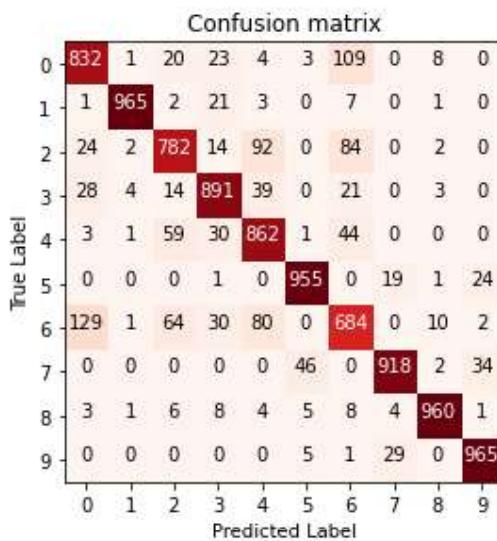
Loss function: categorical crossentropy , Activation function: relu :۲ مدل



شکل ۱۳ نمودار تغییرات صحت در هر ایپاک



شکل ۱۴ نمودار تغییرات خطا در هر ایپاک

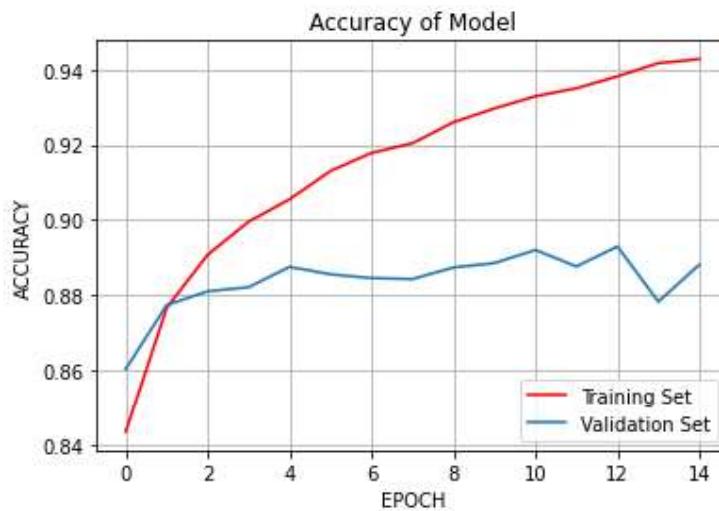


شکل ۱۵ ماتریس آشونگی

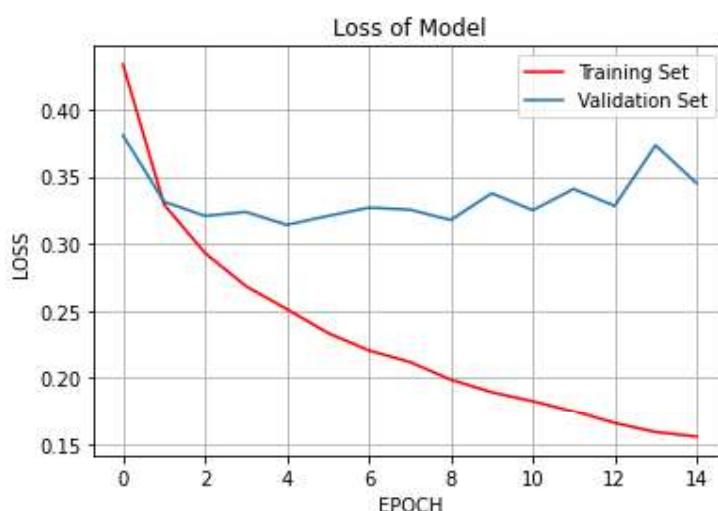
Test Accuracy , Test Loss مقدار Table 6

Test Loss	Test Accuracy
0.4935930073261261	0.8813999891281128

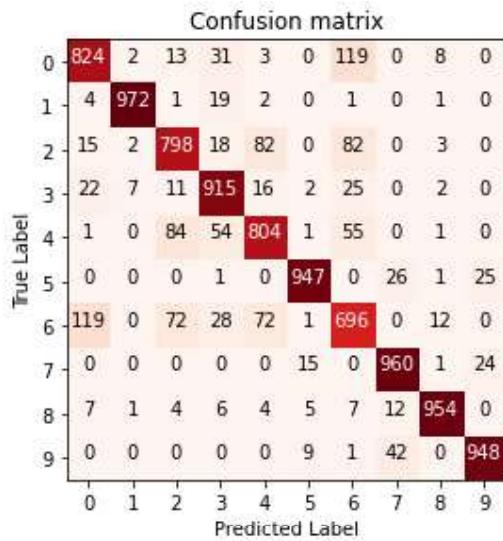
Loss function: categorical crossentropy , Activation function: sigmoid مدل ۳



شکل ۱۶ نمودار تغییرات صحت در هر ایپاک



شکل ۱۷ نمودار تغییرات خطأ در هر ایپاک

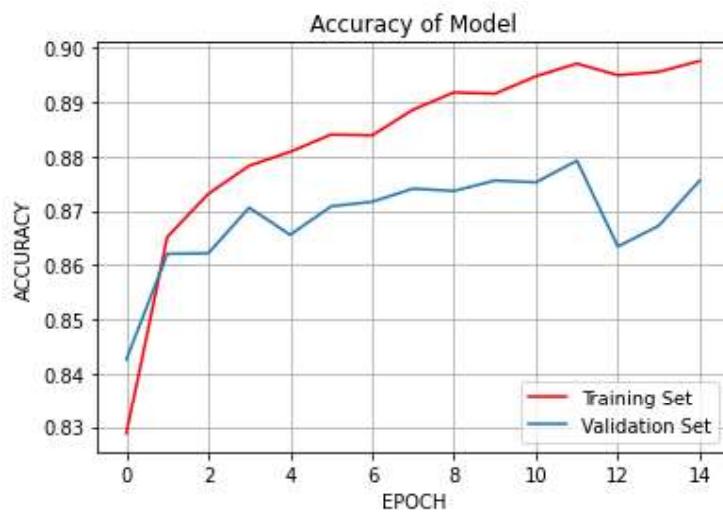


شكل ۱۸ ماتریس آشنازگی

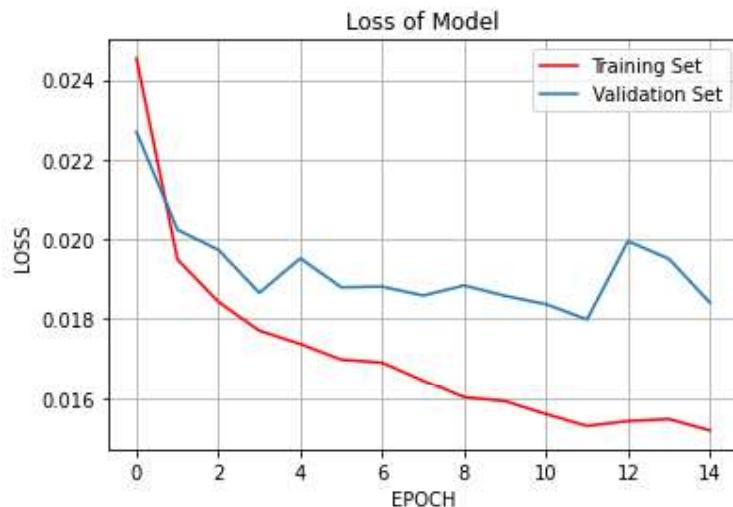
Test Accuracy , Test Loss مقادیر Table 7

Test Loss	Test Accuracy
0.3724557161331177	0.8817999958992004

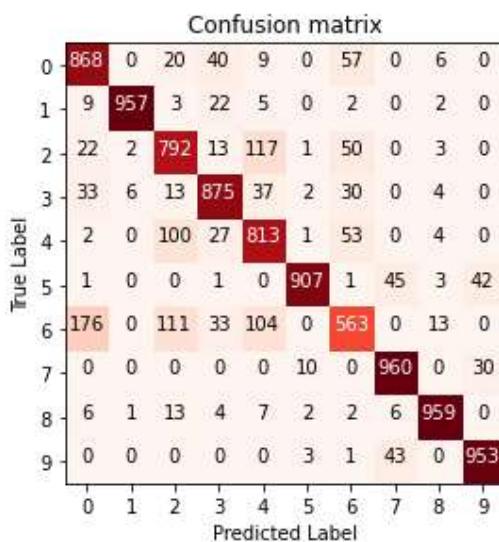
Loss function: mean squared error ، Activation function: tanh :۴ مدل



شكل ۱۹ نمودار تغییرات صحت در هر ایپاک



شکل ۲۰ نمودار تغییرات خطا در هر ایپاک

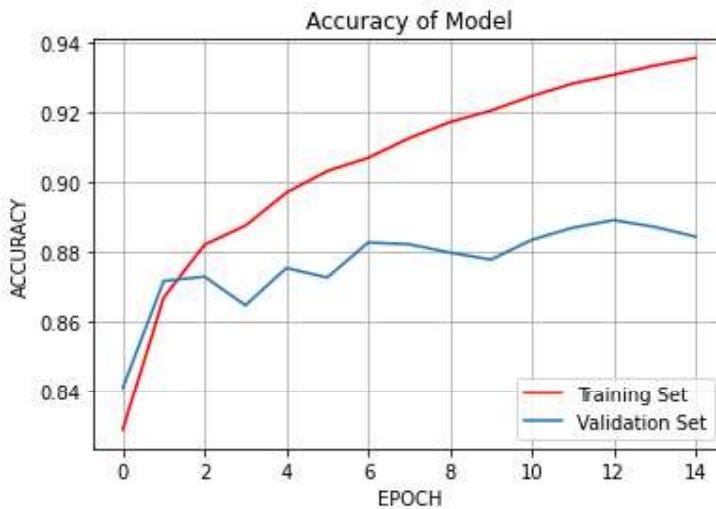


شکل ۲۱ ماتریس آشیتگی

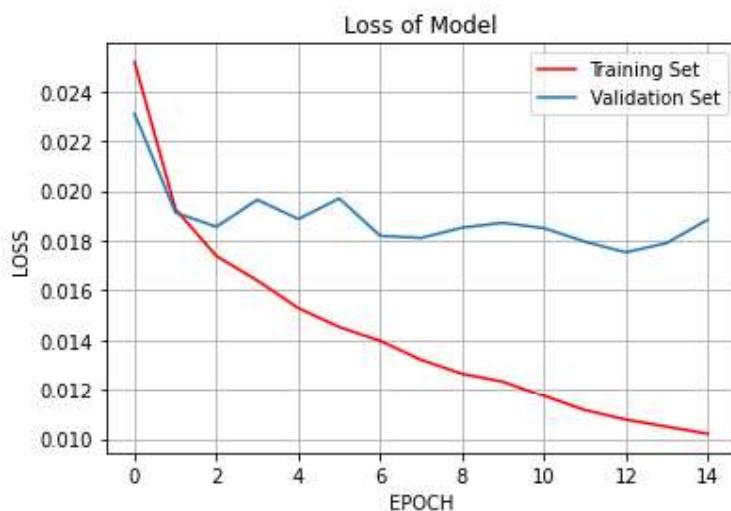
Test Accuracy , Test Loss مقدار Table 8

Test Loss	Test Accuracy
0.020205672830343246	0.8647000193595886

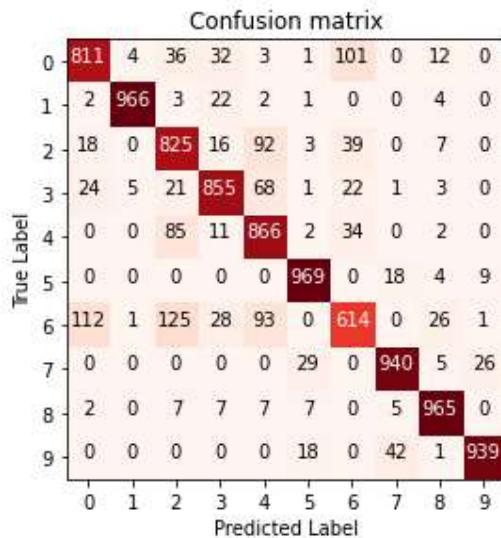
Loss function: mean squared error , Activation function: relu :۵ مدل



شکل ۲۲ نمودار تغییرات صحت در هر ایپاک



شکل ۲۳ نمودار تغییرات خطا در هر ایپاک

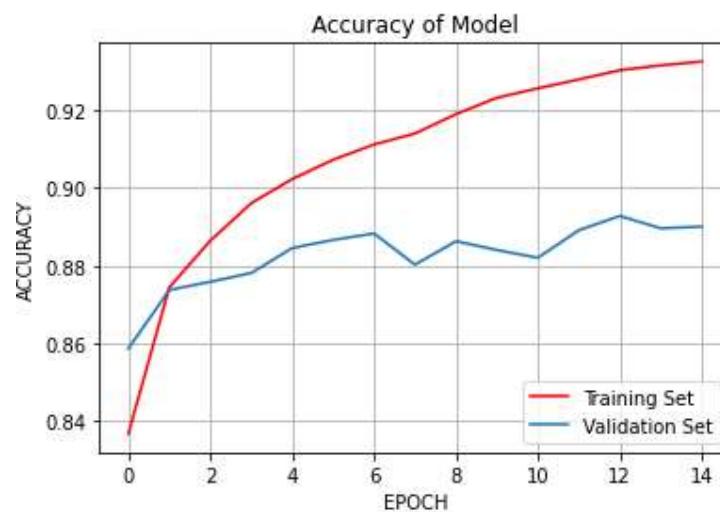


شکل ۲۴ ماتریس آشناستگی

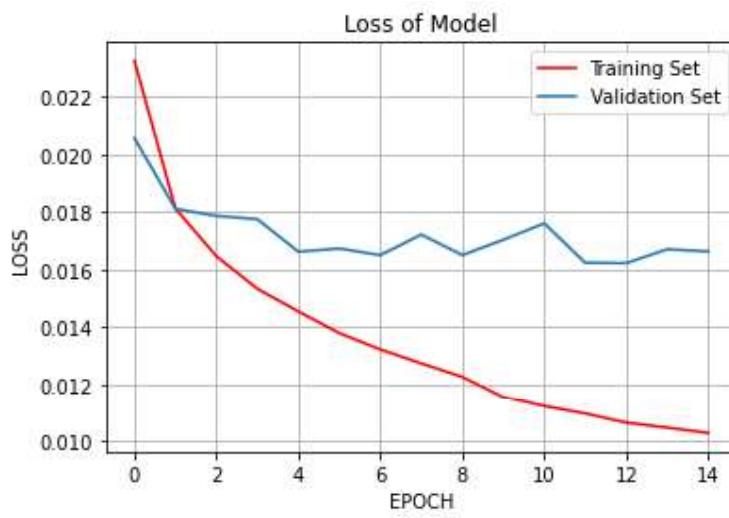
Test Accuracy , Test Loss مقدار Table 9

Test Loss	Test Accuracy
0.020624933764338493	0.875

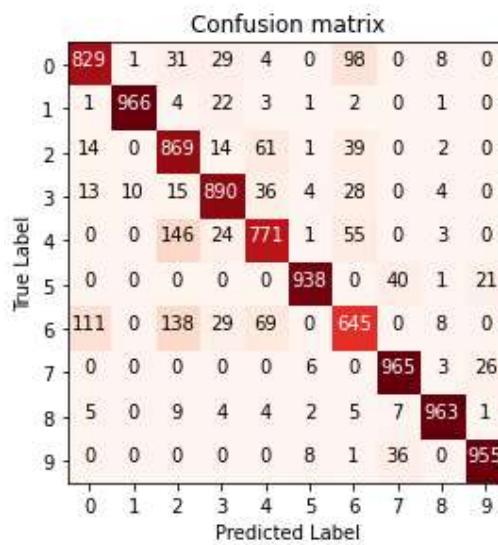
Loss function: mean squared error , Activation function: sigmoid :۶ مدل



شکل ۲۵ نمودار تغییرات صحت در هر اپاک



شکل ۲۶ نمودار تغییرات خطای در هر ایپاک



شکل ۲۷ ماتریس آشفتگی

Test Accuracy و Test Loss مقادیر Table 10

Test Loss	Test Accuracy
0.018216384574770927	0.8791000247001648

نتایج کلی به شرح زیر است:

برای هر ۶ مدل ذکر شده Train Time و Test Accuracy و Test Loss مقدار Table 11

Loss function	Activation function	Test Loss	Test Accuracy	Train Time
categorical crossentropy	tanh	0.3845779001712799	0.8672000169754028	142.3707778453827
categorical crossentropy	relu	0.4935930073261261	0.8813999891281128	121.80039548873901
categorical crossentropy	sigmoid	0.3724557161331177	0.8817999958992004	126.03720545768738
mean squared error	tanh	0.020205672830343246	0.8647000193595886	122.2554624080658
mean squared error	relu	0.020624933764338493	0.875	125.57075595855713
mean squared error	sigmoid	0.018216384574770927	0.8791000247001648	127.74426341056824

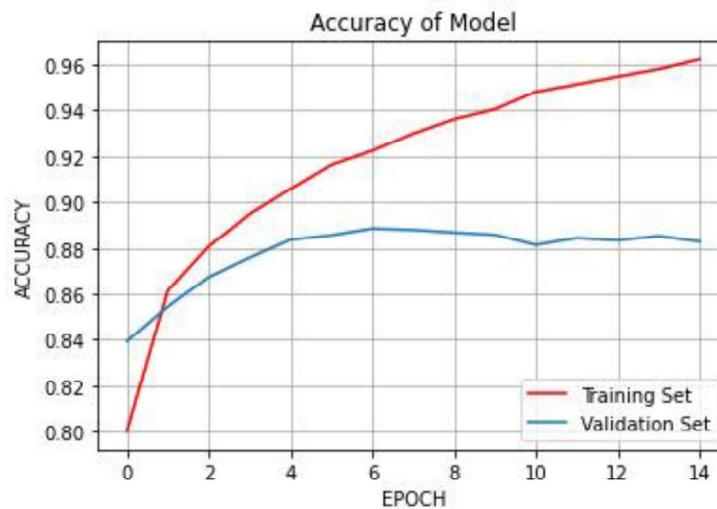
همانطور که مشاهده می‌نمایید، بهترین نتایج در مجموع برای Loss Function: Categorical Cross Activation Function: Sigmoid و Entropy حاصل شد. حال به بررسی نتایج می‌پردازیم. کاملا ملموس است که از نظر خطای تابع هزینه‌ی categorical cross entropy بیشتر از mean square error عمل می‌کند. با این حال خطای برای هر دو تابع خطای برای تابع فعالساز sigmoid کمترین مقدار را در گروه خود نشان می‌دهد. همچنین از نقطه نظر صحت نیز، این موضوع برقرار است. در کل این تابع در هنگام همگرایی، گرادیان اسموئی از خود نشان می‌دهد. و نیز یکی بهترین توابع نرمالایزد شده است. در نهایت ما با توجه به صحت و نیز زمان آموزش نتیجه‌ی فوق را به عنوان مدل برنده ارائه کردیم هر چند اگر میزان loss را در نظر بگیریم، تابع mean square error گزینه‌ی بهتری است.

ج) با استفاده از بهترین شبکه بخش قبل که همان مدل ۳ بخش ب بود یعنی:

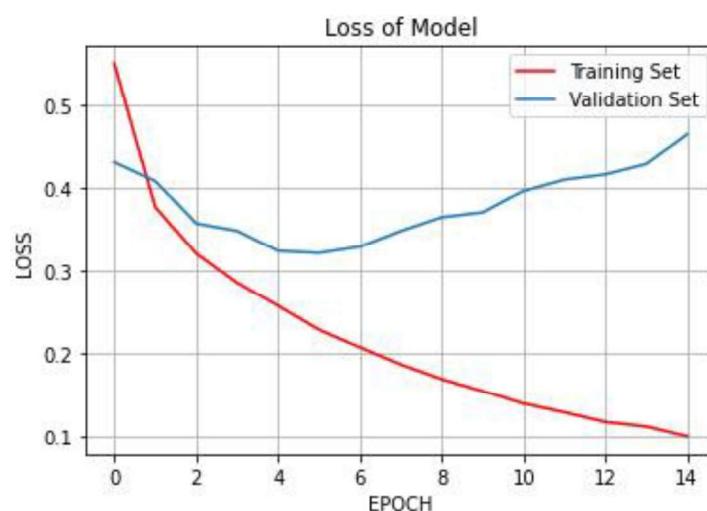
Loss function: categorical crossentropy , Activation function: sigmoid

به شبکه تعداد ۳ لایه کانولوشنی با کرنل سایز (3,3) و strides برابر با ۲ اضافه کردیم.

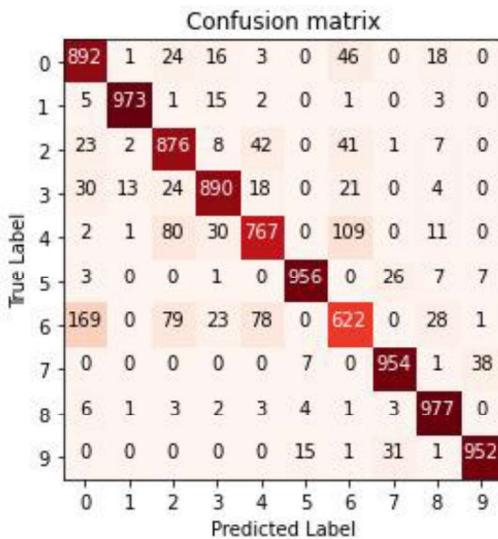
نتایج بصورت زیر شد:



شکل ۲۸ نمودار تغییرات صحت در هر اپاک



شکل ۲۹ نمودار تغییرات خطأ در هر اپاک



شکل ۳۰ ماتریس آشیانه

نتایج خطأ و دقت بصورت زیر شد:

Test Accuracy و Test Loss مقادیر Table 12

Test Loss	Test Accuracy	Train Time
0.46549493074417114	0.8859000205993652	370.61479210853577

و در مقایسه با حالت بدون CNN داریم:

CNN و با CNN برای دو مدل بدون Train Time و Test Accuracy و Test Loss مقادیر Table 13

Model	Test Loss	Test Accuracy	Train Time
With CNN	0.46549493074417114	0.8859000205993652	370.61479210853577
Without CNN	0.3724557161331177	0.8817999958992004	126.03720545768738

همانطور که از نتایج بدست آمده مشاهده می‌نمایید، صحت تشخیص افزایش یافته و این قابلیت اضافه کردن لایه‌های کانولوشن را در افزایش صحت تشخیص نشان می‌دهد. در واقع این نتایج برآمده از دو مزیت مهم این ساختار است. اولین مزیت parameter sharing و دومین مزیت sparsity of connections می‌باشد. در واقع این ساختار اتصالات بیهوده‌ای که در MLP گاها مشاهده می‌کردیم، حذف شده و بحث

فیلترینگ و استخراج ویژگی‌ها مطرح می‌شود. لذا با استخراج اطلاعات مفید از طریق استخراج ویژگی‌ها به ساختار فوق و با صحتی بالاتر باید می‌رسیدیم که بدین گونه نیز بود. متأسفانه مدل از نظر زمانی بدتر از ساختارهای قبلی عمل نموده که بواسطه‌ی اضافه نمودن سه لایه‌ی کانولوشن جدید است و از نظر loss نیز نتایج بدست آمده با اختلاف حدودی  $0.09$  مشابه با نتایج قبل است.

---

## ۵) عملکرد لایه‌های Pooling و Batch normalization

### Batch normalization

فرآیندی که ورودی‌های یک لایه را برای هر مینی بچ (mini-batch) استاندارد می‌کند و برای سریع تر و پایدارتر کردن (stabilizing) شبکه‌های عصبی از طریق افزودن لایه‌های اضافی در یک شبکه عصبی عمیق است. لایه جدید عملیات استانداردسازی (standardizing) و نرمال‌سازی (normalizing) را روی ورودی لایه‌ای که از لایه قبلی می‌آید انجام می‌دهد. و فرآیند نرمال‌سازی (normalizing) در نرمال‌سازی دسته‌ای به صورت دسته‌ای انجام می‌شود، نه به عنوان یک ورودی. که باعث کاهش چشمگیر تعداد ایپاک (epoch)‌های آموزشی مورد نیاز برای آموزش شبکه‌های عمیق می‌شود.

نرمال‌سازی دسته‌ای را درست بعد از هر کانولوشن و قبل از فعال‌سازی (activation) می‌توان اعمال کرد.

### Pooling

افزودن یک لایه Pooling پس از لایه کانولوشنال یک الگوی رایج است که برای مرتب کردن لایه‌ها در یک شبکه عصبی کانولوشن استفاده می‌شود که ممکن است یک یا چند بار در یک مدل مشخص تکرار شود.

لایه Pooling بر روی هر feature map به طور جداگانه عمل می‌کند تا مجموعه جدیدی از همان تعداد Pooled feature map ایجاد کند.

شامل انتخاب یک عملیات ادغام، بسیار شبیه filter است که برای feature map ها اعمال می‌شود. اندازه عملیات Pooling یا filter کوچکتر از اندازه feature map است. به طور خاص، تقریباً همیشه  $2 \times 2$  پیکسل با گام(stride)  $2$  پیکسل اعمال می‌شود.

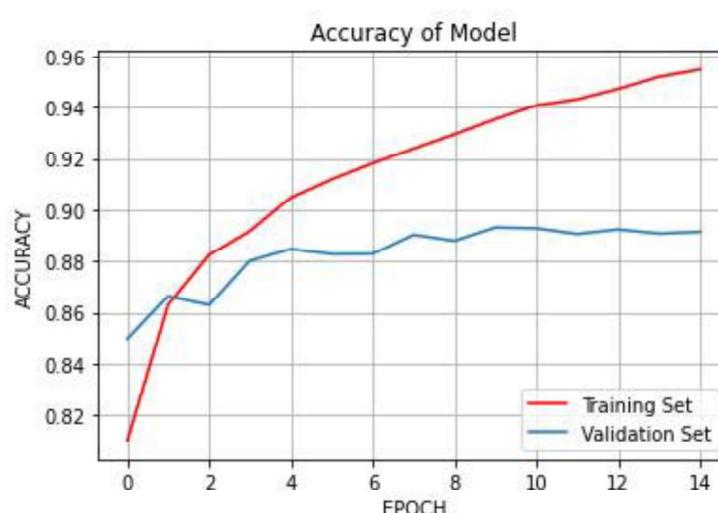
این بدان معنی است که لایه Pooling همیشه سایز هر feature map را با فاکتور ۲ کاهش می دهد، به عنوان مثال هر بعد نصف می شود و تعداد پیکسل ها یا مقادیر در هر feature map به یک چهارم اندازه کاهش می یابد. به عنوان مثال، یک لایه Pooling اعمال شده بر روی feature map از  $6 \times 6$  (۳۶ پیکسل) منجر به یک خروجی Pooled feature map از  $3 \times 3$  (۹ پیکسل) می شود.

در عمل این عملیات باعث کاهش ابعاد و کاهش پیچیدگی شبکه و در نهایت افزایش سرعت می شود. که این نیز باعث کاهش زمان آموزش نیز می شود.

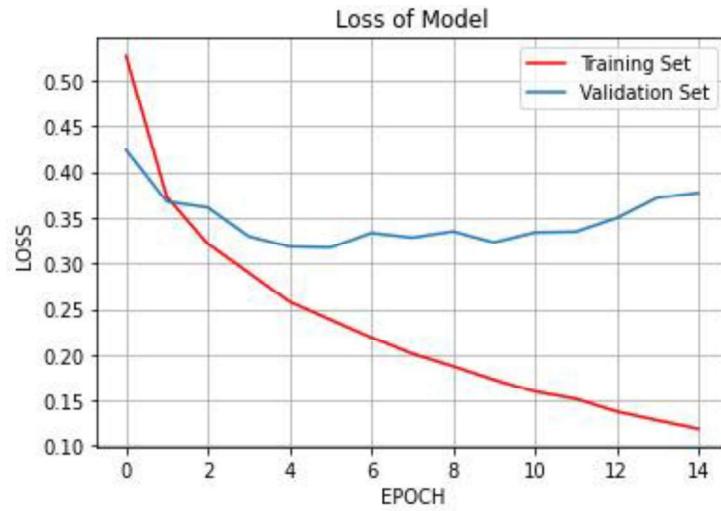
عملیات Pooling به جای آموختن (Learned) مشخص شده است. دو عملکرد رایج مورد استفاده در عملیات Pooling عبارتند از:

میانگین مقدار هر Patch را در feature map محاسبه می کند. Average Pooling  
حداکثر مقدار را برای هر Patch از feature map محاسبه می کند. Maximum Pooling (Max Pooling)

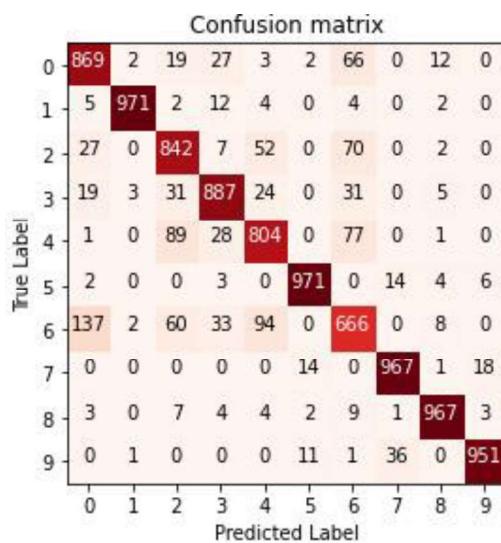
با اضافه کردن لایه های Pooling و Batch normalization نتایج بصورت زیر می شود:



شکل ۳۱ نمودار تغییرات صحت در هر ایپاک



شکل ۳۲ نمودار تغییرات خطا در هر ایپاک



شکل ۳۳ ماتریس آشناگی

Train Time , Test Accuracy , Test Loss مقادیر Table 14

Test Loss	Test Accuracy	Train Time
-----------	---------------	------------

0.3806668817996979

0.8895000219345093

333.26642203330994

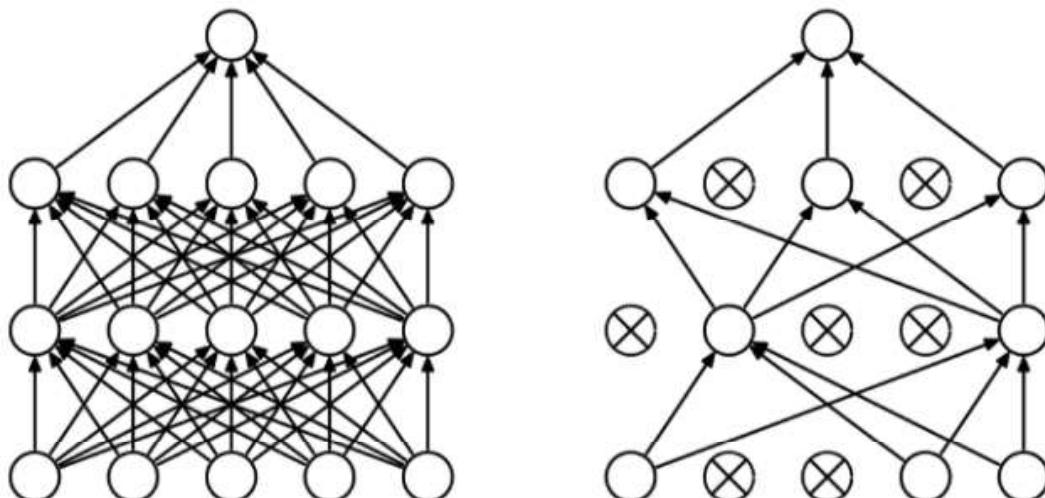
نتایج کلی به شرح زیر است:

Batch Normalization و Pooling و CNN برای مدل با Train Time و Test Accuracy و Test Loss مقادیر Table 15

Model	Test Loss	Test Accuracy	Train Time
With CNN	0.46549493074417114	0.8859000205993652	370.61479210853577
With CNN & Pooling & Batch Normalization	0.3806668817996979	0.8895000219345093	333.26642203330994

همانطور که مشاهده می‌نمایید، نتایج صحت و خطای مدل برای داده‌گان تست با اضافه کردن batch normalization و pooling بهبود یافته به گونه‌ای که شاهد کاهش ۱۰٪ خطا و نیز شاهد افزایش ۵٪ درصدی در صحت مدل بر روی داده‌گان تست هستیم. حال به بررسی می‌پردازیم. مسلماً با اضافه کردن لایه‌ی pooling باید کاهش ابعاد نمونه‌ها را پیذیریم در ازای آنکه زمان آموزش به مقدار قابل ملموسی کاهش یابد و لذا در اینجا کاهش پیچیدگی زمانی را شاهد هستیم. از طرفی با batch normalization یک عملیات نرمالیزیشن را روی مجموعه داده‌گان ورودی در یک دسته را اعمال می‌کنیم که منجر به افزایش صحت و کاهش خطا می‌شود و نتیجه به صورت فوق یعنی یک مدل بهبود یافته بدست می‌آید.

۵) کاربرد های لایه Dropout: یک تکنیک است که جلوی «over-fit» شدن شبکه را می‌گیرد. یعنی یک راه ساده برای جلوگیری از اتصال بیش از حد در شبکه‌های عصبی استفاده از dropout است این روش



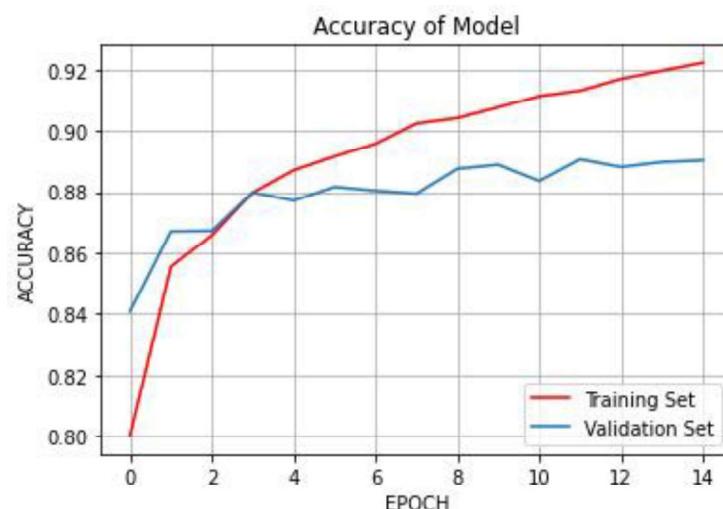
شکل ۲۴ مثالی از اضافه کردن لایه Dropout

روشی برای منظم سازی است و خطای تعمیم پذیری را با کاهش گنجایش مدل کمتر می کند. بدین ترتیب در هر دور آموزشی، به جای استفاده از همه نورون ها، تنها برخی از نورون ها (با احتمال  $p$ ) فعال می شوند.

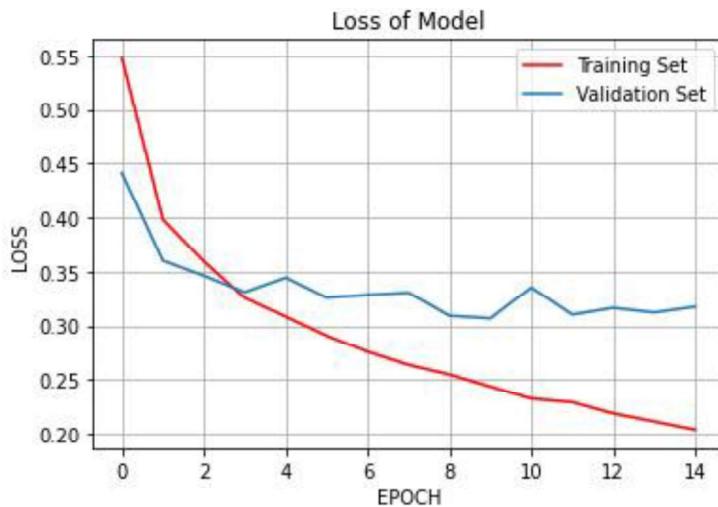
منظم سازی (regularization): مقدار خطای که شبکه های عصبی روی داده های آموزشی بدست می آورند بسیار کمتر از خطای آنها روی داده های آزمایشی است. به همین دلیل در معرض بیش برآذش قرار می گیرند. برای مقابله با این مساله، از تکنیک های منظم سازی استفاده می شود که گنجایش مدل را کاهش دهیم. بدین ترتیب تفاوت بین خطای آموزش و آزمایش کاهش یافته و مدل آموزش و آزمایش، عملکرد مشابهی از خود نشان خواهد داد.

حال با اضافه کردن Dropout به شبکه داریم:

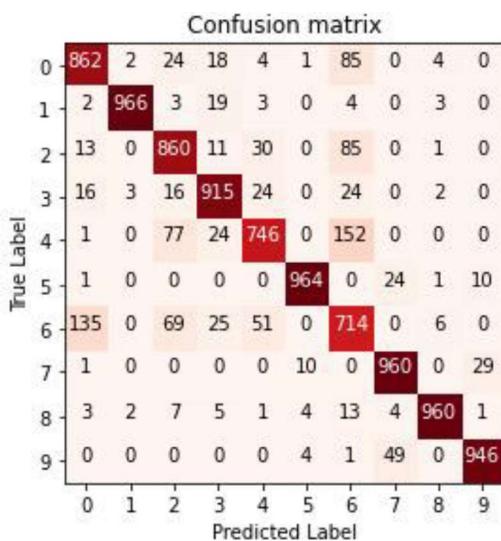
مدل ۱: اضافه کردن Dropout با درصد ۰,۰۵



شکل ۳۵ نمودار تغییرات صحت در هر ایپاک



شکل ۳۶ نمودار تغییرات خطا در هر اپاک

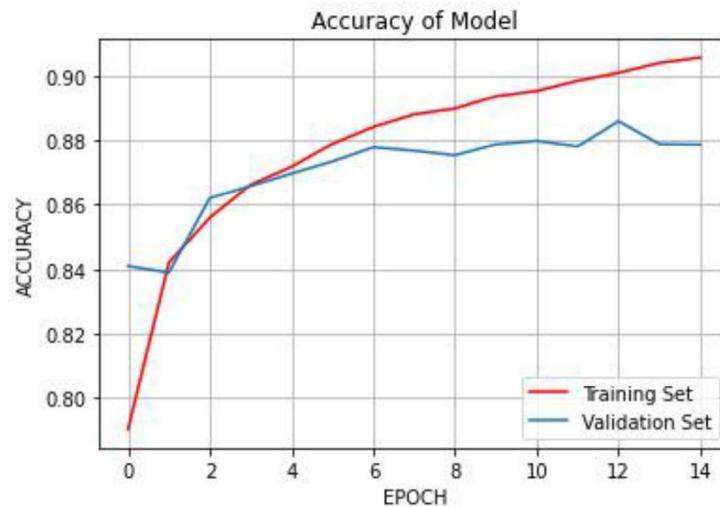


شکل ۳۷ ماتریس آشتقی

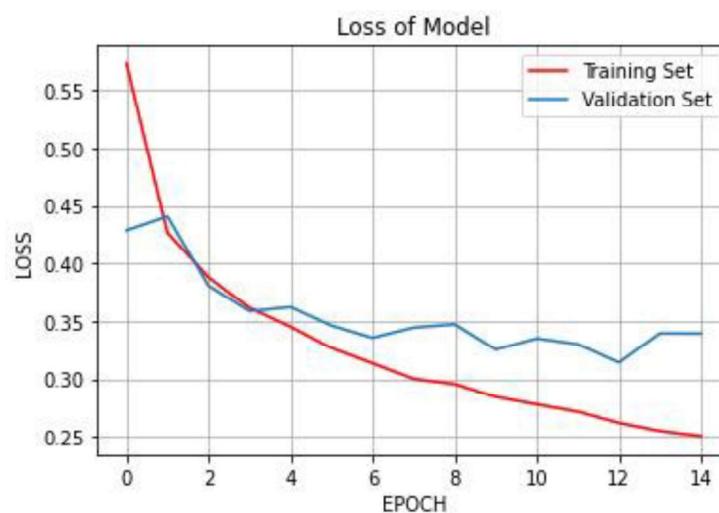
Test Accuracy و Test Loss مقادیر Table 16

Test Loss	Test Accuracy
0.3219941556453705	0.8892999887466431

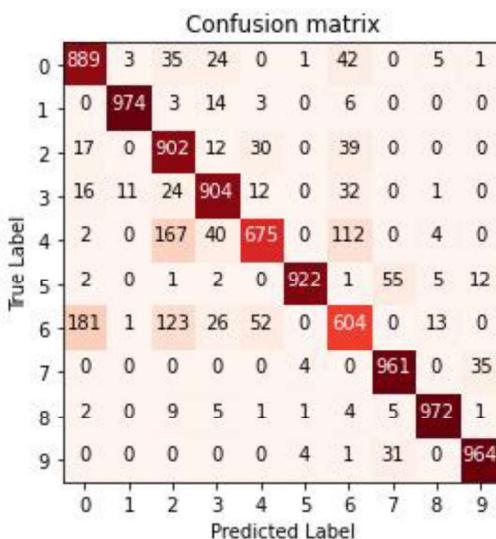
مدل ۲: اضافه کردن Dropout با درصد ۱۰



شکل ۳۸ نمودار تغییرات صحت در هر ایپاک



شکل ۳۹ نمودار تغییرات خطا در هر ایپاک

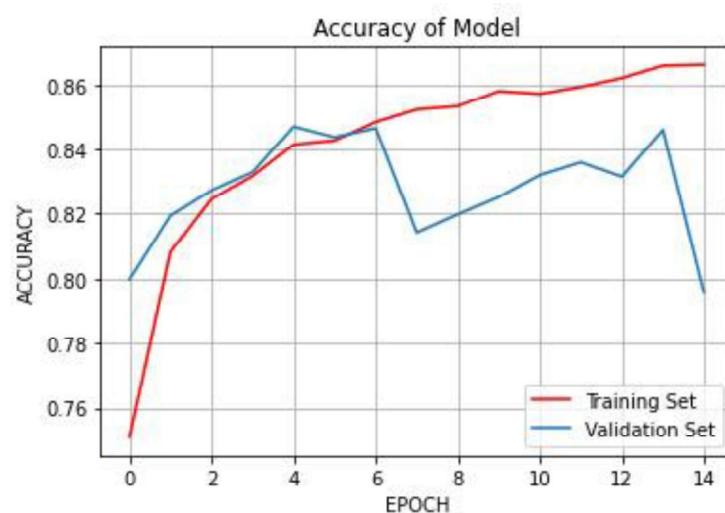


شکل ۴۰ ماتریس آشوبگی

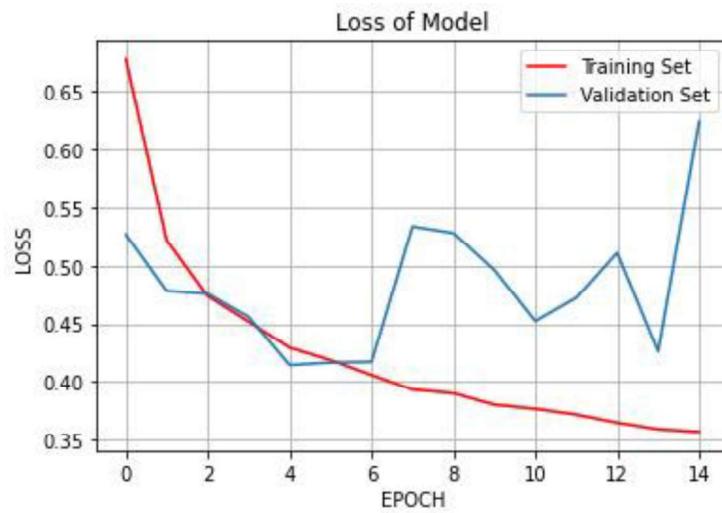
Test Accuracy و Test Loss مقادیر Table 17

Test Loss	Test Accuracy
0.35648494958877563	0.8766999840736389

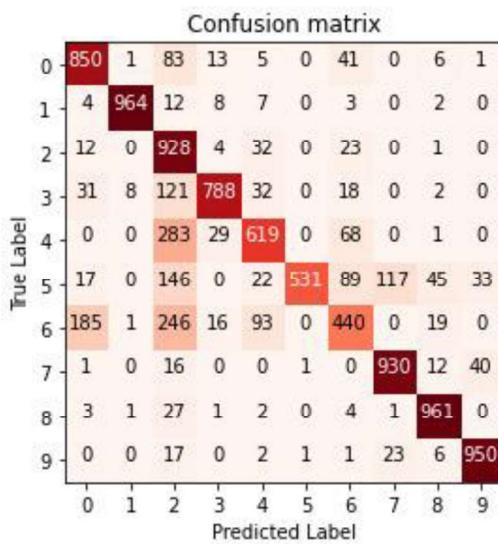
مدل ۳: اضافه کردن Dropout با درصد ۳۰٪.



شکل ۴۱ نمودار تغییرات صحت در هر اپیاک



شکل ۴۲ نمودار تغییرات خطا در هر اپیک



شکل ۴۳ ماتریس آشتفتگی

Test Accuracy , Test Loss مقادیر Table 18

Test Loss	Test Accuracy
0.6284169554710388	0.7961000204086304

نتایج کلی به شرح زیر است:

Dropout برای ۳ مدل با درصد های متفاوت Train Time , Test Accuracy , Test Loss مقادیر Table 19

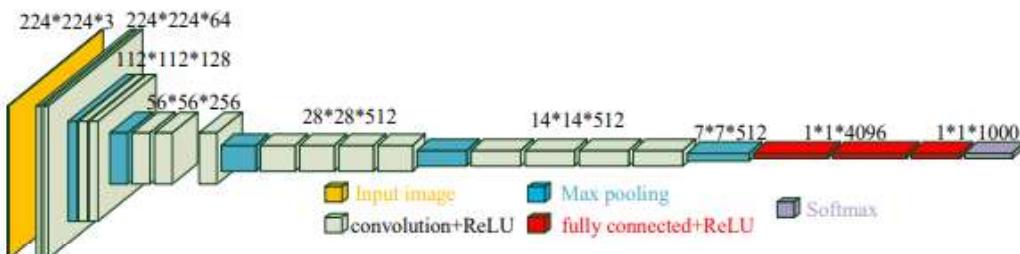
Dropout	Test Loss	Test Accuracy	Train Time
0.05	0.3219941556453705	0.8892999887466431	338.70936727523804
0.10	0.35648494958877563	0.8766999840736389	343.24688482284546
0.30	0.6284169554710388	0.7961000204086304	382.671954870224

در رابطه با نتایج این قسمت باید نتیجه بدست آمده در قسمت قبل را هم لحاظ کنیم. با اضافه نمودن dropout با مقدار ۰،۰۵ کاملا مشهود است که از نظر زمانی و صحت نتایجی مشابهی با مدل برنده دارد ولی مزیت آن کاهش خطا (Loss) به میزان ۶،۰ می باشد. ولی ما این روند را همچنان ادامه دادیم تا تاثیر بزرگ کردن dropout را مشاهده کنیم. نتایج نشان می دهد که dropout باید مقادیر کوچکی در این ساختارها داشته باشد تا کلیت ساختار شبکه را تحت الشاعع قرار ندهد. مسلما همینکه در حدود ۰،۱ یا ۰،۳ این مقدار را انتخاب کنیم تا شاهد حذف ۱۰ یا ۳۰ درصد از نورون های لایه به صورت تصادفی باشیم، کفایت می کند که شبکه بر روی داده گان آموزش over fit نشود ولی اینکه کلیت ساختار را با زیاد نمودن dropout از بین ببریم طبیعتا کار درستی نیست و شبکه در هر ایتریشن در حال اموزش قسمتی از وزن ها می شود و نتیجه ای آن تغییر زیاد وزن ها در هر آپدیت و آموزش غلط شبکه است. لذا برای ساختار شبکه ای که در این مساله داشتیم، مقدار dropout برابر با ۰،۰۵ یا ۰،۱ به بهبودی تشخیص توسط مدل می پردازد.

## سوال ۲ – Transfer Learning

با توجه به اینکه رقم آخر شماره دانشجویی هر دوی ما ۶ می باشد مجموع آنها ۱۲ می باشد، پس مدل انتخابی ما مدل شماره ۲ یعنی مدل VGG-19 می شود.

الف) ساختار مدل VGG-19 در شکل زیر نشان داده شده است:



شکل ۴۴ ساختار مدل VGG-19

اندازه هسته کانولوشن  $3 \times 3$  و اندازه ورودی  $224 \times 224 \times 3$  است. که تصویر RGB به عنوان ورودی به این شبکه داده شد. پیش پردازشی که انجام شد این بود که میانگین مقدار RGB را از هر پیکسل کم کرده که در کل مجموعه آموزشی محاسبه شد.

هسته های استفاده شده در اندازه  $(3 \times 3)$  با اندازه گام ۱ پیکسل، آنها را قادر می سازد تا کل مفهوم تصویر را پوشش دهند.

از spatial padding برای حفظ وضوح فضایی تصویر استفاده شد.

Maxpooling بر روی یک پنجره  $2 \times 2$  پیکسل با اندازه گام ۲ انجام شد.

سه لایه fully connected را پیاده سازی کرد که دو لایه اول با سایز  $4096$  و بعد از آن یک لایه با  $1000$  کanal برای طبقه بندی ILSVRC و لایه نهایی یکتابع softmax است.

بنابراین به زبان ساده VGG یک CNN عمیق است که برای طبقه بندی تصاویر استفاده می شود. این لایه ها همان طور که از نامش پیداست شامل  $19$  لایه CNN و  $3$  لایه fully connected است که این لایه ها در مدل VGG19 به شرح زیر است:

- Conv3x3 (64)
- Conv3x3 (64)
- MaxPool
- Conv3x3 (128)
- Conv3x3 (128)
- MaxPool
- Conv3x3 (256)
- Conv3x3 (256)
- Conv3x3 (256)
- Conv3x3 (256)
- MaxPool
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- MaxPool
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- MaxPool
- Fully Connected (4096)
- Fully Connected (4096)
- Fully Connected (1000)
- SoftMax

VGG-19 در مقایسه با شبکه‌های عصبی کانولوشنال سنتی، از نظر عمق شبکه بهبود یافته است. از ساختار متناوب چندین لایه کانولوشن و لایه‌های فعال‌سازی غیرخطی استفاده می‌کند که بهتر از یک downsampling برای Maxpooling و از  $relu$  را به عنوان تابع فعال‌ساز در نظر می‌گیرد، و بزرگترین مقدار در ناحیه تصویر را به عنوان مقدار  $down\left(\chi_j^{(n-1)}\right)$  عمده برای بهبود توانایی ضداعوجاج شبکه به تصویر استفاده می‌شود، در حالی که ویژگی‌های اصلی نمونه را حفظ می‌کند و تعداد پارامترها را کاهش می‌دهد. از لایه طبقه‌بندی نهایی که دارای فعال‌سازی softmax است برای پیش‌بینی احتمالات هر کلاس استفاده می‌شود.

$down\left(\chi_j^{(n-1)}\right)$ : the maximum pooling sample function

$\tau_j^n$  : the coefficient corresponding to the  $j$ -th feature map of the  $n$ -th layer

$f\left(\tau_j^n \ down\left(\chi_j^{(n-1)}\right) + b_j^{(n)}\right)$ : the ReLu activation function

$$\chi_{p_j}^{(n)} = f\left(\tau_j^n \ down\left(\chi_j^{(n-1)}\right) + b_j^{(n)}\right)$$

#### مزایا:

- به عنوان یک معماری طبقه‌بندی خوب برای بسیاری از مجموعه داده‌ها استفاده می‌شود و همانطور که نویسنده‌گان مدل‌ها را در دسترس عموم قرار دادند، می‌توان آنها را همانطور که هست یا با اصلاح برای کارهای مشابه دیگر استفاده کرد.
- یادگیری انتقالی دارد، یعنی می‌تواند برای کارهای تشخیص چهره نیز استفاده شود. به راحتی با فریمورک‌های دیگری مانند keras در دسترس هستند، بنابراین می‌توان آنها را اصلاح کرد و هر طور که می‌خواهیم استفاده کرد.
- برای ساخت Neural Art مورد استفاده است.
- توانایی استخراج ویژگی خوبی دارد.

#### معایب:

- تعداد بسیار زیادی پارامتر دارد.
- مدل بسیار سنگینی است.
- پر هزینه است.

- پیچیدگی زمانی بالا در آموزش دارد.
- زمان آموزش زیاد است.
- مشکل Vanishing Gradient دارد.

ب) سه عکس بصورت زیر به شبکه داده شد و نتایج بصورت زیر است:



```

1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('gdrive/MyDrive/Colab Notebooks/lion.jpg', target_size=(224, 224))
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print('%s (%.2f%%)' % (label[1], label[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly remount,
lion (100.00%)

```

شکل ۴۵ ورودی و خروجی مدل VGG-19 برای تصویر شیر

خروجی شبکه:

Lion (100.00%)



```

1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(224, 224))
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print('%s (%.2f%%)' % (label[1], label[2]*100))

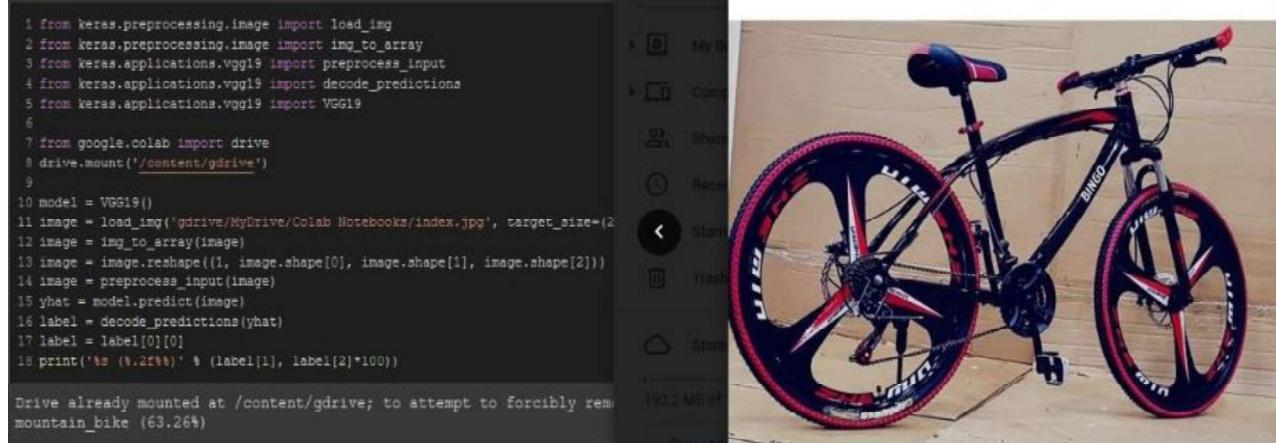
Drive already mounted at /content/gdrive; to attempt to forcibly remount,
snowplow (67.69%)

```

شکل ۴۶ ورودی و خروجی مدل VGG-19 برای تصویر برف روب

خروجی شبکه:

Snowplow (67.69%)



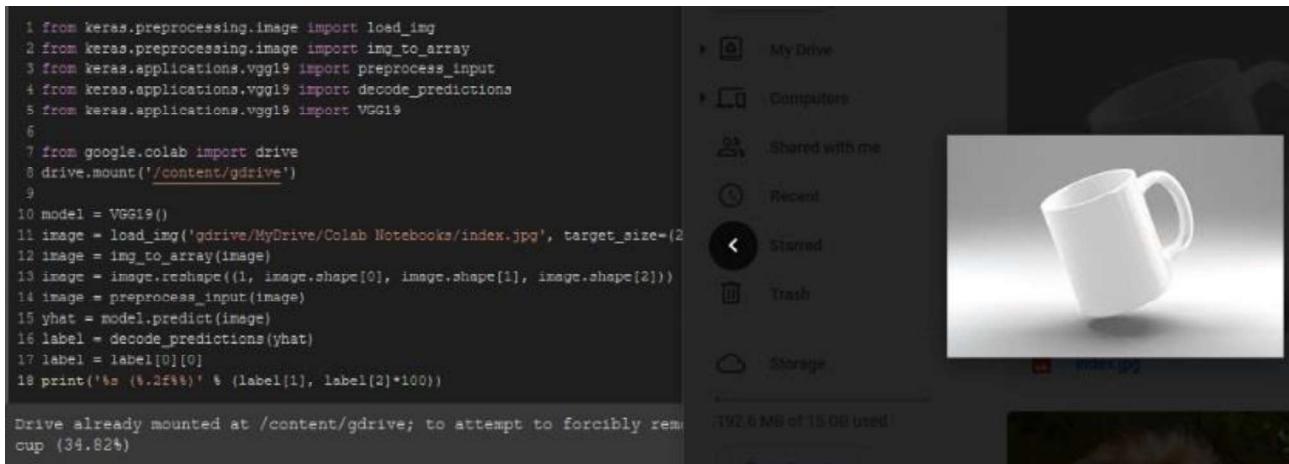
```
1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(224, 224))
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print('%s (%.2f%%)' % (label[1], label[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly remount, unmount first. 192.2 MB of 15.00 used
mountain_bike (63.26%)
```

شکل ۴۷ ورودی و خروجی مدل VGG-19 برای تصویر دوچرخه

خروجی شبکه:

Mountain\_bike (63.26%)



```
1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(224, 224))
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print('%s (%.2f%%)' % (label[1], label[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly remount, unmount first. 192.6 MB of 15.00 used
cup (34.82%)
```

شکل ۴۸ ورودی و خروجی مدل VGG-19 برای تصویر لیوان

خروجی شبکه:

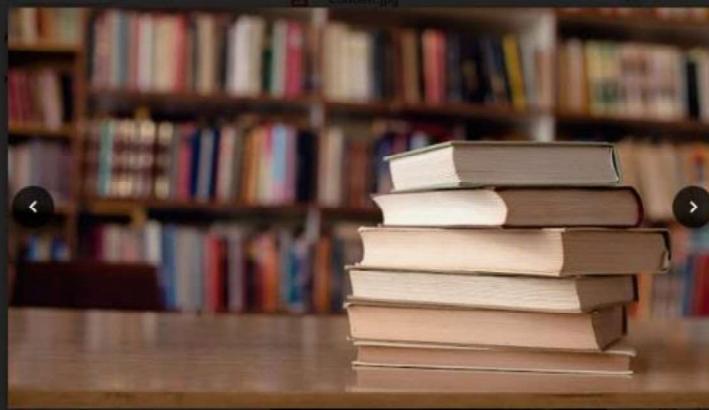
Cup (38.82%)

```

1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(2
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print("%s (%.2f%%)" % (label[1], label[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly rem
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_
bookshop (69.87%

```



شکل ۴۹۴۹ ورودی و خروجی مدل VGG-19 برای تصویر کتاب فروشی

خروجی شبکه:

Bookshop (69.87%)

ج) برای فهمیدن اینکه شبکه ما قابلیت تشخیص چه نوع عکس هایی را دارد، به این گونه عمل کردیم که پس از اینکه VGG19 را فراخوانی کردیم، عکس شماره ۵۳ که در گزارش آمده یعنی گربه ای که روی کتاب نشسته است را وارد کردیم، بجای اینکه نزدیک ترین کلاس را به عکس اختصاص دهد، ۱۰۰۰ کلاس نزدیک به آن را مشاهده کرده، که با این کار تمام کلاسها (دسته ها) می موجود در شبکه VGG را مشاهده می کنیم، که اسامی این کلاس ها در یک فایل به نام labels.txt به همراه کد ها آپلود شد. نهایتا لیبل های قابل شناسایی توسط این شبکه بصورت زیر است:

Persian\_cat', 'paper\_towel', 'toilet\_tissue', 'crossword\_puzzle', 'lynx', 'studio\_couch', 'quilt', 'carton', 'desk', 'crib', 'mouse', 'hamper', 'plastic\_bag', 'remote\_control', 'cradle', 'computer\_keyboard', 'notebook', 'laptop', 'Egyptian\_cat', 'Siamese\_cat', 'Pomeranian', 'envelope', 'scale', 'Angora', 'tabby', 'washbasin', 'rule', 'handkerchief', 'quill', 'file', 'jigsaw\_puzzle', 'bassinet', 'tub', 'wardrobe', 'bath\_towel', 'lens\_cap', 'bow\_tie', 'pillow', 'letter\_opener', 'Band\_Aid', 'printer', 'crate', 'toilet\_seat', 'feather\_boa', 'bathtub', 'sleeping\_bag', 'binder', 'iron', 'photocopier', 'dining\_table', 'rocking\_chair', 'mosquito\_net', 'desktop\_computer', 'ping-pong\_ball', 'lotion', 'Arctic\_fox', 'West\_Highland\_white\_terrier', 'window\_screen', 'tiger\_cat', 'Maltese\_dog', 'radiator', 'table\_lamp', 'packet', 'bib', 'purse', 'shoji', 'bookcase', 'book\_jacket', 'space\_heater', 'dumbbell', 'spatula', 'space\_bar', 'sewing\_machine', 'abacus', 'lampshade', 'face\_powder', 'fountain\_pen', 'Samoyed', 'syringe', 'shower\_curtain', 'teddy', 'candle', 'dishwasher', 'screen', 'stethoscope', 'cup', 'rubber\_eraser', 'keeshond', 'diaper', 'mousetrap', 'four-poster', 'washer', 'mailbag', 'wallet', 'bucket', 'chest', 'monitor', 'ashcan', 'bonnet', 'slide\_rule', 'screwdriver', 'refrigerator', 'harmonica', 'drumstick', 'hamster', 'nipple', 'wooden\_spoon', 'hail', 'window\_shade', 'ballpoint', 'wok', 'cowboy\_hat', 'jean', 'pill\_bottle', 'safety\_pin', 'hammer', 'swab', 'pencil\_box', 'lighter', 'backpack', 'velvet', 'Crock\_Pot', 'Granny\_Smith', 'measuring\_cup', 'papillon', 'thimble', 'screw', 'running\_shoe', 'Pekinese', 'sock', 'tennis\_ball', 'perfume', 'medicine\_chest', 'sweatshirt', 'folding\_chair', 'pinwheel', 'umbrella', 'Christmas\_stocking', 'acoustic\_guitar',

'weasel', 'corn', 'polecat', 'paintbrush', 'lab\_coat', 'ice\_lolly', 'hatchet', 'tray', 'coffee\_mug', 'matchstick', 'hand\_blower', 'toy\_poodle', 'cleaver', 'television', 'electric\_guitar', 'sunscreen', 'Lhasa', 'black-footed\_ferret', 'hook', 'pick', 'ladle', 'beaker', 'goblet', 'electric\_fan', 'seat\_belt', 'loudspeaker', 'Japanese\_spaniel', 'banjo', 'typewriter\_keyboard', 'bookshop', 'spotlight', 'apron', 'pajama', 'upright', 'Great\_Pyrenees', 'hard\_disc', 'toaster', 'punching\_bag', 'waffle\_iron', 'shopping\_basket', 'flute', 'iPod', 'prayer\_rug', 'modem', 'ear', 'menu', 'Shih-Tzu', 'piggy\_bank', 'sliding\_door', 'lipstick', 'pencil\_sharpener', 'Chihuahua', 'neck\_brace', 'safe', 'Scotch\_terrier', 'affenpinscher', 'dial\_telephone', 'frying\_pan', 'Norfolk\_terrier', 'microphone', 'can\_opener', 'wall\_clock', 'projector', 'kuvasz', 'oil\_filter', 'dishrag', 'brassiere', 'red\_fox', 'microwave', 'shower\_cap', 'kit\_fox', 'Yorkshire\_terrier', 'wool', 'Eskimo\_dog', 'jay', 'carpenter's\_kit', 'skunk', 'sandal', 'hair\_slide', 'cellular\_telephone', 'strainer', 'butternut\_squash', 'cairn', 'radio', 'restaurant', 'mask', 'chiffonier', 'soap\_dispenser', 'grand\_piano', 'comic\_book', 'pedestal', 'fire\_screen', 'mink', 'violin', 'soup\_bowl', 'switch', 'guillotine', 'grey\_fox', 'doormat', 'sunglasses', 'malamute', 'Old\_English\_sheepdog', 'mixing\_bowl', 'golf\_ball', 'water\_jug', 'broom', 'oboe', 'wine\_bottle', 'dough', 'vacuum', 'mashed\_potato', 'maraca', 'hare', 'library', 'Siberian\_husky', 'groenendael', 'mittens', 'buckle', 'trench\_coat', 'dalmatian', 'plunger', 'whistle', 'plate', 'CD\_player', 'jersey', 'titis', 'kimono', 'trifle', 'saltshaker', 'stretcher', 'cassette', 'white\_wolf', 'gasmask', 'mortar', 'tripod', 'digital\_clock', 'bottlecap', 'cheetah', 'water\_bottle', 'pitcher', 'custard\_apple', 'tile\_roof', 'snow\_leopard', 'web\_site', 'Shetland\_sheepdog', 'puck', 'mortarboard', 'rifle', 'hair\_spray', 'joystick', 'loupe', 'china\_cabinet', 'schipperke', 'barber\_chair', 'Norwich\_terrier', 'binoculars', 'horned\_viper', 'torch', 'teapot', 'parachute', 'Border\_collie', 'guinea\_pig', 'chow', 'spindle', 'espresso', 'altar', 'hand-held\_computer', 'tick', 'vase', 'soccer\_ball', 'clog', 'scabbard', 'pineapple', 'shovel', 'conch', 'flagpole', 'stove', 'Windsor\_tie', 'shopping\_cart', 'banana', 'entertainment\_center', 'plate\_rack', 'cauliflower', 'obelisk', 'miniature\_schnauzer', 'Sealyham\_terrier', 'magpie', 'baseball', 'collie', 'pool\_table', 'eggnog', 'analog\_clock', 'starfish', 'orange', 'shoe\_shop', 'coffeepot', 'maze', 'pug', 'cash\_machine', 'birdhouse', 'African\_grey', 'pole', 'capuchin', 'combination\_lock', 'stole', 'miniature\_poodle', 'bulletproof\_vest', 'panpipe', 'cornet', 'fur\_coat', 'mailbox', 'oscilloscope', 'oxygen\_mask', 'grocery\_store', 'espresso\_maker', 'barometer', 'bagel', 'Boston\_bull', 'marmoset', 'cockroach', 'ocarina', 'cucumber', 'strawberry', 'honeycomb', 'tape\_player', 'red\_wine', 'half\_track', 'Norwegian\_elkhound', 'king\_snake', 'muzzle', 'accordion', 'daisy', 'cocktail\_shaker', 'beer\_glass', 'miniskirt', 'French\_horn', 'broccoli', 'bassoon', 'toy\_terrier', 'corkscrew', 'reflex\_camera', 'wolf\_spider', 'schooner', 'power\_drill', 'knot', 'ostrich', 'wood\_rabbit', 'knee\_pad', 'cloak', 'projectile', 'barrow', 'picket\_fence', 'burrito', 'revolver', 'patio', 'freight\_car', 'Tibetan\_terrier', 'sombbrero', 'vault', 'vestment', 'ski\_mask', 'magnetic\_compass', 'home\_theater', 'cello', 'mosque', 'chickadee', 'chambered\_nautilus', 'Border\_terrier', 'Polaroid\_camera', 'space\_shuttle', 'maillot', 'pot', 'crutch', 'Brabancion\_griffon', 'missile', 'rhinoceros\_beetle', 'leopard', 'harp', 'trimaran', 'swing', 'wire-haired\_fox\_terrier', 'ski', 'hoopskirt', 'boa\_constrictor', 'assault\_rifle', 'container\_ship', 'crash\_helmet', 'Australian\_terrier', 'Dutch\_oven', 'barrel', 'Petri\_dish', 'mountain\_tent', 'sunglass', 'wig', 'Dungeness\_crab', 'maillot', 'night\_snake', 'borzoi', 'barbell', 'wing', 'swimming\_trunks', 'geyser', 'drum', 'suit', 'military\_uniform', 'silky\_terrier', 'cardigan', 'pomegranate', 'whippet', 'briard', 'breastplate', 'ringneck\_snake', 'bustard', 'pizza', 'plane', 'zucchini', 'porcupine', 'bathing\_cap', 'crayfish', 'English\_setter', 'motor\_scooter', 'cabbage\_butterfly', 'Bedlington\_terrier', 'spaghetti\_squash', 'bulbul', 'prison', 'park\_bench', 'jack-o-lantern', 'theater\_curtain', 'necklace', 'guacamole', 'bow', 'soft-coated\_wheaten\_terrier', 'hotdog', 'crane', 'scorpion', 'lemon', 'bolo\_tie', 'beacon', 'centipede', 'Cardigan', 'quail', 'cougar', 'church', 'car\_mirror', 'fly', 'bobsled', 'bald\_eagle', 'bell\_cote', 'holster', 'airliner', 'Pembroke', 'reel', 'sundial', 'cinema', 'gondola', 'poncho', 'ant', 'organ', 'sandbar', 'breakwater', 'potpie', 'komondor', 'marimba', 'groom', 'rugby\_ball', 'abaya', 'diamondback', 'football\_helmet', 'Bouvier\_des\_Flandres', 'head\_cabbage', 'damselfly', 'bubble', 'tarantula', 'earthstar', 'thunder\_snake', 'Madagascar\_cat', 'manhole\_cover', 'bearskin', 'albatross', 'toyshop', 'throne', 'Scottish\_deerhound', 'green\_mamba', 'beer\_bottle', 'rain\_barrel', 'slug', 'gong', 'basenji', 'cowboy\_boot', 'alp', 'mushroom', 'meerkat', 'tricycle', 'caldron', 'picklehaube', 'vine\_snake', 'cassette\_player', 'stage', 'dragonfly', 'red-breasted\_merganser', 'confectionery', 'Italian\_greyhound', 'digital\_watch', 'hognose\_snake', 'bluetick', 'dome', 'harvestman', 'redbone', 'paddlewheel', 'carbonara', 'bannister', 'Loafer', 'tobacco\_shop', 'clumber', 'giant\_panda', 'barbershop', 'gown', 'timber\_wolf', 'hourglass', 'French\_bulldog', 'Lakeland\_terrier', 'consomme', 'trombone', 'pop\_bottle', 'lesser\_panda', 'balloon', 'sidewinder', 'shield', 'bakery', 'warplane', 'bullet\_train', 'pier', 'badger', 'English\_springer', 'stopwatch', 'giant\_schnauzer', 'llama', 'milk\_can', 'moving\_van', 'ice\_cream', 'chime', 'hen', 'Irish\_wolfhound', 'potter's\_wheel', 'golden\_retriever', 'liner', 'bell\_pepper', 'cocker\_spaniel', 'American\_egret', 'racer', 'gibbon', 'standard\_poodle', 'mantis', 'koala', 'Mexican\_hairless', 'dam', 'coil', 'sea\_urchin', 'airship', 'Bernese\_mountain\_dog', 'street\_sign', 'lifeboat', 'Gila\_monster', 'volleyball', 'triumphal\_arch', 'sax', 'barn', 'pretzel', 'hen-of-the-woods', 'white\_stork', 'forklift', 'fig', 'Labrador\_retriever', 'black\_widow', 'coyote', 'brass', 'padlock', 'lycaenid', 'tank', 'long-horned\_beetle', 'snowplow', 'Irish\_terrier', 'moped', 'langur', 'worm\_fence', 'trailer\_truck', 'yaw!', 'bee', 'speedboat', 'ladybug', 'acorn\_squash', 'Indian\_cobra', 'miniature\_pinscher', 'hay', 'minivan', 'Newfoundland', 'castle', 'Saint\_Bernard', 'parallel\_bars', 'horizontal\_bar', 'ruffed\_grouse', 'tiger\_shark', 'green\_snake', 'fireboat', 'academic\_gown', 'fountain', 'alligator\_lizard', 'seashore', 'pirate', 'planetarium', 'car\_wheel', 'Weimaraner', 'indri', 'Blenheim\_spaniel', 'wallaby', 'stingray', 'otter', 'stone\_wall', 'French\_loaf', 'chainlink\_fence', 'macaw', 'balance\_beam', 'cannon', 'scoreboard', 'green\_lizard', 'sea\_anemone', 'water\_ouzel', 'electric\_ray', 'barn\_spider', 'macaque', 'garden\_spider', 'water\_tower', 'stupa', 'whiskey\_jug', 'sulphur-crested\_cockatoo', 'kite', 'coho', 'guenon', 'water\_snake', 'hyena', 'dhole', 'fox\_squirrel', 'Brittany\_spaniel',

'sports\_car', 'cheeseburger', 'triceratops', 'golfcart', 'standard\_schnauzer', 'boxer', 'ice\_bear', 'hornbill', 'cock', 'walking\_stick', 'solar\_dish', 'Ibizan\_hound', 'hip', 'artichoke', 'killer\_whale', 'acorn', 'house\_finch', 'isopod', 'grille', 'paddle', 'hot\_pot', 'greenhouse', 'German\_shepherd', 'king\_crab', 'gar', 'pay-phone', 'chocolate\_sauce', 'trilobite', 'cab', 'toucan', 'Saluki', 'ptarmigan', 'banded\_gecko', 'whiptail', 'Doberman', 'Dandie\_Dimont', 'cricket', 'yurt', 'agama', 'great\_white\_shark', 'little\_blue\_heron', 'racket', 'meat\_loaf', 'Airedale', 'jaguar', 'Appenzeller', 'canoe', 'American\_lobster', 'tiger', 'black\_and\_gold\_garden\_spider', 'snail', 'Welsh\_springer\_spaniel', 'weevil', 'catamaran', 'Greater\_Swiss\_Mountain\_dog', 'lion', 'turnstile', 'tow\_truck', 'sarong', 'butcher\_shop', 'bikini', 'partridge', 'cuirass', 'Great\_Dane', 'radio\_telescope', 'lumbermill', 'vizsla', 'wombat', 'Model\_T', 'chain', 'Tibetan\_mastiff', 'palace', 'submarine', 'black\_grouse', 'unicycle', 'American\_chameleon', 'chain\_mail', 'limousine', 'hammerhead', 'armadillo', 'go-kart', 'overskirt', 'dingo', 'patas', 'recreational\_vehicle', 'Kerry\_blue\_terrier', 'dock', 'volcano', 'red-backed\_sandpiper', 'spider\_web', 'garbage\_truck', 'Staffordshire\_bullterrier', 'drilling\_platform', 'steel\_arch\_bridge', 'eft', 'goldfish', 'aircraft\_carrier', 'barracouta', 'fiddler\_crab', 'vulture', 'marmot', 'sea\_lion', 'flat-coated\_retriever', 'hermit\_crab', 'convertible', 'redshank', 'agaric', 'sea\_snake', 'American\_Staffordshire\_terrier', 'cardoon', 'great\_grey\_owl', 'beagle', 'rapeseed', 'siamang', 'snowmobile', 'Afghan\_hound', 'ringlet', 'coral\_fungus', 'axolotl', 'otterhound', 'parking\_meter', 'steel\_drum', 'hog', 'disk\_brake', 'tractor', 'indigo\_bunting', 'monastery', 'bull\_mastiff', 'common\_iguana', 'harvester', 'robin', 'lakeside', 'oystercatcher', 'buckeye', 'sulphur\_butterfly', 'beaver', 'echidna', 'orangutan', 'colobus', 'rock\_crab', 'lacewing', 'mongoose', 'basketball', 'traffic\_light', 'platypus', 'slot', 'monarch', 'pelican', 'grasshopper', 'howler\_monkey', 'black-and-tan\_coonhound', 'king\_penguin', 'limpkin', 'German\_short-haired\_pointer', 'American\_alligator', 'Gordon\_setter', 'leatherback\_turtle', 'bloodhound', 'prairie\_chicken', 'EntleBucher', 'pickup', 'squirrel\_monkey', 'amphibian', 'peacock', 'apiary', 'totem\_pole', 'cliff\_dwelling', 'frilled\_lizard', 'Leonberg', 'ballplayer', 'lawn\_mower', 'malinois', 'basset', 'chimpanzee', 'beach\_wagon', 'steam\_locomotive', 'brambling', 'crane', 'bee\_eater', 'dogsled', 'wreck', 'odometer', 'Chesapeake\_Bay\_retriever', 'sea\_slug', 'suspension\_bridge', 'fire\_engine', 'police\_van', 'ground\_beetle', 'goose', 'bison', 'brain\_coral', 'rock\_python', 'rotisserie', 'horse\_cart', 'valley', 'hummingbird', 'Rottweiler', 'spoonbill', 'plow', 'Irish\_water\_spaniel', 'promontory', 'gorilla', 'nematode', 'spotted\_salamander', 'Irish\_setter', 'terrapin', 'ambulance', 'goldfinch', 'red\_wolf', 'viaduct', 'jinrikisha', 'thresher', 'dung\_beetle', 'minibus', 'African\_chameleon', 'jackfruit', 'chain\_saw', 'bittern', 'thatch', 'megalith', 'junco', 'boathouse', 'snorkel', 'admiral', 'coucal', 'carousel', 'Walker\_hound', 'ruddy\_turnstone', 'leaf\_beetle', 'croquet\_ball', 'tiger\_beetle', 'passenger\_car', 'kelpie', 'American\_black\_bear', 'Sussex\_spaniel', 'European\_gallinule', 'tree\_frog', 'mobile\_home', 'school\_bus', 'flamingo', 'bolete', 'zebra', 'jeep', 'puffer', 'English\_foxhound', 'cicada', 'jacamar', 'baboon', 'leafhopper', 'anemone\_fish', 'Rhodesian\_ridgeback', 'European\_fire\_salamander', 'African\_elephant', 'Arabian\_camel', 'American\_coot', 'garter\_snake', 'spider\_monkey', 'box\_turtle', 'ibex', 'cliff', 'common\_newt', 'African\_hunting\_dog', 'African\_crocodile', 'vending\_machine', 'lorikeet', 'bicycle-built-for-two', 'proboscis\_monkey', 'brown\_bear', 'grey\_whale', 'streetcar', 'tailed\_frog', 'eel', 'spiny\_lobster', 'yellow\_lady's\_slipper', 'sturgeon', 'gazelle', 'jellyfish', 'curly-coated\_retriever', 'rock\_beauty', 'black\_swan', 'bullock', 'coral\_reef', 'ram', 'sloth\_bear', 'dowitcher', 'loggerhead', 'Indian\_elephant', 'scuba\_diver', 'black\_stork', 'wild\_boar', 'mountain\_bike', 'stinkhorn', 'ox', 'dugong', 'warthog', 'oxcart', 'sea\_cucumber', 'lionfish', 'flatworm', 'three-toed\_sloth', 'hartebeest', 'maypole', 'Komodo\_dragon', 'drake', 'bighorn', 'chiton', 'trolleybus', 'gas\_pump', 'gyromitra', 'impala', 'tench', 'electric\_locomotive', 'mud\_turtle', 'tusker', 'water\_buffalo', 'sorrel', 'hippopotamus']

همانطور که از نتایج مشاهده می‌نماییم مدل آموزش دیده در تشخیص برخی تصاویر به خوبی عمل نمی‌کند به گونه‌ای که حتی گاهها اشتباه هم تشخیص می‌دهد. تصاویری که در نمونه‌های تست به خوبی تشخیص داده شده اند عبارت‌اند از: حیوانات (مانند شیر)، ماشین (مانند برفروب)، دوچرخه، لیوان (مانند ماق)، کتابفروشی، خانه و البته در برخی نمونه‌های تست نیز به خوبی رفتار نمی‌کند که از جمله‌ی آن گاها لپ تاپ که به عنوان نوت بوک شناسایی کرد و نیز دفتر که به عنوان کیف پول شناسایی کرد.

در باب اینکه اگر عکسی در داخل دسته نباشد چه می‌شود باید ذکر کرد که مدل با استخراج مشخصه و مولفه‌های ویژگی تصویر، مشابه‌ترین تصویر را که مولفه‌های ویژگی نزدیکی به تصویر ورودی دارد، به عنوان شی تشخیص داده شده برمی‌گرداند. ولی اینکه چگونه متوجه شویم که prediction به درستی انجام شده ساده‌ترین راه مشاهده احتمال درستی تشخیص است و البته راه‌های علمی گوناگونی نیز در مراجع

ذکر شده از جمله ساخت یک autoencoder بر روی داده‌های آموزش که در این روش داده‌های ورودی را ابتدا به encoder و متناظر با آن می‌دهند و اگر مدل به خوبی تصویر اولیه را بازگرداند که می‌توانیم نتیجه بگیریم که این نمونه در مجموعه داده‌گان آموزش موجود بوده و قابلیت تشخیص درست را به خوبی دارد و در غیر اینصورت قابل تشخیص با مدل آموزش دیده نخواهد بود.

حال مثال‌هایی که گفته شد و اشتباه تشخیص داده شد:

```

1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('/gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(2
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print('%.2f%%' % (label[1], label[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly rem
notebook (62.02%)

```



شکل ۵۰۵۰ ورودی و خروجی مدل VGG-19 برای تصویر لپ تاپ

ورودی شبکه تصویر لپ تاپ است ولی خروجی شبکه:

Notebook (62.02%)

```

1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6
7 from google.colab import drive
8 drive.mount('/content/gdrive')
9
10 model = VGG19()
11 image = load_img('/gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(2
12 image = img_to_array(image)
13 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
14 image = preprocess_input(image)
15 yhat = model.predict(image)
16 label = decode_predictions(yhat)
17 label = label[0][0]
18 print('%.2f%%' % (label[1], label[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly rem
wallet (21.14%)

```

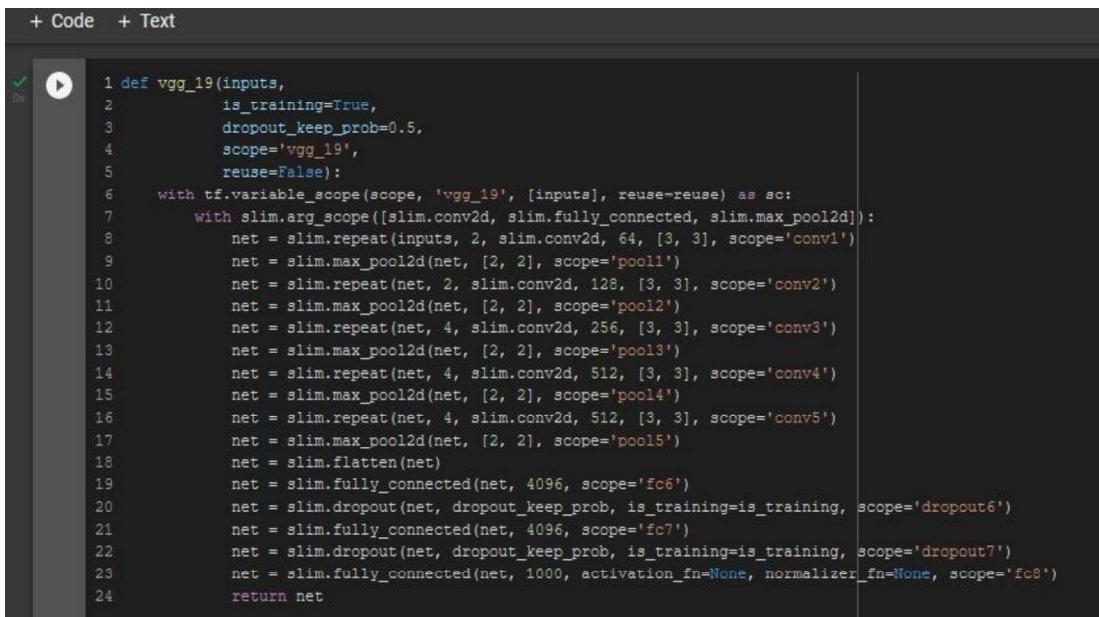


شکل ۵۱۵۱ ورودی و خروجی مدل VGG-19 برای تصویر کتاب یا دفترچه

ورودی شبکه تصویر دفتر است ولی خروجی شبکه:

Wallet (21.14%)

۵) مدل VGG-19 بصورت زیر پیاده سازی شده است:



```
1 def vgg_19(inputs,
2             is_training=True,
3             dropout_keep_prob=0.5,
4             scope='vgg_19',
5             reuse=False):
6     with tf.variable_scope(scope, 'vgg_19', [inputs], reuse=reuse) as sc:
7         with slim.arg_scope([slim.conv2d, slim.fully_connected, slim.max_pool2d]):
8             net = slim.repeat(inputs, 2, slim.conv2d, 64, [3, 3], scope='conv1')
9             net = slim.max_pool2d(net, [2, 2], scope='pool1')
10            net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], scope='conv2')
11            net = slim.max_pool2d(net, [2, 2], scope='pool2')
12            net = slim.repeat(net, 4, slim.conv2d, 256, [3, 3], scope='conv3')
13            net = slim.max_pool2d(net, [2, 2], scope='pool3')
14            net = slim.repeat(net, 4, slim.conv2d, 512, [3, 3], scope='conv4')
15            net = slim.max_pool2d(net, [2, 2], scope='pool4')
16            net = slim.repeat(net, 4, slim.conv2d, 512, [3, 3], scope='conv5')
17            net = slim.max_pool2d(net, [2, 2], scope='pool5')
18            net = slim.flatten(net)
19            net = slim.fully_connected(net, 4096, scope='fc6')
20            net = slim.dropout(net, dropout_keep_prob, is_training=is_training, scope='dropout6')
21            net = slim.fully_connected(net, 4096, scope='fc7')
22            net = slim.dropout(net, dropout_keep_prob, is_training=is_training, scope='dropout7')
23            net = slim.fully_connected(net, 1000, activation_fn=None, normalizer_fn=None, scope='fc8')
24        return net
```

شکل ۵۲ مدل VGG19 پیاده سازی شده

لازم به ذکر است که دیتاست ImageNet بالغ بر ۱۴ میلیون نمونه و ۱۰۰۰ کلاس دارد و عملاً آموزش شبکه با این دیتاست از منظر پیچیدگی زمانی بسیار هزینه بر می باشد.

نمونه خروجی های دیگری که از شبکه گرفتیم:



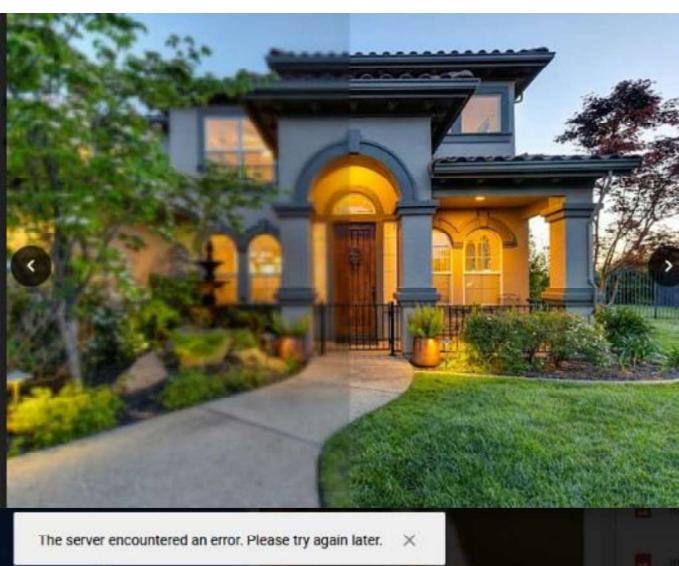
The image shows a white Persian kitten with blue eyes, sitting on an open book. The book is resting on a light-colored surface against a yellow background. The kitten is looking towards the camera.

```
Cell + Text
1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6 import numpy as np
7
8 from google.colab import drive
9 drive.mount('/content/gdrive')
10
11 model = VGG19()
12 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(224, 224))
13 image = img_to_array(image)
14 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
15 image = preprocess_input(image)
16 yhat = model.predict(image)
17 label = decode_predictions(yhat)
18 Len = np.array(label).shape[1]
19 for i in range(Len):
20   label_ = label[0][i]
21   print('%.2f%%' % (label_[1], label_[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly remount, first unmount it.
Persian_cat (37.30%)
paper_towel (29.15%)
toilet_tissue (3.45%)
crossword_puzzle (3.35%)
lynx (3.34%)
```

The server encountered an error. Please try again later. ×

شکل ۵۳ ورودی و خروجی مدل VGG-19 برای تصویر چندتایی



The image shows a large, modern house with a dark exterior and bright yellow lights illuminating its arched entrance and windows. The house is surrounded by greenery and a paved walkway.

```
Cell + Text
1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6 import numpy as np
7
8 from google.colab import drive
9 drive.mount('/content/gdrive')
10
11 model = VGG19()
12 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(224, 224))
13 image = img_to_array(image)
14 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
15 image = preprocess_input(image)
16 yhat = model.predict(image)
17 label = decode_predictions(yhat)
18 Len = np.array(label).shape[1]
19 for i in range(Len):
20   label_ = label[0][i]
21   print('%.2f%%' % (label_[1], label_[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly remount, first unmount it.
triumphal arch (39.85%)
palace (36.74%)
fountain (3.92%)
monastery (3.35%)
pedestal (2.47%)
```

The server encountered an error. Please try again later. ×

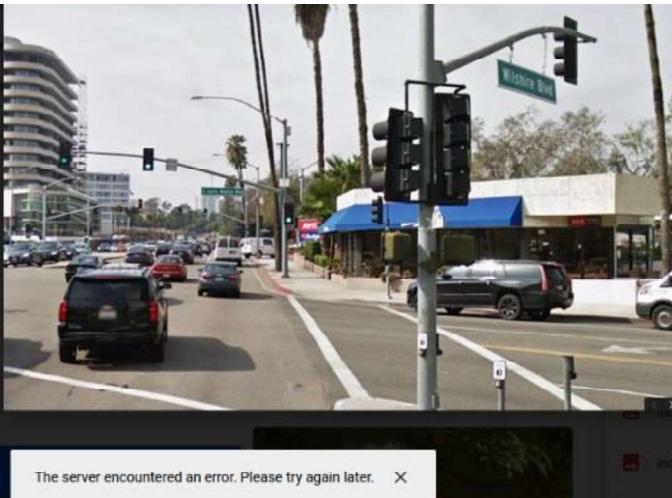
شکل ۵۴ ورودی و خروجی مدل VGG-19 برای تصویر چندتایی

```

1 from keras.preprocessing.image import load_img
2 from keras.preprocessing.image import img_to_array
3 from keras.applications.vgg19 import preprocess_input
4 from keras.applications.vgg19 import decode_predictions
5 from keras.applications.vgg19 import VGG19
6 import numpy as np
7
8 from google.colab import drive
9 drive.mount('/content/gdrive')
10
11 model = VGG19()
12 image = load_img('gdrive/MyDrive/Colab Notebooks/index.jpg', target_size=(2
13 image = img_to_array(image)
14 image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
15 image = preprocess_input(image)
16 yhat = model.predict(image)
17 label = decode_predictions(yhat)
18 Len = np.array(label).shape[1]
19 for i in range(Len):
20   label_=label[0][i]
21   print('*s (%.2f%%)' % (label_[1], label_[2]*100))

Drive already mounted at /content/gdrive; to attempt to forcibly rem
traffic_light (58.66%)
pole (6.76%)
trolleybus (4.56%)
street_sign (2.77%)
tow_truck (2.37%)

```



شکل ۵۵ ورودی و خروجی مدل VGG-19 برای تصویر چندتایی

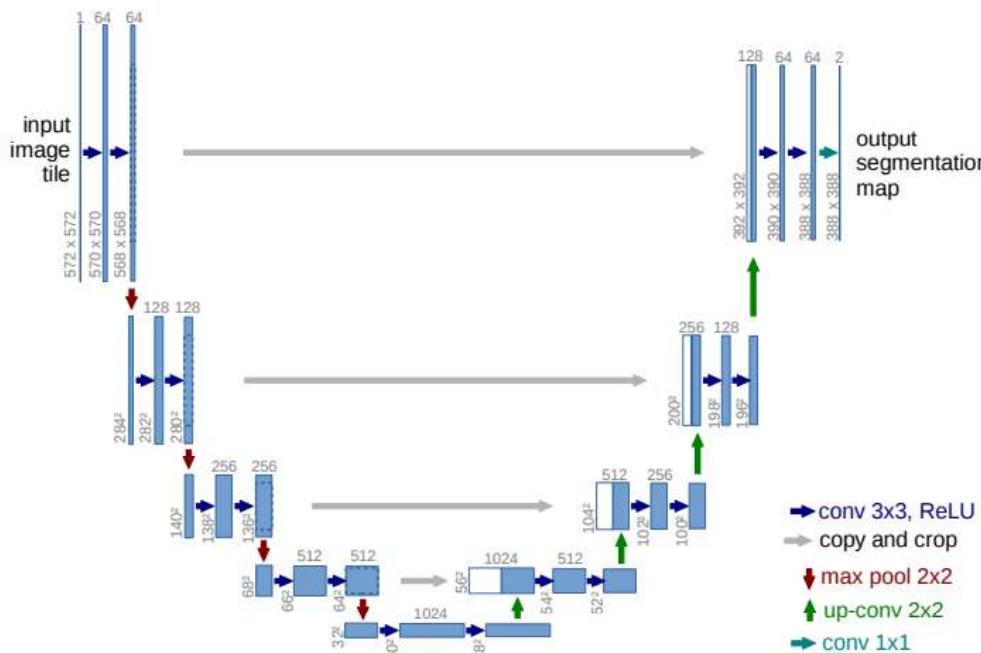
### Semantic Segmentation – ۳

در این سوال از بین دو مدل UNET و DEEPLAB، مدل UNET را انتخاب کردیم.

الف) معماری و کاربرد مدل

#### U-Net معماری مدل

معماری U-Net یک شبکه عصبی پیچشی است که در اصل برای بخشندی تصاویر در حوزه پژوهشی طراحی شده است. شمایل این معماری شبیه حرف U انگلیسی است و به همین دلیل نیز U-Net یا شبکه U شکل نامیده می‌شود. معماری این شبکه از دو بخش تشکیل شده است که در شکل زیر آورده شده است: بخش سمت چپ، مسیر فشرده‌سازی و بخش سمت راست، مسیر گستردگی‌سازی است. هدف مسیر فشرده‌سازی در کضمون تصویر و نقش مسیر گستردگی‌سازی کمک کردن در فرآیند مکانیابی دقیق اشیاء است.



شکل ۵۶ معماری U-Net

مسیر قرارداده شده از معماری معمولی یک شبکه کانولوشن پیروی می کند. این شامل استفاده مکرر از دو کانولوشن  $3 \times 3$  است (Unpadded convolution) که هر کدام توسط یک ReLu و یک max pooling down sampling با گام ۲ برای نمونه برداری پایین (Down sampling) انجام می شود. در هر گام نمونه برداری  $2 \times 2$  تعداد کanal های ویژگی را دو برابر می کنیم. هر مرحله در مسیر گسترده شامل یک نمونه برداری upsampling از feature map و به دنبال آن یک کانولوشن  $2 \times 2$  (Up convolution) است که تعداد کanal های ویژگی را نصف می کند، و دو کانولوشن  $3 \times 3$  که هر کدام توسط یک ReLU دنبال می شوند.

به دلیل از بین رفتن پیکسل های حاشیه در هر کانولوشن، cropping ضروری است. در لایه نهایی یک کانولوشن  $1 \times 1$  برای ترسیم هر بردار ویژگی ۶۴ جزء به تعداد کلاس های مورد نظر استفاده می شود. در مجموع شبکه دارای ۲۲ لایه کانولوشنال است.

برای اجازه دادن به seamless tiling از segmentation map خروجی، مهم است که tile size ورودی را به گونه ای انتخاب کنید که تمام عملیات max-pooling  $2 \times 2$  بر روی یک لایه با اندازه x و y یکنواخت اعمال شود.

در بخش نمونه برداری تعداد زیادی کanal ویژگی داریم که به شبکه اجازه می دهد اطلاعات زمینه را به لایه های با وضوح بالاتر منتشر کند. در نتیجه، مسیر انساطی کم و بیش متقاضی با مسیر انقباض است و یک معماری U شکل ایجاد می کند. شبکه هیچ لایه کاملاً متصل (fully connected) ندارد و فقط از قسمت معتبر هر کانولوشن استفاده می کند، یعنی segmentation map فقط حاوی پیکسل هایی است که زمینه کامل آن در تصویر ورودی موجود است.

داده های آموزشی بسیار کمی در دسترس است، ما از افزایش بیش از حد داده ها با اعمال تغییر شکل های الاستیک در تصاویر آموزشی موجود استفاده می کنیم. این به شبکه اجازه می دهد تا تغییر ناپذیری در برابر چنین تغییر شکل هایی را بدون نیاز به دیدن این تبدیل ها در پیکره تصویر مشروح شده بیاموزد.

## کاربردهای مدل U-Net :

از این مدل برای پردازش تصاویر پزشکی استفاده می شود. مانند تقسیم بندی تصویر مغز و تقسیم بندی تصویر کبد.

- برای تشخیص انواع بیماری ها با استفاده از تصاویر پزشکی
- تشخیص انواع سرطان
- تشخیص بیماری های مغزی
- تشخیص تومورها
- استفاده در موقعي که تصاویر آموزشی کم داریم
- برای بازسازی تصاویر پزشکی استفاده می شود

در میان الگوریتم های یادگیری عمیق، الگوریتم U-Net بدلیل پاسخ دقیق، دقت بالا، سرعت بالای پردازش و یادگیری، عدم نیاز به مجموعه داده های بزرگ برای یادگیری و عدم نیاز به سخت افزارهای پیچیده و گران قیمت، در چند سال اخیر به الگوریتم محبوب شناسایی اجزای تصویر و قطعه بندی کردن آنها در پردازش تصاویر پزشکی بدل شده است.

---

ب) مدل ما به اینصورت است که برای ۳۲ کلاس رنگ های متفاوتی بصورت جدول زیر در نظر گرفته است و یک مپینگ از کلاس ها به رنگ ها خواهیم داشت:

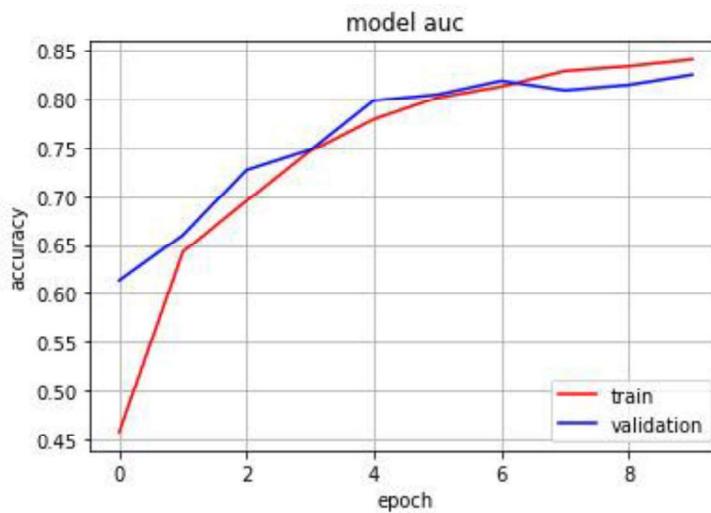
Table 20 ۳۲ کلاس موجود

B	G	R	name
64	128	64	Animal
128	0	192	Archway
192	128	0	Bicyclist
64	128	0	Bridge
0	0	128	Building
128	0	64	Car
192	0	64	CartLuggagePram
64	128	192	Child
128	192	192	Column_Pole

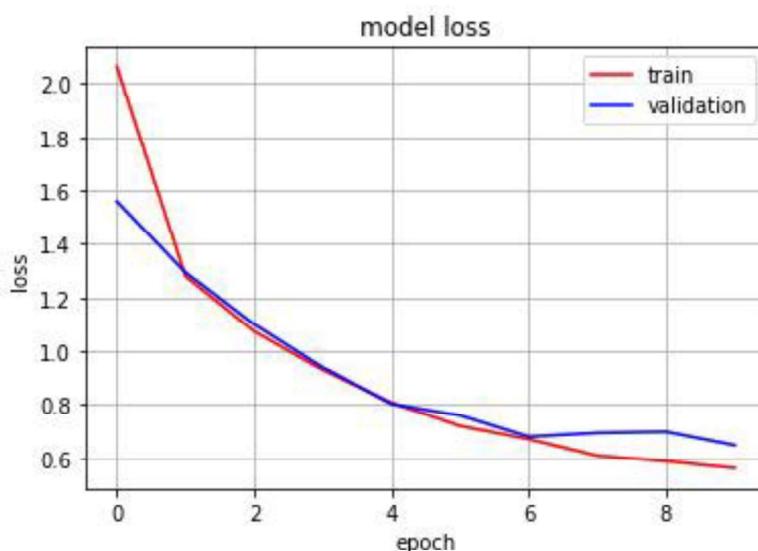
128	64	64	<i>Fence</i>
192	0	128	<i>LaneMkgsDriv</i>
64	0	192	<i>LaneMkgsNonDriv</i>
64	128	128	<i>Misc_Text</i>
192	0	192	<i>MotorcycleScooter</i>
64	64	128	<i>OtherMoving</i>
128	192	64	<i>ParkingBlock</i>
0	64	64	<i>Pedestrian</i>
128	64	128	<i>Road</i>
192	128	128	<i>RoadShoulder</i>
192	0	0	<i>Sidewalk</i>
128	128	192	<i>SignSymbol</i>
128	128	128	<i>Sky</i>
192	128	64	<i>SUVPickupTruck</i>
64	0	0	<i>TrafficCone</i>
64	64	0	<i>TrafficLight</i>
128	64	192	<i>Train</i>
0	128	128	<i>Tree</i>
192	128	192	<i>Truck_Bus</i>
64	0	64	<i>Tunnel</i>
0	192	192	<i>VegetationMisc</i>
0	0	0	<i>Void</i>
0	192	64	<i>Wall</i>

برای پیاده سازی ابتدا داده هایمان را از گیت هاب فراخوانی کردیم، سپس تعداد تصاویر تست و آموزش و ارزیابی را مشخص می کنیم که ۴۲۱ تصویر آموزشی و ۱۶۸ تصویر تست داریم و ۱۱۲ تصویر ارزیابی داریم.

نتایج بر روی چند مثال بصورت زیر است:

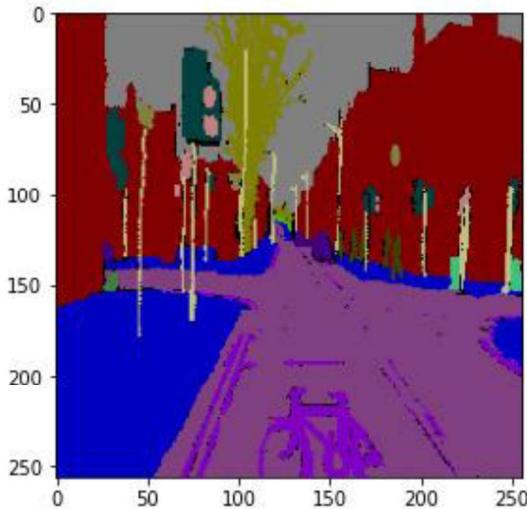


شکل ۵۷ نمودار تغییرات صحت در هر اپیاک



شکل ۵۸ نمودار تغییرات خطأ در هر اپیاک

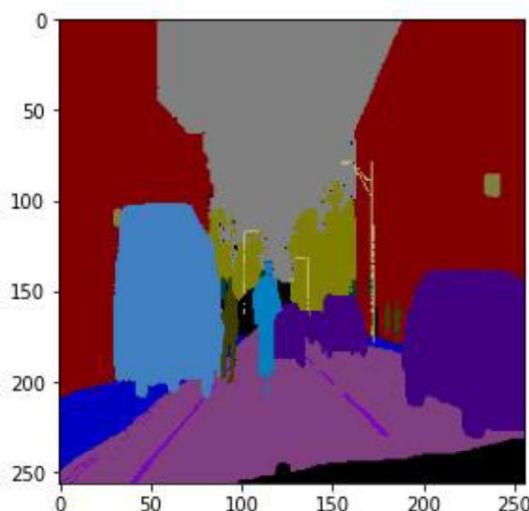
## عکس ها و خروجی شبکه:



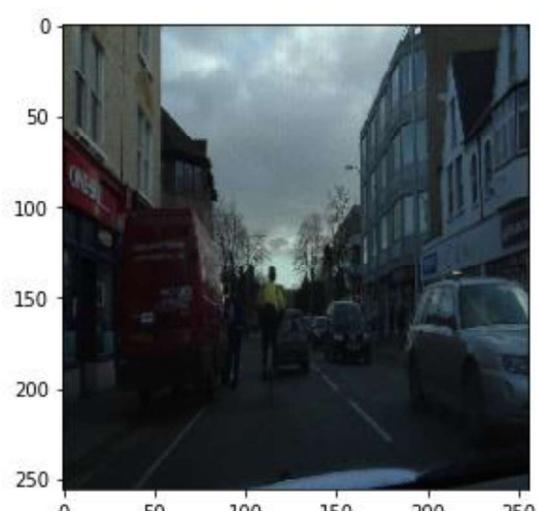
شکل ۶۰ ماسک ارائه شده در تصویر قبل



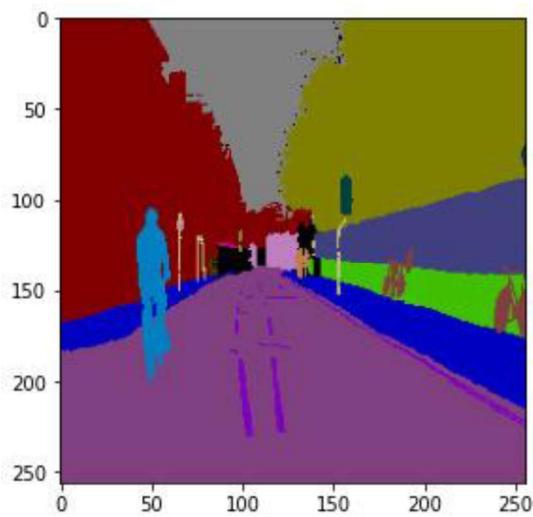
شکل ۵۹ تصویر اولیه



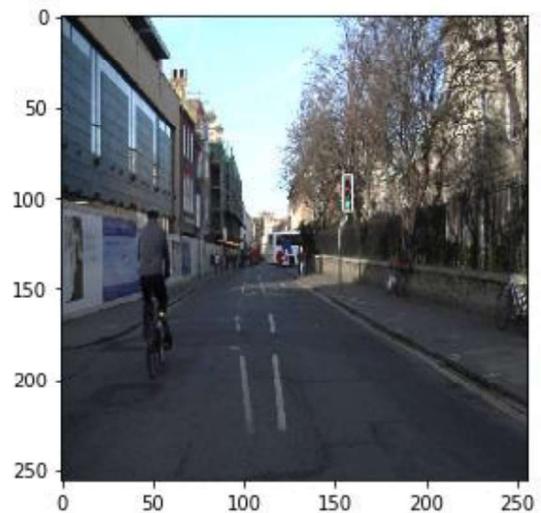
شکل ۶۲ ماسک ارائه شده در تصویر قبل



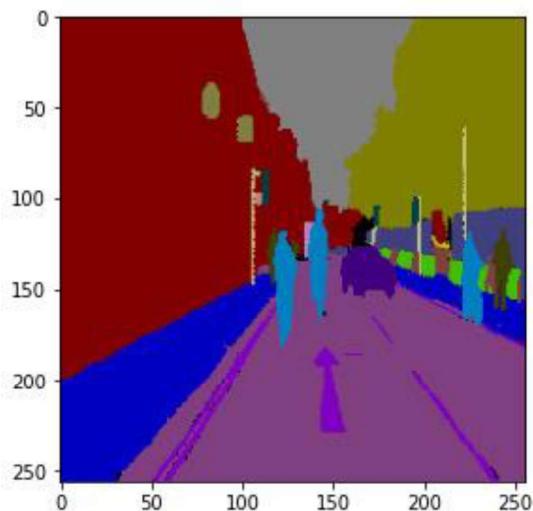
شکل ۶۱ تصویر اولیه



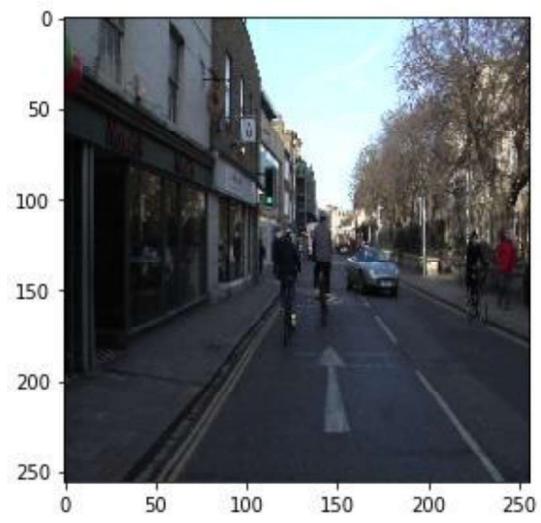
شکل ۶۴ ماسک ارائه شده در تصویر قبل



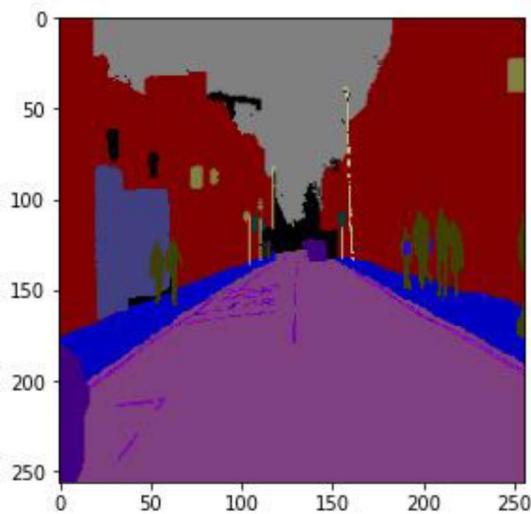
شکل ۶۳ تصویر اولیه



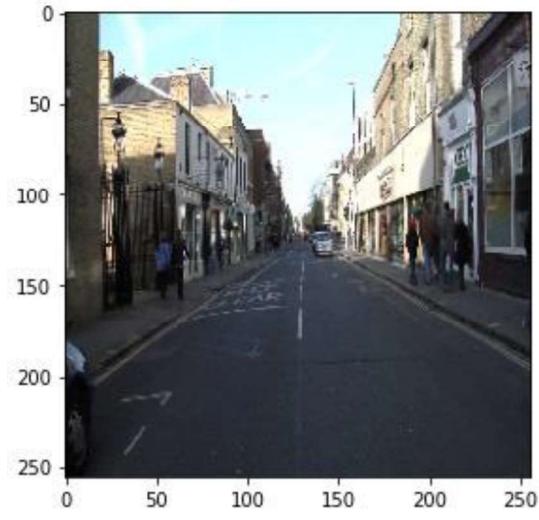
شکل ۶۶ ماسک ارائه شده در تصویر قبل



شکل ۶۵ تصویر اولیه



شکل ۶۵۸ ماسک ارائه شده در تصویر قبل



شکل ۶۷ تصویر اولیه

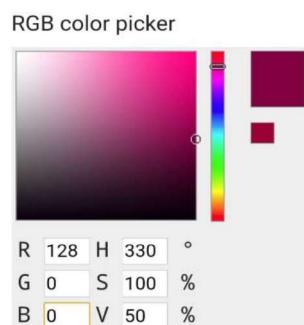
برای توضیح دادن نتایج عکس ۵۹ که خروجی شبکه عکس شماره ۶۰ است داریم:

در این تصویر چراغ راهنمایی وجود دارد که رنگ آن با توجه به Table20 بصورت زیر می شود:



که همانطور که در تصویر ۶۰ می بینیم چراغ راهنمایی دقیقاً به این رنگ شده است.

حال در تصویر شماره ۵۹ دیدیم که ساختمان وجود دارد و با استفاده از Table 20 بصورت زیر می شود:



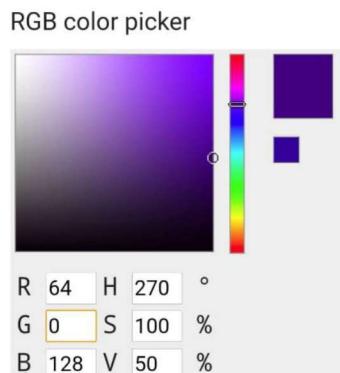
که همانطور که در تصویر ۶۰ می بینیم ساختمان ها دقیقا به این رنگ شده اند.

حال در تصویر شماره ۵۹ دیدیم که جاده وجود دارد و با استفاده از Table 20 بصورت زیر می شود:



که همانطور که در تصویر ۶۰ می بینیم جاده دقیقا به این رنگ شده است.

حال در تصویر شماره ۵۹ دیدیم که ماشین وجود دارد و با استفاده از Table 20 بصورت زیر می شود:



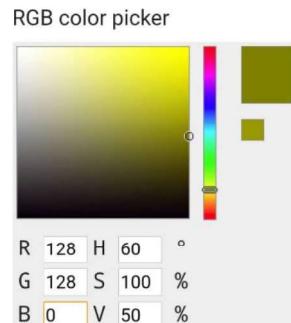
که همانطور که در تصویر ۶۰ می بینیم ماشین ها دقیقا به این رنگ شده اند.

حال در تصویر شماره ۵۹ دیدیم که آسمان وجود دارد و با استفاده از Table 20 بصورت زیر می شود:



که همانطور که در تصویر ۶۰ می بینیم آسمان دقیقا به این رنگ شده است.

حال در تصویر شماره ۵۹ دیدیم که درخت وجود دارد و با استفاده از Table 20 بصورت زیر می شود:



که همانطور که در تصویر ۶۰ می بینیم درخت ها دقیقا به این رنگ شده اند.

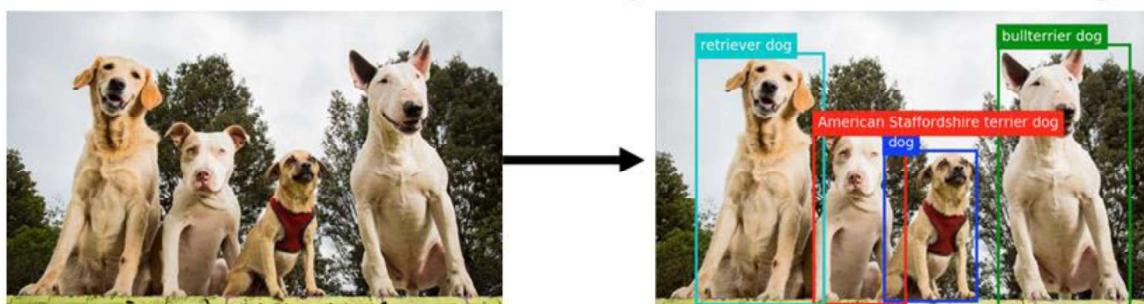
## سوال ۴ – Object Detection

الف) در تشخیص شی، شبکه ها به دو دسته یک مرحله ای و دو مرحله ای تقسیم می شوند:

یک مرحله ای (One-Stage): دو عمل Classification و Localization را در یک آن (Stage) و توسط تنها یک شبکه عصبی عمیق انجام می دهد. به همین دلیل سرعت بالای پردازشی دارد، اما ممکن است کمتری نسبت به دو مرحله ای که در ادامه توضیح خواهیم داد، داشته باشد.

مثال: RetinaNet و SSD network

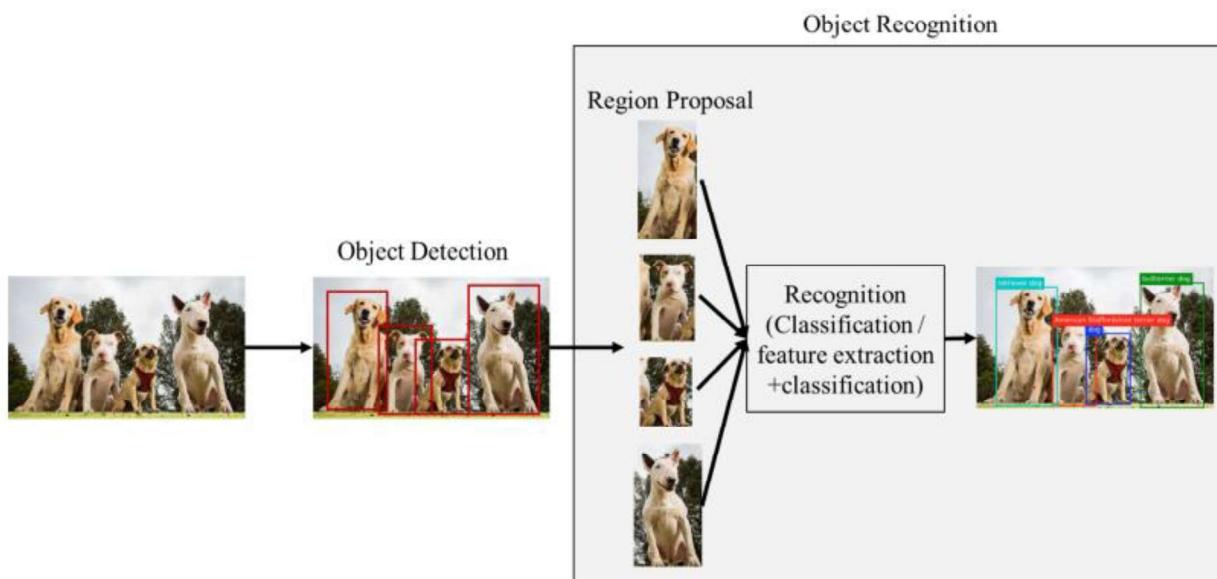
## Object Detection + Recognition



شکل ۶۶۹ مثال از ورودی و خروجی شبکه یک مرحله ای (One-Stage)

**دو مرحله ای(Two-Stage):** بین دو عمل Classification و Localization تمایز قابل می شود و این دو عمل را در دو Stage انجام می دهد، بدین صورت که در ابتدا نواحی مورد نظر (regions of interests) را توسط Select search regional proposal network پیشنهاد می دهد و پس از بدست آوردن نواحی پیشنهادی، در گام دوم(second stage) به کلاسیفایر وارد شده و این کلاسیفایر تنها این نواحی (نواحی منتخب را مورد پردازش قرار می دهد. و نتیجتاً عملیات recognition در این stage صورت می گیرد. به همین دلیل سرعت پردازشی کمتری دارد، اما high localization and recognition ) accuracy منتخب را مورد پردازش قرار می دهد. و بالاتری نسبت به یک مرحله ای دارد.

مثال : Faster RCNN و G-RCNN



شکل ۷۰ مثال از ورودی و خروجی شبکه دو مرحله ای (Two-Stage)

ب) در YOLOv1 هر grid قادر به شناسایی تنها یک شی است و نتیجتاً زمانیکه تصویر متشکل از گروهی از اشیا کوچک (Small Object) یا با تداخل بالا باشند به نحوی که در یک grid قرار می گیرند، عملکرد بدی را از خود نشان خواهد داد.

پس برای حل مشکل فوق و دستیابی به performance بالاتر روش هایی در YOLOv2 اتخاذ شده است که به شرح زیر است:

**اولین راهکار** اضافه کردن (mean Average Precision) mAP است که Batch Normalization را تا ۲ درصد بهبود می یابد.

**دومین راهکار** High Resolution classifier می باشد که در YOLO V2 اتخاذ شده است. بدین صورت که در این روش از تصاویر با رزولوشن بالا به عنوان داده های آموزش و برای ده ایپاک اول، استفاده شده است که نتیجتاً mAP را تا ۴ درصد بهبود می بخشد.

**سومین راهکار** Convolutional with Anchor Boxes است که با اضافه نمودن anchor boxes اقدام به بهبود localization برای تشخیص object ها نمودند. در این روش لایه های فولی کانکت حذف شدند و اقدام به کاهش ابعاد تصاویر ورودی کردند و بدین صورت feature map ها به ابعادی مشابه با ابعاد grid رسیدند و در نهایت با یک عملیات جابجایی از grid cell به bounding box برای کلاس پیش بینی شده از شی به یک ساختاری می رسیم که در آن هر سلول می تواند اشیای مختلف را با باکس های گوناگون مورد پیش بینی (prediction) قرار دهد. این بهبودی مقدار recall را افزایش می دهد.

**چهارمین راهکار** multi scale training است که در این روش آموزش مدل از طریق تصاویری با ابعاد گوناگون صورت می پذیرد. همچنین لازم به ذکر است که در این روش ابعاد تصاویر بصورت تصادفی و در هر ۱۰ عدد batch در طی روند آموزش انتخاب می شوند.

---

(ج)

## نسخه چهارم الگوریتم YOLO

نسخه چهارم الگوریتم YOLO با تمرکز بر دقت و سرعت بهینه در تشخیص اشیاء معروفی شد. این نسخه توانست روی نمونه های دیتاست MS COCO با متوسط دقت ۴۳.۵٪ و نیز با سرعت ۶۵ فریم در ثانیه روی GPU مدل Tesla V100 به نتایج بسیار مطلوبی دست یابد. در راستای کسب این نتایج ویژگی های مختلفی از جمله اتصالات با قیمانده های وزنی (WRC)، اتصالات بخشی میان مرحله ای (CSP)، نرم افزاری

متقابل بسته های داده ای کوچک (CMBn) و نیز خودآموزی خصمانه(SAT) را با تابع فعالسازی Mish مروش های موزاییکی داده افزایی، متده متعادل سازی DropBlock و تابع زیان CIoU ترکیب شدند. این ویژگی ها، ویژگی های کاملی هستند، زیرا بدون وجود مدل ها، دیتاست ها و مسائل بینایی رایانه ای نیز به درستی کار می کنند. BoF از تاثیر YOLOv4 و چند BoS بهره برده است. BoF دقیق تشخیص دهنده را بدون افزایش inference time افزایش داده و تنها training cost افزایش پیدا کرده است. از طرف دیگر BoS سبب افزایش inference cost به مقدار کمی شده است اما دقیق تشخیص اشیا به صورت قابل ملاحظه ای افزایش پیدا کرده است.

## نسخه پنجم الگوریتم YOLO

اساسی ترین پیشرفتهای شامل auto learning bounding box anchors و mosaic data augmentation است. پیشرفتهای دیگر سرعت train شدن بالاتر نسخه پنجم نسبت به نسخه های قبلی می باشد. از آنجا که پیاده سازی YOLOv5 در PyTorch و Darknet در YOLOv4 است، ممکن است تولید YOLOv5 آسان تر باشد. مزیت اخر نسخه پنجم حجم پایین تر آن نسبت به مدل های قبلی می باشد.

---

۵) YOLOv3 یک فایل configuration دارد که می توانیم هایپر پارامتر های شبکه را متناسب با دیتاست دلخواه تنظیم نماییم. این هایپر پارامترها به شرح زیر هستند:

### Batch

پارامتر batch سایز دسته‌ی داده‌گان در هر iteration برای آموش و نهایتاً آپدیت شبکه در iteration مذکور را مشخص می‌نماید. در واقع ما در بحث آموزش ممکن است با میلیون‌ها تصویر سر و کار داشته باشیم و از نظر پیچیدگی زمانی و نیز سخت افزاری معقول نیست که ما وزن‌ها را در هر iteration فقط متناسب با یک تصویر آپدیت کنیم و علاوه بر آن نیز اصلاً این کار ضرورتی ندارد. در واقع ما آپدیت وزن‌ها را می‌توانیم در هر iteration و با تعداد مشخصی تصویر (نمونه‌ی ورودی) انجام دهیم. نحوه‌ی مشخص کردن این پارامتر متأثر از دو عامل (به عنوان مهم ترین عامل‌ها) است. اولین عامل پیچیدگی زمانی و دومین عامل پیچیدگی حافظه می‌باشد. باید توجه کنیم که هر اندازه سایز دسته کوچک باشد مسلماً تعداد iteration هایی که وزن‌ها در آن آپدیت می‌شوند افزایش یافته و زمان آموزش زیاد می‌شود و اما در عامل دوم که پیچیدگی حافظه است باید توجه کنیم که هر اندازه سایز دسته زیاد شود نتایج زیادی از تصاویر مورد ملاحظه قرار می‌گیرد که همین عامل پیچیدگی حافظه و در واقع RAM را مورد تاثیر قرار می‌دهد. همچنین در این حالت شبکه همگرایی نیز به میزان قابل توجهی تیز می‌شود و ممکن است نقطه‌ی بھینه مسئله را رد کند.

البته در زمانیکه سایز دسته کوچک باشد، ممکن است در بهینه‌های محلی گیر کنیم پس باید با در نظر داشتن این عامل‌ها حد وسطی را مناسب با تعداد نمونه‌ها انتخاب نماییم.

### Subdivisions

این هایپر پارامتر به جهت مدیریت حافظه برای batch size هایی است که سیستم قادر نیست با GPU موجود، این تعداد نمونه را در هر iteration بکار گیرد. به همین جهت مقدار subdivision در واقع کسری از سایز دسته را برای پردازش تصاویر در یک واحد زمانی GPU معین می‌کند. در رابطه با مقدار این هایپر پارامتر می‌توانیم در ابتدا با مقدار ۱ اقدام به آموزش تصویر کنیم و سپس در صورت بروز هشدار out of memory مقدار را به ۲ یا ۴ (بستگی به GPU مورد استفاده در سیستم) افزایش دهیم مادامی که فرایند آموزش با موفقیت صورت گیرد. دقت کنید که در هر حال آپدیت وزن‌ها پس از سپری شدن یک iteration و پردازش به اندازه تعداد سایز دسته صورت می‌گیرد.

### Width, height, channels

پارامتر عرض و ارتفاع که به سایز تصاویر مربوط می‌شود و پارامتر کanal هم به عمق تصویر (رنگی یا سیاه سفید) لذا با تنظیم دو پارامتر عرض و ارتفاع می‌توانیم سایز تصویر ورودی را به سایز تنظیم شده قبل از عملیات آموزش شبکه تغییر دهیم. در رابطه با مقدار این سه پارامتر باید نکاتی را توجه داشته باشیم. اول اینکه هر اندازه سایز تصویر زیاد و البته حاوی اطلاعات بیشتری باشد، مسلماً عملیات آموزش بهبود یافته و شبکه‌ی بهتری ساخته می‌شود ولی توجه داشته باشیم که در اینصورت زمان آموزش نیز افزایش خواهد یافت. همچنین در رابطه با تعداد کanal نیز مسلماً هر چه کanal ما بیشتر باشد مسلماً اطلاعات بیشتر است ولی مجدداً این افزایش اطلاعات هزینه‌ی زمانی را به دنبال خواهد داشت. توجه نمایید که تعداد کanal ۱ مبین سیاه و سفید و ۳ مبین RGB و ۲ نیز دو مولفه‌ی رنگ را نشان می‌دهد و البته ابعاد می‌توانند بستگی به کاربرد مانند پژوهشی بیشتر از ۳ تا هم باشند. لذا علاوه بر اطلاعات و پیچیدگی زمانی، می‌بایست به کاربرد سیستم هم توجه کنیم تا تعداد مناسبی را برای کanal‌ها ارائه دهیم.

### Momentum and decay

پارامترهای فوق برای کنترل و چگونگی آپدیت وزن‌ها به کار می‌روند. در واقع ما ذکر نمودیم که سایز دسته در آپدیت وزن‌ها چه تاثیری دارد ولی می‌دانیم که در هر iteration نمونه‌های متفاوتی به تعداد سایز دسته وارد می‌شود و آموزش در iteration مذکور با نمونه‌های جدید صورت می‌گیرد. این حقیقت گاهی اوقات منجر می‌شود که بین دو iteration تغییر زیادی در وزن قدیمی و آپدیت شده بوجود آید و لذا momentum را به جهت یک معیار بازدارنده برای اینگونه تغییرات به کار می‌برند. و اما decay یکی از دیگر پارامترهای کنترلی برای جلوگیری از بوجود آمدن overfitting بسیار مورد استفاده قرار می‌گیرد. در این موارد ممکن است شبکه تنها با داده‌های آموزش سازگار شود و به گونه‌ای این داده‌ها را حفظ نماید و نتیجتاً کارایی خوبی بر روی داده‌های تست از خود نشان ندهد. لذا این پارامتر یک عامل بازدارنده‌گی برای افزایش مقدار وزن‌ها می‌باشد. به طور معمول momentum را مقداری حدود ۰,۷ تا ۰,۹ و decay را مقداری بسیار کوچک (مثلاً ۰,۰۰۵) در نظر می‌گیرند.

## Data augmentation

پارامترهای بعدی مربوط به data augmentation می‌باشد. در واقع هدف از data augmentation ایجاد داده‌های جدید از نمونه‌های قدیمی و به عبارتی استفاده‌ی حداکثری از داده‌های موجود است. هایپرپارامترهای متعلق به این گروه که شامل angle, saturation, exposure و hue می‌شود در واقع هر یک به نوعی داده‌های جدید را ایجاد می‌کند. مثلاً با تغییر زاویه و ایجاد rotation می‌توانیم نمونه‌ی جدیدی را ایجاد کنیم که حاصل یکی از روش‌های data augmentation است. این هایپر پارامترها را با دو هدف در مدل تنظیم می‌کنیم. هدف اول تعمیم پذیری مدل به داده‌گان جدید و نویزی و به گونه‌ای عدم overfitting است و هدف دوم استفاده‌ی حداکثری از داده‌گان موجود در موقعیت کمبود دیتا می‌باشد. مقادیر هر یک از این هایپر پارامترها را می‌توان متناسب با کاربرد تغییر داد. مثلاً یک تصویر سیب را می‌توان با ۹۰ درجه تغییر جهت به چهار عکس تبدیل کرد.

## iteration و learning rate

پارامترهای دیگری مانند learning rate و iteration نیز قابل تنظیم هستند. Learning rate نرخی است که در آپدیت وزن‌ها بکار گرفته می‌شود و به طور معمول مقدار کوچکی دارد که جهش زیادی در وزن‌های جدید و قدیم بین دو iteration به وجود نماید و هم در واقع مراحلی است که در انتهای آن وزن‌ها آپدیت می‌شود و با سایز دسته رابطه دارد. باید توجه داشته باشیم که در مسائل multi-class

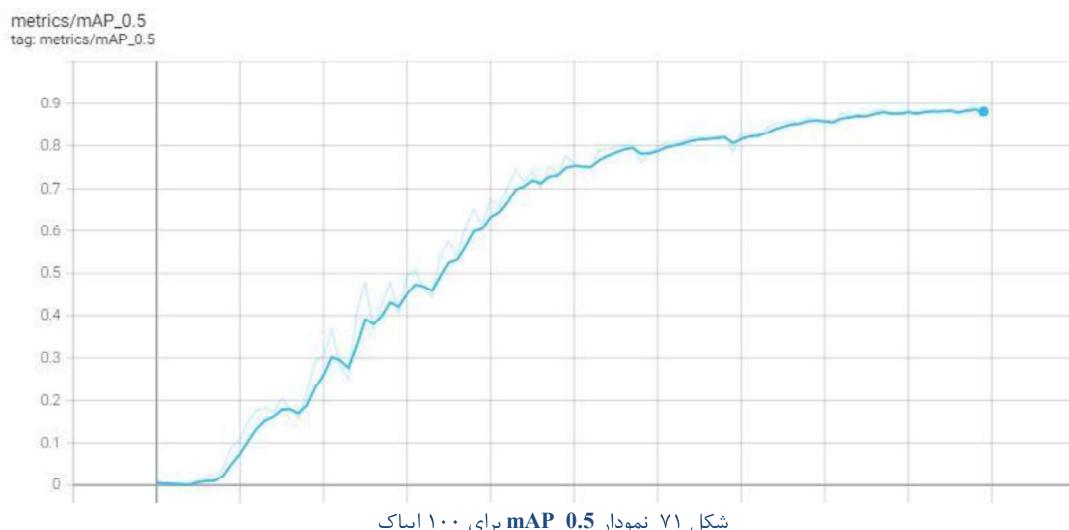
object detectors قابل تنظیم برای مدل max batch از بیشتر های ما در هر ایپاک iteration تعداد نباشد.

۶) از آنجاییکه حجم فایل دیتا زیاد بود، بنابراین ابتدا آن را در گوگل درایو بارگذاری نمودیم، سپس گوگل درایو را بر روی فایل نوت بوک مونت کردیم، این کار باعث می شود که نتایج هر RUN که میگیریم در گوگل درایو ذخیره شود. بنابراین بعد از هر بار آموزش وزن های بدست آمده در گوگل درایو ذخیره می شود، به علاوه ماتریس آشفتگی و نتایج دیگر نیز ذخیره خواهد شد، سپس آدرس پوشه ای را که در داخل گوگل درایو دیتا در آن ذخیره شده است را به نوت بوک می دهیم و سپس YOLOv5 را از گیتاب فراخوانی می کنیم و بعد آدرس پوشه ایجاد شده در گوگل درایو برای YOLOv5 را به نوت بوک می دهیم. در مرحله بعد می بایست YOLOv5 dependencies را نصب کنیم، و متعاقباً آدرسی را که دیتا در آن قرار گرفته است را بدهیم و دیتا را unzip نماییم. در ادامه کلاس های موجود در دیتاها را نمایش می دهیم که عبارتند از :

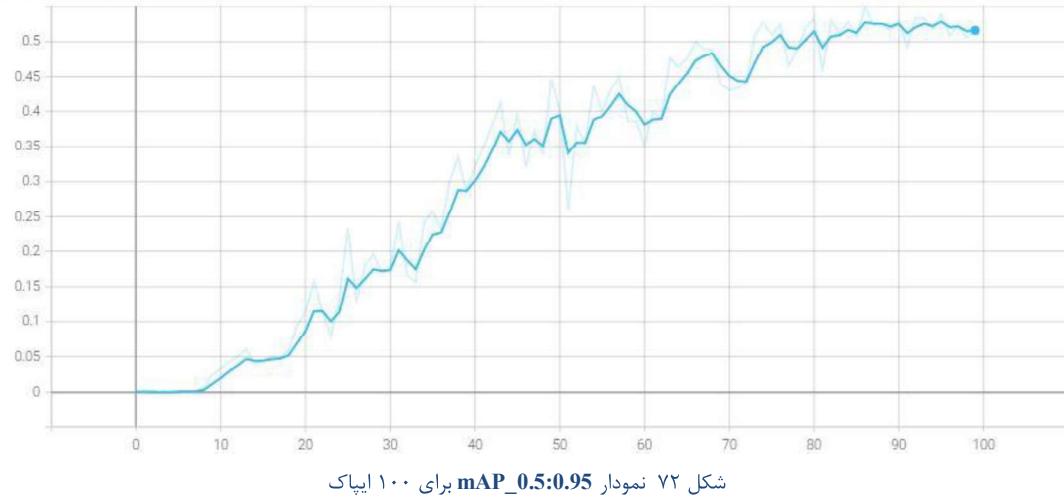
```
names: ['blue', 'green', 'red', 'vline', 'white', 'yellow']
```

سپس باید پیکربندی و معما را مشخص نماییم که در این گام تنها قسمتی که اصلاح می شود آدرس دهی پوشه مدل می باشد، در این گام ما پارامترهای مدل مشخص می شود. در گام بعدی باید مدل را آموزش دهیم پس آدرس پوشه YOLOv5 را می دهیم که نتیجه run در آن ذخیره شود و بعد ۱۰۰ ایپاک را تعیین می کنیم و مدل را آموزش می دهیم. که نمودارها به صورت مقابل بدست امده:

### نتایج شبکه YOLOv5s

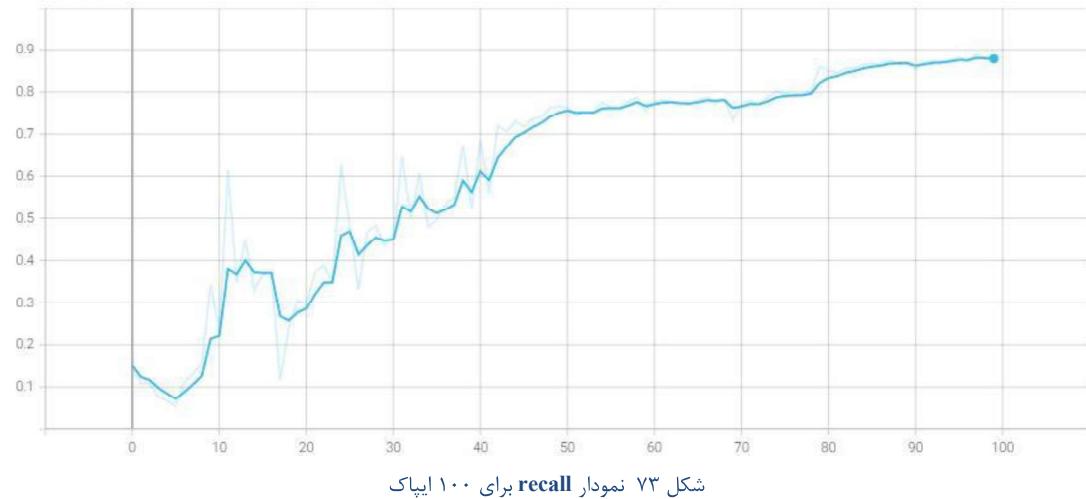


metrics/mAP\_0.5:0.95  
tag: metrics/mAP\_0.5:0.95

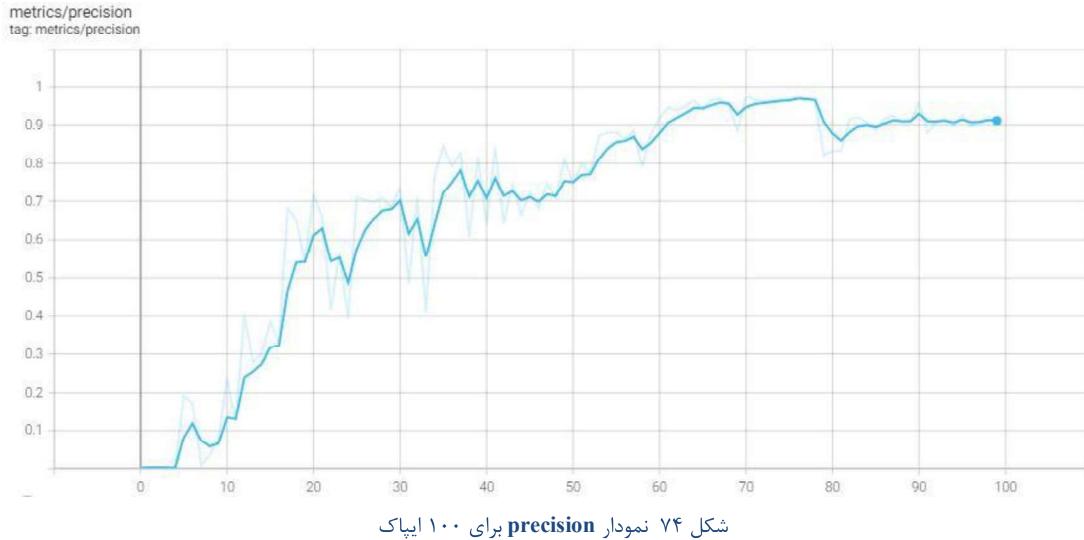


شکل ۷۲ نمودار mAP\_0.5:0.95 برای ۱۰۰ ایپاک

metrics/recall  
tag: metrics/recall



شکل ۷۳ نمودار recall برای ۱۰۰ ایپاک



```

Optimizer stripped from runs/train/yolov5s_results4/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results4/weights/best.pt, 14.8MB
100 epochs completed in 0.988 hours.

CPU times: user 23.9 s, sys: 3.85 s, total: 27.7 s
Wall time: 59min 55s

```

شکل ۷۵ زمان آموزش برابر با 59min 55sec

```

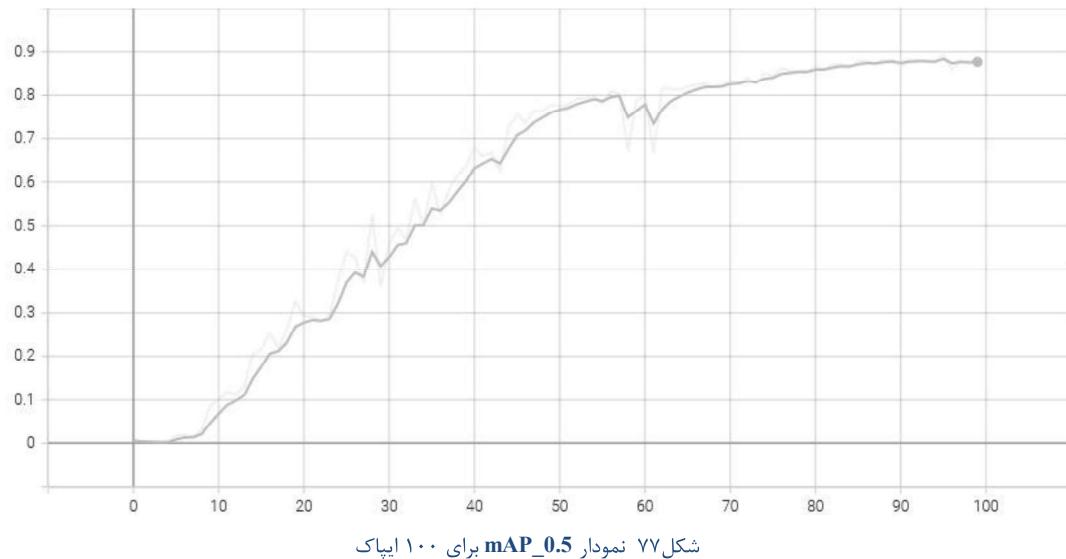
image 55/58 /content/gdrive/MyDrive/roboflow/v
image 56/58 /content/gdrive/MyDrive/roboflow/v
image 57/58 /content/gdrive/MyDrive/roboflow/v
image 58/58 /content/gdrive/MyDrive/roboflow/v
Results saved to runs/detect/exp8
Done. (2.672s)

```

شکل ۷۶ زمان تشخیص 2.672 s

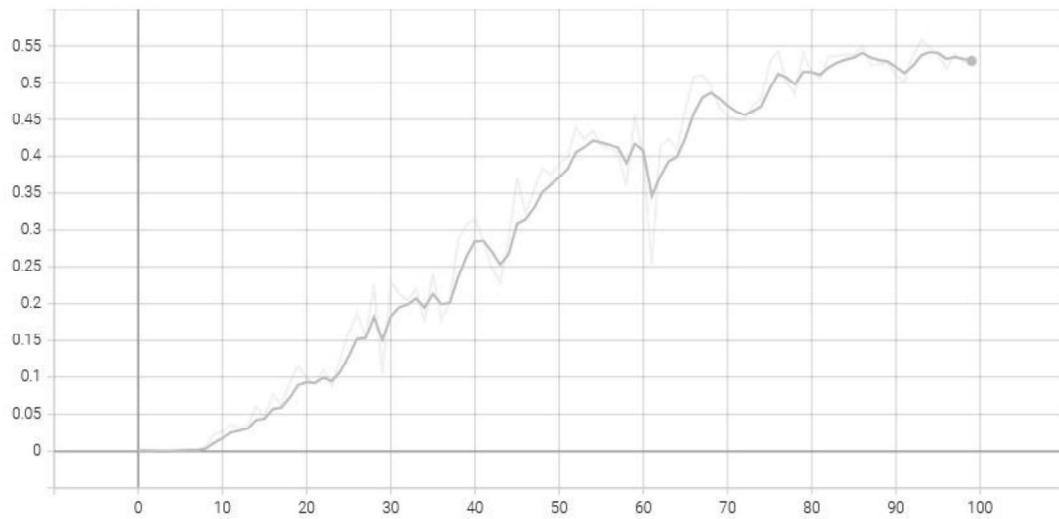
## نتایج شبکه YOLOv5x

metrics/mAP\_0.5  
tag: metrics/mAP\_0.5



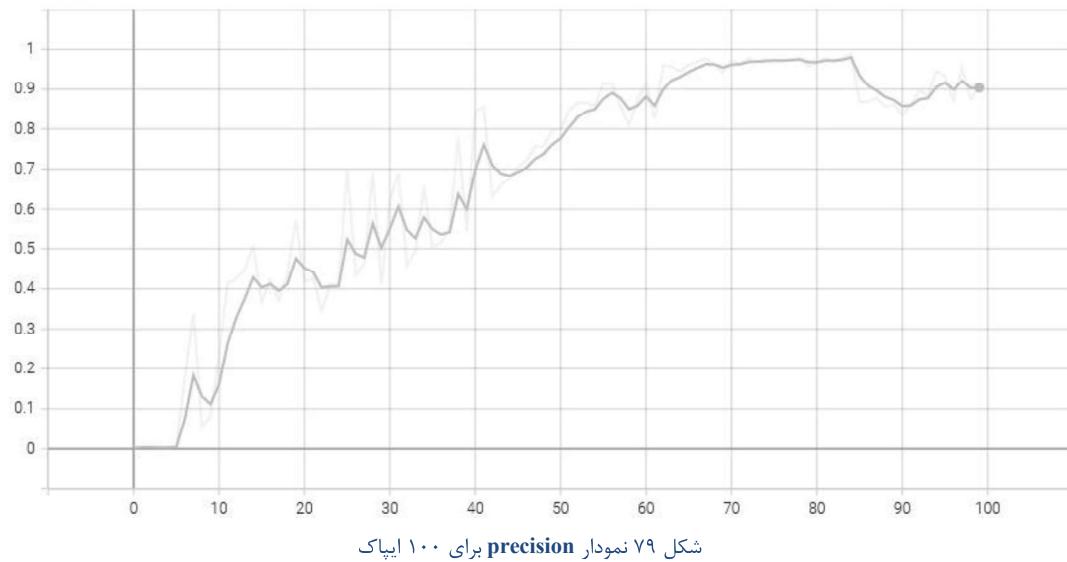
شکل ۷۷ نمودار mAP\_0.5 برای ۱۰۰ ایپاک

metrics/mAP\_0.5:0.95  
tag: metrics/mAP\_0.5:0.95



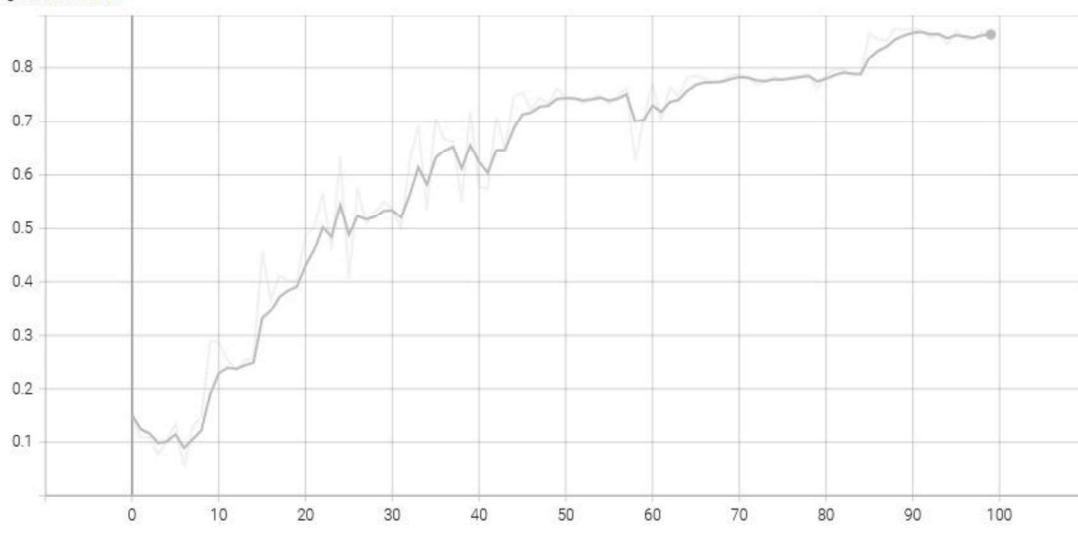
شکل ۷۸ نمودار mAP\_0.5:0.95 برای ۱۰۰ ایپاک

metrics/precision  
tag: metrics/precision



شکل ۷۹ نمودار precision برای ۱۰۰ ایپاک

metrics/recall  
tag: metrics/recall



شکل ۸۰ نمودار recall برای ۱۰۰ ایپاک

```

100 epochs completed in 0.953 hours.

CPU times: user 29.5 s, sys: 4.36 s, total: 33.8 s
Wall time: 57min 53s

```

شکل ۸۱ زمان آموزش ۵۸min ۵۳sec

```

image 55/58 /content/gdrive/MyDrive/runs/detect/exp5/images/00000.jpg
image 56/58 /content/gdrive/MyDrive/runs/detect/exp5/images/00001.jpg
image 57/58 /content/gdrive/MyDrive/runs/detect/exp5/images/00002.jpg
image 58/58 /content/gdrive/MyDrive/runs/detect/exp5/images/00003.jpg
Results saved to runs/detect/exp5.
Done. (2.805s)

```

شکل ۸۲ زمان تشخیص 2.805 sec

Table 21 زمان آموزش و زمان تشخیص دو شبکه

Network	Train Time	Detect Time
YOLOv5s	59 min 55 sec	2.672 sec
YOLOv5x	58 min 53 sec	2.805 sec

با توجه به نتایج بدست آمده به نظر می رسد YOLOv5s نسبت به YOLOv5x کمی بهتر عمل می کند، این بهبود عملکرد از نقطه نظر recall و precision و mAP\_0.5:0.95 و mAP\_0.5 با توجه به نتایج بدست آمده از مساله ملموس است. از نظر زمانی تقریبا مشابه همدیگر می باشند.

## مزایا و معایب استفاده از YOLOv5x نسبت به YOLOv5s

مدل بزرگتر (YOLOv5x) accuracy بهتری نسبت به مدل کوچک تر (YOLOv5s) دارد.

این دو مدل از منظر لایه ای تفاوتی ندارند، اما تفاوت آن ها در مقیاس پذیری عرض و عمق شبکه می باشد.

سرعت YOLOv5x نسبت به YOLOv5s بیشتر است، هم با استفاده از نتایج خودمان و هم با توجه به مقاله زیر:

Malta, Ana, Mateus Mendes, and Torres Farinha. "Augmented Reality Maintenance Assistant Using YOLOv5." *Applied Sciences* 11.11 (2021): 4758.