به نام خدا

**دانشگاه تهران**

**دانشکده مهندسی برق و کامپیوتر**

# شبکه های عصبی و یادگیری عمیق

## مینی پروژه اول

| | |
|---|---|
| نام و نام خانوادگی | مریم ریاضی |
| شماره دانشجویی | 810197518 |
| تاریخ ارسال گزارش | ۳ اردیبهشت ۱۴۰۱ |

## فهرست گزارش سوالات

# سوال ۱ – CNN(classification)

در این سوال از داده های Cifar 10 قرار است استفاده شود که شامل داده هایی با ۶۰۰۰۰ تصویر ۳۲*۳۲ هستند که در کل ۱۰ کلاس مختلف label دار شده است ئ هر کلاس ۶۰۰۰ نمونه دارد . ازاین ۶۰۰۰۰ داده ۵۰۰۰۰ داده برای train در نظر گرفته شده است و ۱۰۰۰۰ داده برای آزمون در نظر گرفته شده است.


شکل ۱ : نمونه های کلاس مجموعه داده cifar10

داده ها به ۵ batch که هر کدام ۱۰۰۰۰ تصویر در خود جای داده اند برای تمرین و یک batch که ۱۰۰۰۰ تصویر در خود جای داده است برای آزمون تقسیم شده اند .

داده های هر کدام از این batch ها 3072 * 10000 میباشد که ۱۰۰۰۰ تا کهنمونه های ما هستند و ۳۰۷۲ ستون به ترتیب ۱۰۲۴ ستون آن کانال قرمز آن تصویر ۱۰۲۴ ستون بعد آن کانال سبز آن تصویر و بعد آن کانال آبی آن تصویر میباشد .

هدف از این پارت توسعه گام به گام بخش استخراج ویژگی است .

**الف)**

برای قسمت اول گفته شده بود از conv استفاده کنیم و نمودار های خطا و دقت را برای مدل معرفی شده بکشیم :

مدل تعریف شده :

```
[32] model2 = Sequential([Input(shape = X_train[0].shape) ,Conv2D(16,(3,3), activation='relu',  padding="same" , input_shape=(32,32,1)),
                          Conv2D(32,(3,3), padding="same" , activation='relu'),

                          Conv2D(32,(3,3), padding="same" , activation='relu'),
                          Conv2D(64,(3,3), padding="same" , activation='relu'),

                          Conv2D(64,(3,3),  padding="same" ,activation='relu'),
                          Conv2D(128,(3,3), padding="same" , activation='relu'),
                          Flatten(),
                          Dense(128, activation='relu'),
                          Dense(10, activation='softmax')])
```

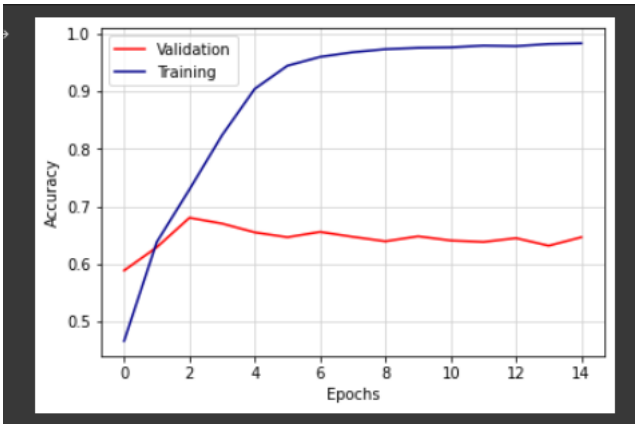
شکل ۲ : مدل تعریف شده

خلاصه ای از مدل در شکل ۳ ضمیمه شده است.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 32, 32, 16)        448

 conv2d_29 (Conv2D)          (None, 32, 32, 32)        4640

 conv2d_30 (Conv2D)          (None, 32, 32, 32)        9248

 conv2d_31 (Conv2D)          (None, 32, 32, 64)        18496

 conv2d_32 (Conv2D)          (None, 32, 32, 64)        36928

 conv2d_33 (Conv2D)          (None, 32, 32, 128)       73856

 flatten_5 (Flatten)         (None, 131072)            0

 dense_9 (Dense)             (None, 128)               16777344

 dense_10 (Dense)            (None, 10)                1290

=================================================================
Total params: 16,922,250
Trainable params: 16,922,250
Non-trainable params: 0
_________________________________________________________________
```

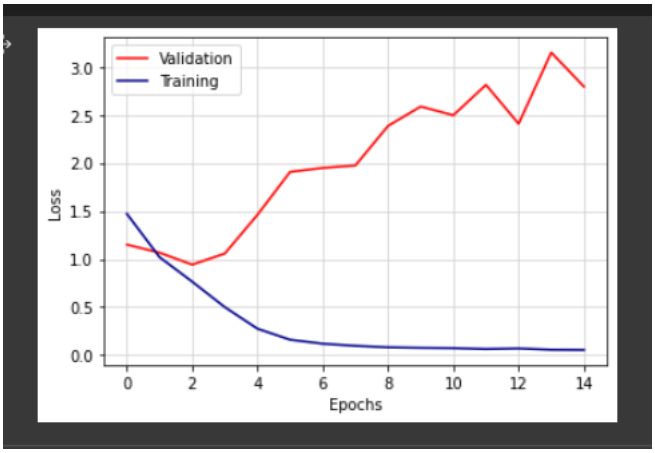
شکل ۳

نمودار خطا برای ۱۵ ایپاک در شکل ۴ قرار داده شده است.




شکل ۴

نمودار دقت برای ۱۵ ایپاک در شکل ۵ قرار داده شده است




شکل ۵

دقت و خطا برای  هر ایپاک :

```
Epoch 1/15
1563/1563 [==============================] - 57s 36ms/step - loss: 1.4700 - accuracy: 0.4662 - val_loss: 1.1496 - val_accuracy: 0.5885
Epoch 2/15
1563/1563 [==============================] - 56s 36ms/step - loss: 1.0190 - accuracy: 0.6380 - val_loss: 1.0663 - val_accuracy: 0.6289
Epoch 3/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.7645 - accuracy: 0.7293 - val_loss: 0.9414 - val_accuracy: 0.6801
Epoch 4/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.4992 - accuracy: 0.8236 - val_loss: 1.0574 - val_accuracy: 0.6700
Epoch 5/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.2749 - accuracy: 0.9041 - val_loss: 1.4605 - val_accuracy: 0.6547
Epoch 6/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.1605 - accuracy: 0.9442 - val_loss: 1.9068 - val_accuracy: 0.6463
Epoch 7/15
1563/1563 [==============================] - 57s 37ms/step - loss: 0.1192 - accuracy: 0.9596 - val_loss: 1.9475 - val_accuracy: 0.6556
Epoch 8/15
1563/1563 [==============================] - 56s 36ms/step - loss: 0.0971 - accuracy: 0.9677 - val_loss: 1.9737 - val_accuracy: 0.6470
Epoch 9/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0816 - accuracy: 0.9730 - val_loss: 2.3848 - val_accuracy: 0.6392
Epoch 10/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0761 - accuracy: 0.9753 - val_loss: 2.5863 - val_accuracy: 0.6479
Epoch 11/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0731 - accuracy: 0.9761 - val_loss: 2.4959 - val_accuracy: 0.6406
Epoch 12/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0638 - accuracy: 0.9791 - val_loss: 2.8120 - val_accuracy: 0.6381
Epoch 13/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0700 - accuracy: 0.9782 - val_loss: 2.4072 - val_accuracy: 0.6447
Epoch 14/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0563 - accuracy: 0.9819 - val_loss: 3.1487 - val_accuracy: 0.6315
Epoch 15/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.0545 - accuracy: 0.9831 - val_loss: 2.7937 - val_accuracy: 0.6463
```

شکل ۶

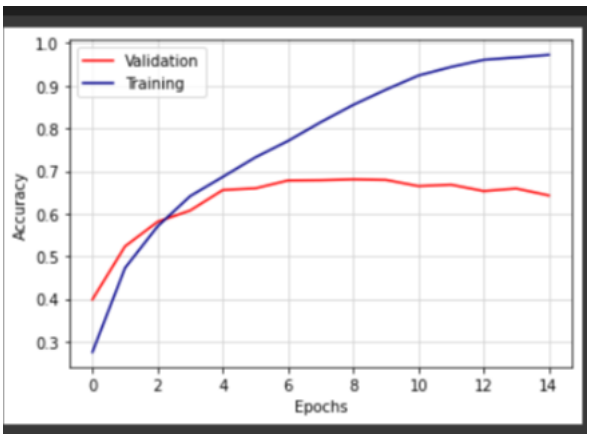
با توجه به نمودار های خطا در شکل ۵ میفهمیم که مدل در حالت overfit هست و  نمودار val-loss به صورت صعودی میباشد. برای رفع این مشکل تعداد نورون ها لایه desne آخر را از ۱۲۸ به ۳۲ کاهش دادیم و نتایج به صورت زیر شد :

```
modele = Sequential([Input(shape = X_train[0].shape) ,Conv2D(16,(3,3), activation='relu',  padding="same" , input_shape=(32,32,1)),
                     Conv2D(32,(3,3), padding="same" , activation='relu'),

                     Conv2D(32,(3,3), padding="same" , activation='relu'),
                     Conv2D(64,(3,3), padding="same" , activation='relu'),

                     Conv2D(64,(3,3),  padding="same" ,activation='relu'),
                     Conv2D(128,(3,3), padding="same" , activation='relu'),
                     Flatten(),
                     Dense(32, activation='relu'),
                     Dense(10, activation='softmax')])
```

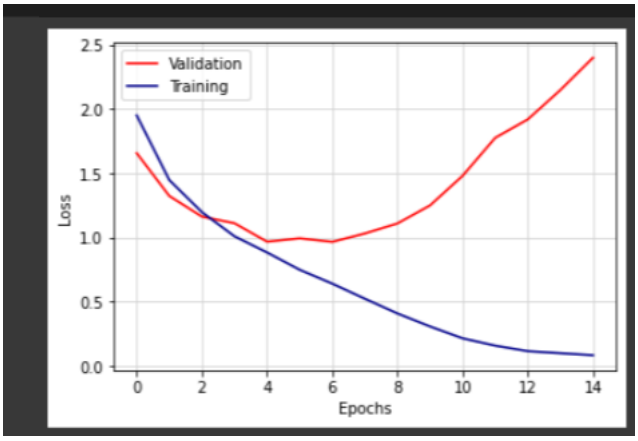
شکل ۷ – کد مدل تعریف شده مدل اصلاح شده

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 32, 32, 16)        448

 conv2d_29 (Conv2D)          (None, 32, 32, 32)        4640

 conv2d_30 (Conv2D)          (None, 32, 32, 32)        9248

 conv2d_31 (Conv2D)          (None, 32, 32, 64)        18496

 conv2d_32 (Conv2D)          (None, 32, 32, 64)        36928

 conv2d_33 (Conv2D)          (None, 32, 32, 128)       73856

 flatten_5 (Flatten)         (None, 131072)            0

 dense_10 (Dense)            (None, 32)                4194336

 dense_11 (Dense)            (None, 10)                330

=================================================================
Total params: 4,338,282
Trainable params: 4,338,282
Non-trainable params: 0
```

شکل ۸ – مشخصات و  خلاصه مدل اصلاح شده




شکل ۹ – نمودار دقت مدل اصلاحی




شکل ۱۰ – نمودار خطا مدل اصلاحی

```
Epoch 1/15
196/196 [==============================] - 48s 155ms/step - loss: 1.9495 - accuracy: 0.2754 - val_loss: 1.6554 - val_accuracy: 0.3988
Epoch 2/15
196/196 [==============================] - 30s 152ms/step - loss: 1.4455 - accuracy: 0.4727 - val_loss: 1.3216 - val_accuracy: 0.5235
Epoch 3/15
196/196 [==============================] - 30s 152ms/step - loss: 1.1959 - accuracy: 0.5704 - val_loss: 1.1610 - val_accuracy: 0.5811
Epoch 4/15
196/196 [==============================] - 31s 156ms/step - loss: 1.0077 - accuracy: 0.6413 - val_loss: 1.1094 - val_accuracy: 0.6076
Epoch 5/15
196/196 [==============================] - 30s 153ms/step - loss: 0.8815 - accuracy: 0.6865 - val_loss: 0.9655 - val_accuracy: 0.6556
Epoch 6/15
196/196 [==============================] - 31s 156ms/step - loss: 0.7471 - accuracy: 0.7323 - val_loss: 0.9918 - val_accuracy: 0.6595
Epoch 7/15
196/196 [==============================] - 30s 153ms/step - loss: 0.6390 - accuracy: 0.7707 - val_loss: 0.9642 - val_accuracy: 0.6780
Epoch 8/15
196/196 [==============================] - 30s 153ms/step - loss: 0.5216 - accuracy: 0.8146 - val_loss: 1.0297 - val_accuracy: 0.6786
Epoch 9/15
196/196 [==============================] - 30s 153ms/step - loss: 0.4068 - accuracy: 0.8553 - val_loss: 1.1078 - val_accuracy: 0.6806
Epoch 10/15
196/196 [==============================] - 30s 154ms/step - loss: 0.3056 - accuracy: 0.8909 - val_loss: 1.2480 - val_accuracy: 0.6793
Epoch 11/15
196/196 [==============================] - 30s 153ms/step - loss: 0.2132 - accuracy: 0.9242 - val_loss: 1.4790 - val_accuracy: 0.6647
Epoch 12/15
196/196 [==============================] - 30s 153ms/step - loss: 0.1562 - accuracy: 0.9443 - val_loss: 1.7745 - val_accuracy: 0.6680
Epoch 13/15
196/196 [==============================] - 30s 153ms/step - loss: 0.1133 - accuracy: 0.9610 - val_loss: 1.9182 - val_accuracy: 0.6531
Epoch 14/15
196/196 [==============================] - 30s 153ms/step - loss: 0.0974 - accuracy: 0.9666 - val_loss: 2.1468 - val_accuracy: 0.6590
Epoch 15/15
196/196 [==============================] - 31s 156ms/step - loss: 0.0819 - accuracy: 0.9726 - val_loss: 2.3982 - val_accuracy: 0.6428
```

شکل ۱۱ - اطلاعات مربوط به هر ایپاک مدل اصلاحی

**ب)** لایه batch normalization با استاندارد سازی داده های ورودی به آن باعث میشود سرعت شبکه بهبود یابد .

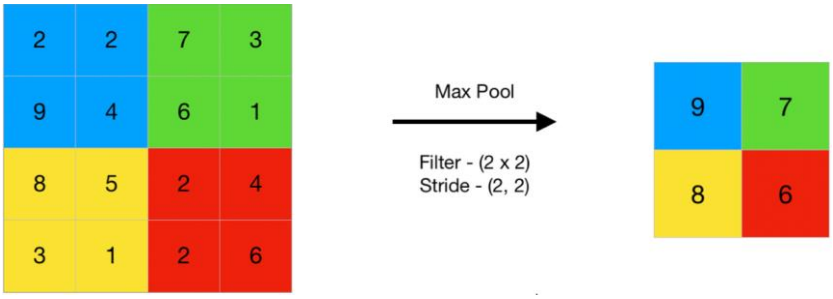
(تقریبا مانند feature scaling است و در واقع خروجی لایه قبلی را نرمالسازی میکند ( میانگین را کم میکند و بر انحراف معیار تقسیم میکند ) این لایه علاوه بر سرعت بخشیدن internal covariate shift را میتواند هندل کند

برای مثال :

فرض کنید لیستی از عکس هایی داریم که فقط میخواهیم لیبل ماشین و غیر ماشین را به آن بدهیم / حال فرض کنید که عکس های ما فقط شامل عکس های ماشین سفید هستند که distribution خود را دارند . حال فرض کنید به ما

مجموعه عکسی دادند که ماشین غیر از رنگ سفید دارد. این مدل دارای distributionهای متفاوتی نسبت به دیتا های قبلی هست و مدل با این دیتا ها پارامتر های خود را آپدیت میکند. از این رو توزیع فعال سازی لایه پنهان نیز تغییر خواهد کرد. این تغییر در فعال سازی لایه پنهان به عنوان یک شیفت کوواریات داخلی شناخته می شود.

یکی از مشکلات شبکه های عصبی حساس بودن به محل تصویر مورد نظر در تصویر است. برای حل این مشکل از روش های down sampling استفاده میکنیم . لایه ی pooling با نمونه برداری از تصویر اصلی این عمل را برای ما انجام میدهد. این تابع pooling آن این است که اندازه representationرا کاهش دهد تا پیچیدگی شبکه و هزینه محاسباتی کاهش دهد. Max pooling به سادگی یک قانون برای گرفتن حداکثر یک منطقه است و با مهم ترین ویژگی ها از تصویر کمک می کند(در واقع اثر مهم ترین ویژگی به طور واضح دیده میشود ). ... هم چنین میتواند سرعت مدل کردن را کاهش دهد.




خلاصه مدل برای این پارت در شکل ۱۲ نشان داده شده است:

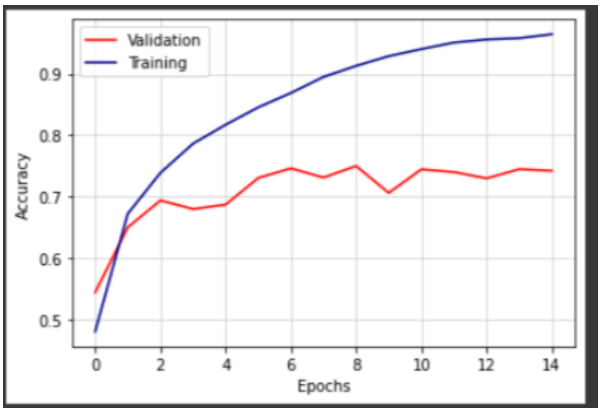
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 32, 32, 16)        448

 conv2d_41 (Conv2D)          (None, 32, 32, 32)        4640

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_42 (Conv2D)          (None, 16, 16, 32)        9248

 conv2d_43 (Conv2D)          (None, 16, 16, 64)        18496

 batch_normalization (BatchN  (None, 16, 16, 64)       256
 ormalization)

 conv2d_44 (Conv2D)          (None, 16, 16, 64)        36928

 conv2d_45 (Conv2D)          (None, 16, 16, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 flatten_7 (Flatten)         (None, 8192)              0

 dense_14 (Dense)            (None, 32)                262176

 dense_15 (Dense)            (None, 10)                330

=================================================================
Total params: 406,378
Trainable params: 406,250
Non-trainable params: 128
_________________________________________________________________
```

شکل ۱۲

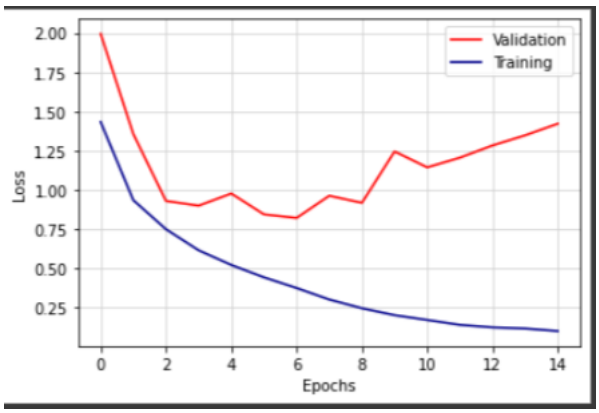
با توجه به نتایج میبینیم که با استفاده از لایه batch normalization سرعت مدل از هر ایپاک ۳۰ ثانیه به تقریبا هر ایپاک ۱۰ ثانیه رسیده است و سرعت مدل افزایش یافته است.

```
Epoch 1/15
196/196 [==============================] - 12s 60ms/step - loss: 1.4324 - accuracy: 0.4808 - val_loss: 1.9943 - val_accuracy: 0.5445
Epoch 2/15
196/196 [==============================] - 11s 58ms/step - loss: 0.9340 - accuracy: 0.6724 - val_loss: 1.3558 - val_accuracy: 0.6503
Epoch 3/15
196/196 [==============================] - 11s 57ms/step - loss: 0.7490 - accuracy: 0.7389 - val_loss: 0.9291 - val_accuracy: 0.6940
Epoch 4/15
196/196 [==============================] - 11s 57ms/step - loss: 0.6146 - accuracy: 0.7864 - val_loss: 0.8990 - val_accuracy: 0.6800
Epoch 5/15
196/196 [==============================] - 11s 57ms/step - loss: 0.5212 - accuracy: 0.8172 - val_loss: 0.9766 - val_accuracy: 0.6872
Epoch 6/15
196/196 [==============================] - 11s 57ms/step - loss: 0.4422 - accuracy: 0.8455 - val_loss: 0.8435 - val_accuracy: 0.7306
Epoch 7/15
196/196 [==============================] - 11s 57ms/step - loss: 0.3742 - accuracy: 0.8688 - val_loss: 0.8209 - val_accuracy: 0.7463
Epoch 8/15
196/196 [==============================] - 11s 58ms/step - loss: 0.3009 - accuracy: 0.8950 - val_loss: 0.9624 - val_accuracy: 0.7312
Epoch 9/15
196/196 [==============================] - 11s 57ms/step - loss: 0.2453 - accuracy: 0.9130 - val_loss: 0.9170 - val_accuracy: 0.7501
Epoch 10/15
196/196 [==============================] - 11s 58ms/step - loss: 0.2011 - accuracy: 0.9288 - val_loss: 1.2448 - val_accuracy: 0.7063
Epoch 11/15
196/196 [==============================] - 11s 57ms/step - loss: 0.1705 - accuracy: 0.9403 - val_loss: 1.1432 - val_accuracy: 0.7446
Epoch 12/15
196/196 [==============================] - 11s 57ms/step - loss: 0.1394 - accuracy: 0.9509 - val_loss: 1.2051 - val_accuracy: 0.7402
Epoch 13/15
196/196 [==============================] - 11s 57ms/step - loss: 0.1234 - accuracy: 0.9559 - val_loss: 1.2830 - val_accuracy: 0.7299
Epoch 14/15
196/196 [==============================] - 11s 57ms/step - loss: 0.1160 - accuracy: 0.9580 - val_loss: 1.3476 - val_accuracy: 0.7449
Epoch 15/15
196/196 [==============================] - 11s 58ms/step - loss: 0.0999 - accuracy: 0.9645 - val_loss: 1.4217 - val_accuracy: 0.7423
```

شکل ۱۳ – اطلاعات مربوط به هر ایپاک




شکل ۱۴ – دقت مدل




شکل ۱۵ – خطا مدل

با توجه به شکل های بالا میبینیم دقت برای validation از ۶۰ درصد به حدود ۷۰ درصد رسیده است و دقت افزایش یافته است . هم چنین در حالت قبل خطا برای هم validation هم training بیشتر از حالت این بخش بود هم چنین با دقت به شکل میتوانیم ببینیم که overfit برای مدل را نیز نسبت به حالت قبل بهتر شده است .

**ج)** در dropoutدر حین یادگیری به صورت تصادفی تعدادی از نورون ها را ignoreمی کنیم و در نتیجه در هر مرحله از یادگیری از زاویه ی دیدی متفاوت به نورون ها و داده های ورودی آن لایه نگاه می کنیم در نتیجه overfittingانجام نمی شود یا کمتر می شود.

لایه ی drop out با حذف رندوم تعدادی از داده ها در هر بار پردازش باعث میشود تا overfitting اتفاق نیفتد و به عمل یادگیری کمک کند . اما اگر نرخ آن خیلی بالا باشد میتواند باعث شود تا نتایج بدتر شوند . در این بخش این لایه را با نرخ ۰,۵ و ۰,۲۵ پردازش میکنیم. این لایه به منظور جلوگیری از overfit زودهنگام گذاشته میشود. به این صورت که یک ورودی دارد که یک عدد بین 0تا ،۱که در واقع یک احتمال است را می گیرد و به صورت تصادفی و با احتمال ورودی اش، تعدادی از لایه های شبکه را غیر فعال یا به اصطلاح drop میکند

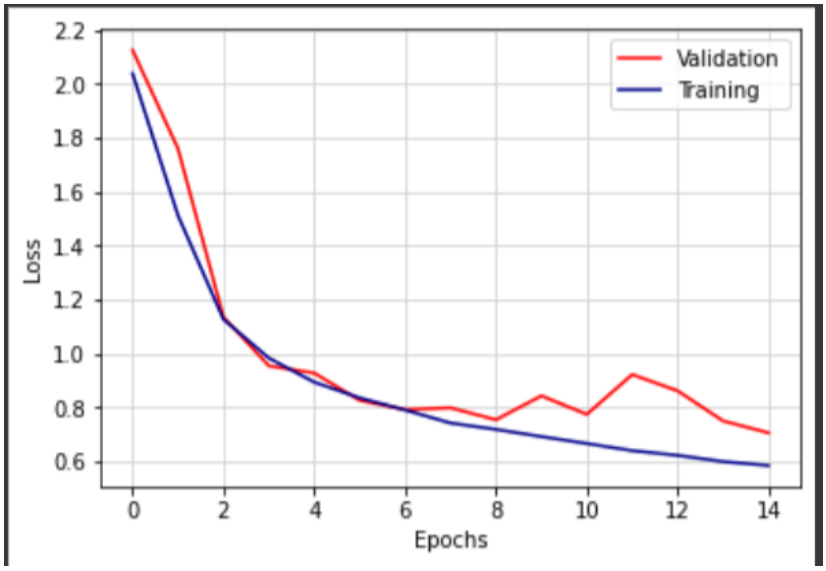
با dropout = 0.25:

```
55] model5 = Sequential([Input(shape = X_train[0].shape) ,Conv2D(16,(3,3), activation='relu',  padding="same" , input_shape=(32,32,1)),
                         Conv2D(32,(3,3), padding="same" , activation='relu'),
                         MaxPooling2D(2,2),
                         Dropout(0.25),

                         Conv2D(32,(3,3), padding="same" , activation='relu'),
                         Conv2D(64,(3,3), padding="same" , activation='relu'),
                         BatchNormalization(),
                         Dropout(0.25) ,

                         Conv2D(64,(3,3),  padding="same" ,activation='relu'),
                         Conv2D(128,(3,3), padding="same" , activation='relu'),
                         MaxPooling2D(2,2),
                         Dropout(0.25),
                         Flatten(),
                         Dense(32, activation='relu'),
                         Dense(10, activation='softmax')])
```

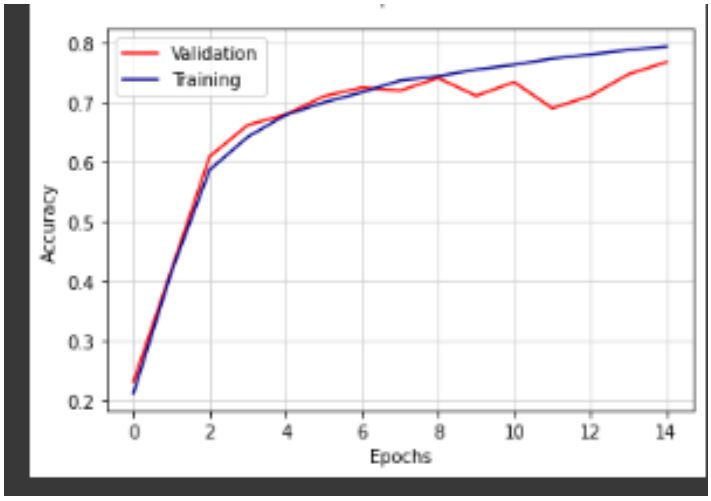
شکل ۱۶- مدل

```
Epoch 1/15
196/196 [==============================] - 13s 64ms/step - loss: 2.0389 - accuracy: 0.2117 - val_loss: 2.1261 - val_accuracy: 0.2311
Epoch 2/15
196/196 [==============================] - 12s 61ms/step - loss: 1.5119 - accuracy: 0.4161 - val_loss: 1.7601 - val_accuracy: 0.4195
Epoch 3/15
196/196 [==============================] - 12s 62ms/step - loss: 1.1266 - accuracy: 0.5869 - val_loss: 1.1351 - val_accuracy: 0.6095
Epoch 4/15
196/196 [==============================] - 13s 65ms/step - loss: 0.9824 - accuracy: 0.6424 - val_loss: 0.9536 - val_accuracy: 0.6624
Epoch 5/15
196/196 [==============================] - 12s 62ms/step - loss: 0.8929 - accuracy: 0.6794 - val_loss: 0.9267 - val_accuracy: 0.6800
Epoch 6/15
196/196 [==============================] - 12s 63ms/step - loss: 0.8347 - accuracy: 0.7004 - val_loss: 0.8254 - val_accuracy: 0.7108
Epoch 7/15
196/196 [==============================] - 12s 62ms/step - loss: 0.7895 - accuracy: 0.7174 - val_loss: 0.7910 - val_accuracy: 0.7254
Epoch 8/15
196/196 [==============================] - 12s 62ms/step - loss: 0.7412 - accuracy: 0.7371 - val_loss: 0.7977 - val_accuracy: 0.7202
Epoch 9/15
196/196 [==============================] - 12s 62ms/step - loss: 0.7176 - accuracy: 0.7445 - val_loss: 0.7533 - val_accuracy: 0.7416
Epoch 10/15
196/196 [==============================] - 12s 62ms/step - loss: 0.6909 - accuracy: 0.7556 - val_loss: 0.8422 - val_accuracy: 0.7115
Epoch 11/15
196/196 [==============================] - 12s 62ms/step - loss: 0.6651 - accuracy: 0.7638 - val_loss: 0.7735 - val_accuracy: 0.7347
Epoch 12/15
196/196 [==============================] - 12s 62ms/step - loss: 0.6386 - accuracy: 0.7741 - val_loss: 0.9214 - val_accuracy: 0.6906
Epoch 13/15
196/196 [==============================] - 12s 62ms/step - loss: 0.6212 - accuracy: 0.7810 - val_loss: 0.8608 - val_accuracy: 0.7114
Epoch 14/15
196/196 [==============================] - 12s 62ms/step - loss: 0.5980 - accuracy: 0.7886 - val_loss: 0.7490 - val_accuracy: 0.7478
Epoch 15/15
196/196 [==============================] - 12s 62ms/step - loss: 0.5835 - accuracy: 0.7944 - val_loss: 0.7049 - val_accuracy: 0.7684
```

شکل ۱۷ – اطلاعات برای هر ایپاک




شکل۱۸ – نمودار خطا




شکل ۱۹ – نمودار دقت

```
score, acc = model5.evaluate(X_test, y_test)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.7049 - accuracy: 0.7684
Test score: 0.7048894762992859
Test accuracy: 0.7684000134468079
```

شکل ۲۰ – نتایج تست مدل

همانطور که در شکل ۲۰ نشان داده شده است برای حالتی که dropout(0.25) استفاده شده است  دقت تقریبا ۷۷ درصد است و مقدار خطا ۰,۷ است . هم چنین با توجه به نمودار خطا  در شکل ۱۸ مقدا زیادی از  overfiting از بین رفته است و  در ایپاک های دیرتری شاهد  overfiting هستیم/ اما باید توجه داشت نسبت به مدل قسمت ب که اطلاعات آن در شکل ۲۱ آمده است مقدار دقت افزایش و مقدار خطا کاهش یافته است.

```
[67] score, acc = model4.evaluate(X_test, y_test)
     print('Test score:', score)
     print('Test accuracy:', acc)

     313/313 [==============================] - 2s 7ms/step - loss: 1.4217 - accuracy: 0.7423
     Test score: 1.4216575622558594
     Test accuracy: 0.7422999739646912
```

شکل۲۱

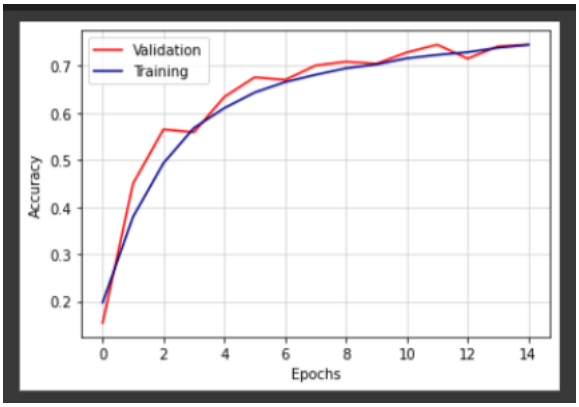
حال مقدار  dropoutرا  ۰,۵ قرار دادیم و مدل را تست کردیم :

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_52 (Conv2D)           (None, 32, 32, 16)        448

conv2d_53 (Conv2D)           (None, 32, 32, 32)        4640

max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)        0
2D)

dropout_3 (Dropout)          (None, 16, 16, 32)        0

conv2d_54 (Conv2D)           (None, 16, 16, 32)        9248

conv2d_55 (Conv2D)           (None, 16, 16, 64)        18496

batch_normalization_2 (Batc  (None, 16, 16, 64)        256
hNormalization)

dropout_4 (Dropout)          (None, 16, 16, 64)        0

conv2d_56 (Conv2D)           (None, 16, 16, 64)        36928

conv2d_57 (Conv2D)           (None, 16, 16, 128)       73856

max_pooling2d_7 (MaxPooling  (None, 8, 8, 128)         0
2D)

dropout_5 (Dropout)          (None, 8, 8, 128)         0

flatten_9 (Flatten)          (None, 8192)              0

dense_18 (Dense)             (None, 32)                262176

dense_19 (Dense)             (None, 10)                330

=================================================================
Total params: 406,378
Trainable params: 406,250
Non-trainable params: 128
_________________________________________________________________
```

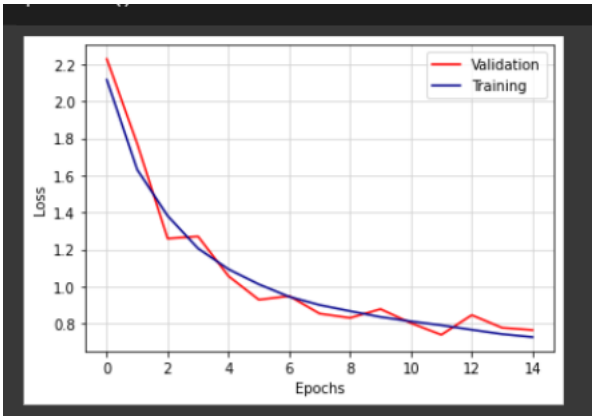
شکل ۲۲ – خلاصه مدل برای dropout(0.5)

```
Epoch 1/15
196/196 [==============================] - 13s 64ms/step - loss: 2.1158 - accuracy: 0.1971 - val_loss: 2.2280 - val_accuracy: 0.1540
Epoch 2/15
196/196 [==============================] - 12s 62ms/step - loss: 1.6304 - accuracy: 0.3795 - val_loss: 1.7701 - val_accuracy: 0.4502
Epoch 3/15
196/196 [==============================] - 12s 61ms/step - loss: 1.3813 - accuracy: 0.4934 - val_loss: 1.2588 - val_accuracy: 0.5652
Epoch 4/15
196/196 [==============================] - 12s 62ms/step - loss: 1.2055 - accuracy: 0.5681 - val_loss: 1.2703 - val_accuracy: 0.5589
Epoch 5/15
196/196 [==============================] - 12s 62ms/step - loss: 1.0933 - accuracy: 0.6102 - val_loss: 1.0560 - val_accuracy: 0.6341
Epoch 6/15
196/196 [==============================] - 12s 62ms/step - loss: 1.0119 - accuracy: 0.6436 - val_loss: 0.9284 - val_accuracy: 0.6759
Epoch 7/15
196/196 [==============================] - 12s 62ms/step - loss: 0.9445 - accuracy: 0.6655 - val_loss: 0.9483 - val_accuracy: 0.6703
Epoch 8/15
196/196 [==============================] - 12s 62ms/step - loss: 0.9001 - accuracy: 0.6812 - val_loss: 0.8534 - val_accuracy: 0.7008
Epoch 9/15
196/196 [==============================] - 12s 62ms/step - loss: 0.8671 - accuracy: 0.6949 - val_loss: 0.8303 - val_accuracy: 0.7090
Epoch 10/15
196/196 [==============================] - 12s 62ms/step - loss: 0.8353 - accuracy: 0.7030 - val_loss: 0.8782 - val_accuracy: 0.7045
Epoch 11/15
196/196 [==============================] - 12s 62ms/step - loss: 0.8115 - accuracy: 0.7160 - val_loss: 0.8023 - val_accuracy: 0.7285
Epoch 12/15
196/196 [==============================] - 12s 63ms/step - loss: 0.7907 - accuracy: 0.7233 - val_loss: 0.7391 - val_accuracy: 0.7452
Epoch 13/15
196/196 [==============================] - 12s 62ms/step - loss: 0.7663 - accuracy: 0.7290 - val_loss: 0.8456 - val_accuracy: 0.7154
Epoch 14/15
196/196 [==============================] - 12s 62ms/step - loss: 0.7427 - accuracy: 0.7385 - val_loss: 0.7770 - val_accuracy: 0.7415
Epoch 15/15
196/196 [==============================] - 12s 62ms/step - loss: 0.7265 - accuracy: 0.7447 - val_loss: 0.7649 - val_accuracy: 0.7454
```

شکل ۲۳ – اطلاعات هر ایپاک برای dropout(0.5)




شکل ۲۴ – نمودار دقت برای dropout(0.5)




شکل ۲۵ – نمودار خطا برای dropout(0.5)

```
313/313 [==============================] - 3s 8ms/step - loss: 0.7649 - accuracy: 0.7454
Test score: 0.764855682849884
Test accuracy: 0.7454000115394592
```

شکل ۲۶ – نتایج تست مدل

برای حالت ۰,۵ نیز همه اتفاقاتی که در ۰,۲۵ بود اتفاق افتاده است ولی علاوه بر آن overfit در شکل ۲۵ کاملا از بین رفته و احتمالا در ایپاک های خیلی بالا رخ میدهد و یا اصلا رخ نمیدهد.

( هم چنین باید توجه کرد در این حالت ۰,۵ دقت کمی از حالت ۰,۲۵ کاهش یافته و از ۷۷ درصد به ۷۵ درصد رسیده است و مقدار خطا از ۰,۷ به ۰,۷۶ رسیده است که مقدار این تغییرات بسیار ناچیز است که میتواند مورد قبول باشد .)

د)

توقف زود هنگام :

این روش شایع ترین روش است که برای از بین بردن overfitting استفاده میشود که همیشه در مدل های iterative استفاده میشود به این صورت که کدل پس از چند مرحله آموزش روی داده های آموزش بیش از اندازه fit میشود و خاصیت general بودن خود را از دست میدهد پس برای جلوگیری از این اتفاق از داده های validation استفاده میکنیم و در هر ایپاک عملکرد مدل را روی داده های validation چک میکنیم و اگر مشاهده شد که با ادامه ی یادگیری روی train بهبودی روی validation صورت نگرفته است و یا حتی بدتر شده است متوجه میشویم که باید در مرحله ی قبل آموزش متوقف میشده است و باید early stopping ( توقف زود هنگام را انجام دهیم )

در ادامه این فرآیند روی مدل پیاده سازی شده است که میبینیم نمودار های دقت و خطا smooth تر شده اند و در حقیقت نوسانات کمتری در آن ها ایجاد میشود.

```
callback = tf.keras.callbacks.EarlyStopping(patience=5,restore_best_weights=True)
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', baseline=0.4)
```

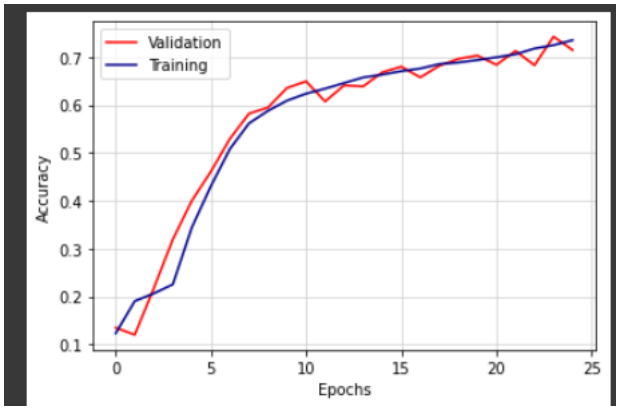
شکل ۲۷ – کد استفاده شده برای early stopping

در کد شکل ۲۷ مشخص کردیم که اگر بعد از ۵ ایپاک آثار بهبودی مدل را ندید training را متوقف کند.

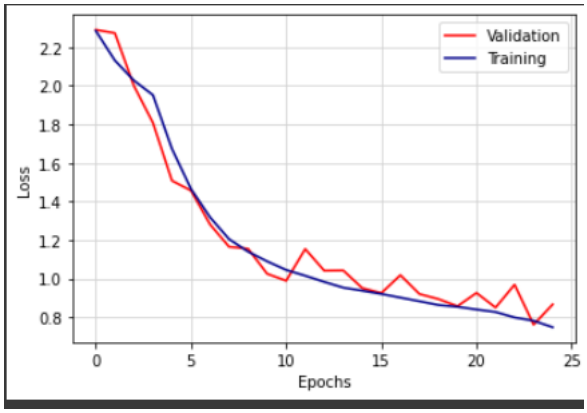
توجه شود برای این پارت تعداد ایپاک ها را زیاد کردیم.

```
Epoch 10/25
196/196 [==============================] - 12s 61ms/step - loss: 1.0900 - accuracy: 0.6089 - val_loss: 1.0256 - val_accuracy: 0.6353
Epoch 11/25
196/196 [==============================] - 12s 62ms/step - loss: 1.0453 - accuracy: 0.6233 - val_loss: 0.9896 - val_accuracy: 0.6490
Epoch 12/25
196/196 [==============================] - 12s 61ms/step - loss: 1.0155 - accuracy: 0.6338 - val_loss: 1.1541 - val_accuracy: 0.6070
Epoch 13/25
196/196 [==============================] - 12s 61ms/step - loss: 0.9835 - accuracy: 0.6456 - val_loss: 1.0417 - val_accuracy: 0.6408
Epoch 14/25
196/196 [==============================] - 12s 62ms/step - loss: 0.9537 - accuracy: 0.6572 - val_loss: 1.0428 - val_accuracy: 0.6385
Epoch 15/25
196/196 [==============================] - 12s 62ms/step - loss: 0.9378 - accuracy: 0.6631 - val_loss: 0.9513 - val_accuracy: 0.6678
Epoch 16/25
196/196 [==============================] - 12s 62ms/step - loss: 0.9210 - accuracy: 0.6703 - val_loss: 0.9254 - val_accuracy: 0.6797
Epoch 17/25
196/196 [==============================] - 12s 62ms/step - loss: 0.9021 - accuracy: 0.6758 - val_loss: 1.0187 - val_accuracy: 0.6572
Epoch 18/25
196/196 [==============================] - 12s 61ms/step - loss: 0.8831 - accuracy: 0.6851 - val_loss: 0.9208 - val_accuracy: 0.6815
Epoch 19/25
196/196 [==============================] - 12s 62ms/step - loss: 0.8637 - accuracy: 0.6883 - val_loss: 0.8945 - val_accuracy: 0.6960
Epoch 20/25
196/196 [==============================] - 12s 62ms/step - loss: 0.8544 - accuracy: 0.6935 - val_loss: 0.8577 - val_accuracy: 0.7028
Epoch 21/25
196/196 [==============================] - 12s 62ms/step - loss: 0.8404 - accuracy: 0.6992 - val_loss: 0.9266 - val_accuracy: 0.6835
Epoch 22/25
196/196 [==============================] - 12s 62ms/step - loss: 0.8274 - accuracy: 0.7055 - val_loss: 0.8504 - val_accuracy: 0.7125
Epoch 23/25
196/196 [==============================] - 12s 62ms/step - loss: 0.7990 - accuracy: 0.7175 - val_loss: 0.9693 - val_accuracy: 0.6826
Epoch 24/25
196/196 [==============================] - 12s 62ms/step - loss: 0.7820 - accuracy: 0.7245 - val_loss: 0.7619 - val_accuracy: 0.7425
```

شکل ۲۸ – اطلاعات هر ایپاک



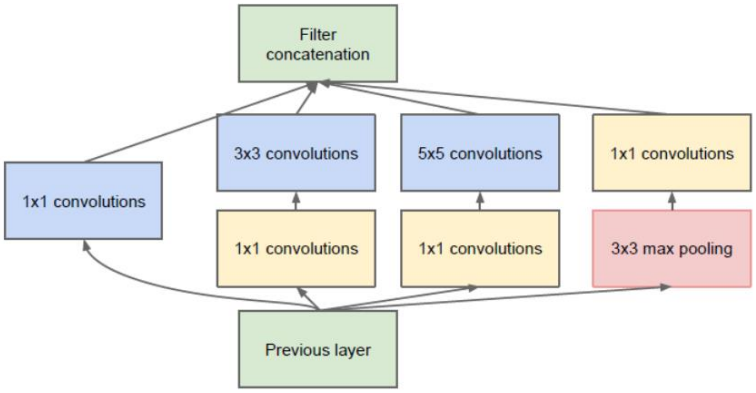

شکل ۲۹ – نمودار دقت




شکل ۳۰ – نمودار خطا

## سوال ۲ – transfer learning

در این سوال مدل انتخاب شده مدل inception است. این مدل، یک مدل محاسباتی برای استفاده در دسته بندی تصاویر است. نسخه اولیه آن یعنی inception v1 در سال ۲۰۱۴ به عنوان GoogLeNet معرفی شد. بعد از اعمال چند تغییر برای بهبود عملکرد و سرعت این مدل، inveption v3 حاصل شد. مدل اولیه به صورت شکل زیر است.




شکل ۳۱: معماری مدل inception v3

مدل inception v3 دارای ۴۲ لایه است و خطای کمتر و دقت بیشتری نسبت به مدل اولیه دارد. عمق این مدل بیشتر است که در عین حال باعث کاهش سرعت آن نشده است.

مدل نهایی به صورت زیر است:

| TYPE | PATCH / STRIDE SIZE | INPUT SIZE |
|---|---|---|
| Conv | 3×3/2 | 299×299×3 |
| Conv | 3×3/1 | 149×149×32 |
| Conv padded | 3×3/1 | 147×147×32 |
| Pool | 3×3/2 | 147×147×64 |
| Conv | 3×3/1 | 73×73×64 |
| Conv | 3×3/2 | 71×71×80 |
| Conv | 3×3/1 | 35×35×192 |
| 3 × Inception | Module 1 | 35×35×288 |
| 5 × Inception | Module 2 | 17×17×768 |
| 2 × Inception | Module 3 | 8×8×1280 |
| Pool | 8 × 8 | 8 × 8 × 2048 |
| Linear | Logits | 1 × 1 × 2048 |
| Softmax | Classifier | 1 × 1 × 1000 |

شکل ۳۲: لایه های مدل inception v3

تغییراتی که بر روی مدل اولیه انجام شده اند، عبارت است از:

۱. کوچک کردن کانولوشن های بزرگ: در مدل اولیه یک کانولوشن ۵*۵ وجود دارد که هزینه بسیاری را برای مدل به دنبال دارد. این کانولوشن در مرحله اول با دو کانولوشن ۳*۳ جایگزین می شود. این کار باعث کاهش چشم گیر در تعداد پارامترهای لازم می شود.
۲. جایگزینی کانولوشن های دو بعدی با کانولوشن های تک بعدی: قدم قبلی باعث کاهش تعداد پارامترها شد اما میتوان مدل را از این نیز بهتر کرد. برای این کار کانولوشن های دو بعدی که در مدل قرار داردند را با کانولوشن های تک بعدی یعنی با ابعاد n*۱ جایگزین می کنیم. برای مثال کانولوشن ۳*۳ به دو کانولوشن ۱*۳ و ۳*۱ تبدیل می شود. مدل حاصل شده با این کار ۳۳٪ از مدل قبلی ارزان تر است.
۳. اضافه کردن :Auxiliary classifiersدر واقع Auxiliary classifiers ها کامپوننت هایی هستند که قبل از آخرین لایه مدل اضافه می شوند. کارکرد آن ها این است که همگرایی را در شبکه های عمیق افزایش دهند. استفاده کردن این شبکه در مدل inception باعث شده به عنوان یک تنظیم کننده عمل کند
۴. کاهش ابعاد

تفاوت inception v2 , inception v3 در این موارد است:

۱. در inception v3 از RMSprop ب عنوان بهینه ساز استفاده شده است.
۲. از کانولوشن های ۱*۷ برای کاهش پیچیدگی استفاده شده.
۳. در لایه های Auxiliary classifiers از Batch Normalization استفاده شده است.
۴. از متد Label Smoothing Regularization استفاده شده است. این متد باعث می شود که مدل یک دسته بندی را بیش از حد پیش بینی نکند.

مزایا:

۱) این مدل در مقایسه با نسخه های قبلی یعنی inception v1, inception v2 دقت بسیار بیشتری دارد.
۲) این مدل یک شبکه عصبی عمیق تری نسبت به نسخه های قبلی است اما سرعت آن کاهش پیدا نکرده است.
۳) عملکرد بهینه تری دارد.

۴) پیچیدگی محاسباتی کمتری دارد.
۵) از Auxiliary classifiers به عنوان تنظیم کننده استفاده می کند.

پیش پردازش های لازم:

۱) (1سایز تصویر را به ۲۹۹*۲۹۹ تغییر می دهیم.
۲) (2یک بعد در راستای ۰ اضافه می کنیم.
۳) (3از تابع preprocess_input برای پیش پردازش عکس ها استفاده می کنیم. این تابع مقدار تمام پیکسل ها را بین -۱ , ۱ تغییر می دهد.

ب)

با توجه به آنچه در درس آموختیم، یکی از روش های یادگیری در شبکه های CNN استفاده از متد transfer learning است. این متد به این معنی است که برای داده های کوچک و منابع محدود از مدل هایی که قبلا ترین شده اند استفاده شود. علت این کار این است که آموزش دادن یک مدل پیچیده و عظیم، هزینه و زمان زیادی لازم دارد که برای اهداف کوچک و دیتاست های محدود خرج کردن چنین هزینه هایی ممکن نیست؛ بنابراین برای ساده تر کردن کار و در عین حال رسیدن به هدف مطلوب از مدل هایی استفاده می شود که قبلا روی دیتاست های بزرگ آموزش دیده اند و ضرایب درستی دارند. نکته مهم در استفاده از این روش این است که در هنگام استفاده از سایر مدل ها ما تنها می توانیم لایه های filter مدل ها را استفاده کنیم. یعنی لایه هایی که استخراج ویژگی را انجام می دهند و ضرایب را به روز می کنند. لایه های fully connected که در انتهای مدل قرار می گیرند وابسته به جنس دیتاست و تعداد کلاس های آن هستند و باید منحصرا تعریف شوند.

ج)

مدل inception v3 بر روی دیتاست عظیم ImageNet ترین شده است. در این دیتاست تعداد زیادی از حیوانات آورده شده اند. همچنین اشیا مختلفی در این دیتاست حضور دارند. اعضای بدن انسان و مکان هایی مانند نانوایی و فعالیت هایی مانند اسکی هم در این دیتاست لحاظ شده اند. این دیتاست حتی نژاد های مختلف سگ را هم به تفکیک در دسته بندی خود لحاظ کرده است.

در ابتدا یک تصویر از دیتاست imagenet دانلود کرده که به صورت زیر است.

شکل۳۳: عکس استفاده شده برای پیاده سازی inception v3

در مرحله بعد پیش پردازش ها را طبق مراحلی که در قسمت الف گفته شد، انجام دادیم.

برای آموزش دادن مدل با استفاده از مدل inception v3 این مدل را از کتابخانه keras آوردیم.

نتایج پیش بینی و سه کلاس اول با بیشترین احتمال به صورت زیر هستند.

```
golden_retriever (56.87%)
Labrador_retriever (15.79%)
English_foxhound (3.04%)
```

شکل ۳۴: نتایج پیش بینی مدل inception v3

د)

در این بخش دیتاست انتخابی همان دیتاست cifar10 است. در این قسمت ابتدا پس از نرمالایز کردن عکس ها با استفاده از ImageDataGenerator عکس ها را افزایش دادم.

برای این قسمت مدل را به صورت زیر وارد کردم.

```
pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = 'imagenet')
```

شکل ۳۵: وارد کردن مدل inception v3

در این قسمت پارامتر include_top را برابر false قرار دادم به این منظور که لایه های انتهایی مدل که لایه های fully conncted هستند غیرفعال شوند و در ادامه لایه های انتهایی را خودم به مدل اضافه کردم.

## سوال ۳ – segmentation

توضیح معماری شبکه Deep Lab :

در این مدل از DCNN برای semantic segmentation استفاده شده است. این نوع مدل ها برای تبدیل تصویر ورودی به یک تصویر انتزاعی عملکرد خوبی برای کلاس بندی دارند. ولی در semantic segmentation ممکن است عملکرد بدی از خود نشان دهند.

به همین دلیل برای از بین بردن این مشکلات راه حل هایی پیشنهاد دادند که موجب ساخت مدل deeplab گشت.

اولین مشکل که به وجود آمده بود وجود اشیا مختلف با سایز های مختلف بود برای حل آن با استفاده از ایده spatial pyramid pooling روشی ارایه شد که با نمونه برداری دوباره از یک لایه ویژگی ها با نرخ های مختلف قبل از کانولوشن بتواند این مشکل را حل کند.

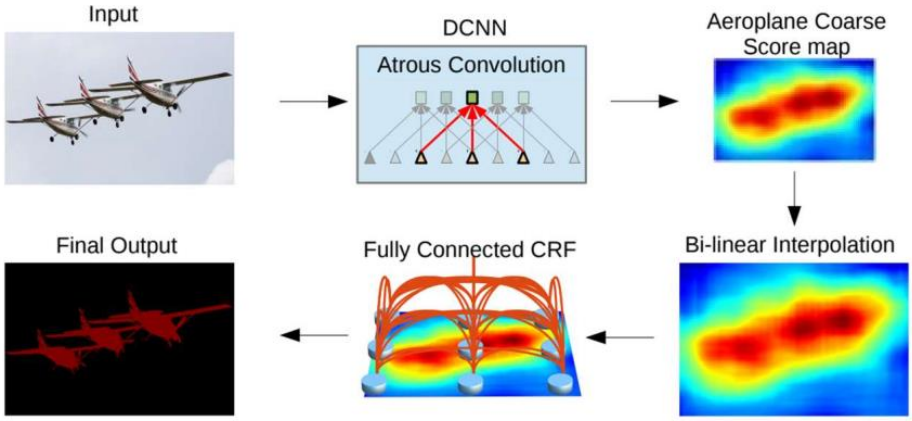
ولی به ازای نمونه برداری دوباره هزینه زیاد میشد بنابراین کاری که کردند لایه های کانولوشنی موازی ایجاد کردند که نرخ نمونه برداری در لایه های مختلف آن متفاوت بود( لایه های atrous )این روش را ASSP ( atrous spatial pyramid pooling ) نامیدند.

دومین مشکل که ناشی از تکرار ترکیب max pooling , down sampling به وجود میآید .این عمل باعث کاهش رزولوشن تصویر میشود . در مدل deeplab برای حل این مشکل لایه های down sampling از چند لایه آخر max pooling برداشته شده است و up sampling جایگزین آن ها شده است.در نتیجه feature map با نرخ نمونه برداری بالاتری استفاده شده است.

مشکل سوم

مدل باید برای اشکالی که تغییر وضعیت داده اند ( چرخیده اند) نیز تشخیص خوبی بدهد برای حل این مشکل سعی شده قابلیت مدل در یافتن جزییات ریز را با استفاده از CRF ( conditional random filed )افزایش داد .

CRF : با ترکیب نمره کلاس ها و کلاس بندی های انجام شده روی super pixle ها باعث میشود مرز تمیزتر و بدون نویز تری ایجاد کند




شکل ۳۶ - مدل deep lab

منبع: [DeepLab Explained | Papers With Code]

معماری مدل FCN:

در هر مدلی که می خواهد عمل semantic segmentation را انجام دهد از دو بخش استفاده می شود. این دو بخش downsampling, upsampling هستند.

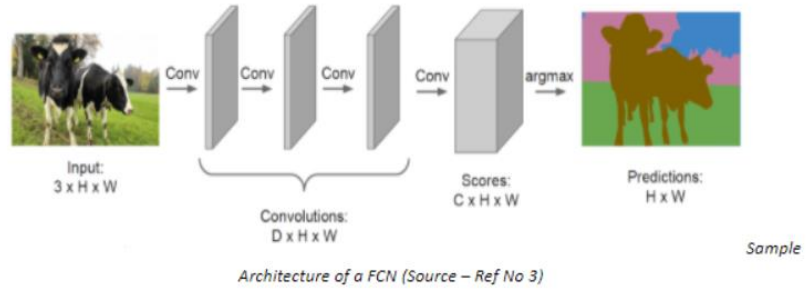
ایده اولیه ای که برای segmentation مطرح شد این بود که هر پیکسل از عکس ه صورت جداگانه بررسی شود و کلاس بندی آن تعیین گردد. این ایده دو مشکل عمده داشت. اول اینکه هزینه محاسباتی این کار بسیار زیاد است زیرا نیاز است که لایه های dense با تعداد نورون های بسیار و به تعداد پیکسل های عکس استفاده شوند. مشکل دوم یکسان نبودن سایز تصاویر است و به همین دلیل نمیتوان تعیین کرد که تعداد نورون های لایه های dense باید چقدر باشند.

راه حل بیان شده برای این مشکل همان طراحی دو مرحله ای است.

مرحله اول: این مرحله همان مرحله downsampling است. در این مرحله ابعاد عکس کوچک شده و به هر پیکسل از عکس عددی نسبت داده می شود. این عدد احتمال تعلق آن پیکسل به کلاسی خاص است. برای طراحی این مرحله از مدل هایی که برای classification استفاده می شود مانند ResNet, DenseNet استفاده می شود. در واقع انگار که این مرحله به خودی خود نوعی classification است. همانطور که در سوال دوم دیدیم در این مرحله از لایه هایی مانند convoloution , pooling استفاده می شود.

مرحله دوم: این مرحله همان مرحله upsampling است. در این مرحله سعی می شود که سایز عکس را به سایز اولیه تصویر ورودی برسانند. برای این کار از لایه هایی مانند transposed convoloution استفاده می شود. کاری که در این مرحله صورت می گیرد در واقع همان localization است. در نهایت هر پیکسل از عکس به عددی نسبت داده می شود که بیشترین احتمال تعلق آن به کلاسی خاص است.

در شبکه FCN در مرحله اول ابعاد عکس از W*H به W/2*H/2 و سپس به W/4*H/4 تغییر می کند. در مرحله دوم نیز ابعاد برعکس مرحله اول تغییر می کنند تا در نهایت به ابعاد اولیه برگردند.




*Architecture of a FCN (Source – Ref No 3)*

شکل ۳۷: معماری مدل FCN

مقایسه معماری DeepLab, FCN:

با توجه به توضیحات معماری هر دو مدل به نظر می رسد که DeepLab دقت و سرعت بیشتری داشته باشد. زیرا در این مدل مشکلات متعددی از جمله چرخش اشیا، اشیایی با ابعا خیلی بزرگ و یا خیلی کوچک و غیره را پوشش داده. در این مدل از ساختارهای بهینه تری برای محاسبات و تشخیص نیز استفاده شده است مانند Atrous convolutions.

برای پیاده سازی این مدل ها بر روی عکس دلخواه، از pretrained models های موجود در کتابخانه pytorch استفاده شد. در توضیح این مدل ها گفته شده که این مدل ها بر روی دیتاست COCO آموزش دیده اند و قادر به تشخیص ۲۱ کلاس هستند. این کلاس ها عبارتند از :

```
sem_classes = [
background__', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',__'
car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike','
person', 'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor"  ]
```

با توجه به این کلاس ها که قدرت تشخیص مدل ها را نشان می دهد، دو عکس برای امتحان کردن روی مدل های انتخاب شد که به صورت زیر هستند.

شکل ۳۸: تصویر اول امتحان شده (ماشین)

شکل ۳۹: تصویر دوم امتحان شده (اسب و سگ)

مدل را به صورت زیر وارد می کنیم.

```
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'deeplabv3_resnet50', pretrained=True)
```

شکل ۴۰: وارد کردن مدل

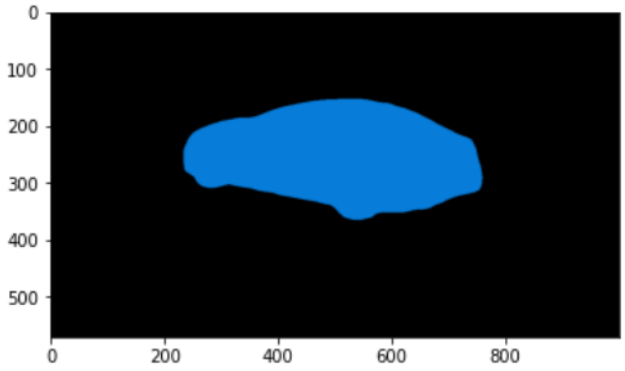
در مدل پارامتر pertained را برابر true ثرار دادم که نیازی به آموزش دادن شبکه از بتدا نباشد. در توضیح کتابخانه آمده است که عکس های ورودی باید نرمالایز شوند. برای این کار از فرایند زیر استفاده شد. لازم به ذکر ایت که عددهای میانگین و ورایانس در خود کتابخانه نوشته شده بودند.

```
preprocess = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

input_tensor = preprocess(input_image)
```

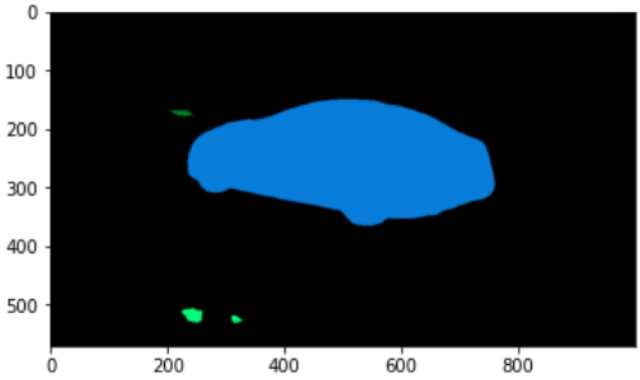
شکل ۴۱: کد نرمالایز کردن عکس ها

این نکته قابل ذکر است که عکس ها به صورت دستی و در محل فایل های کد بارگذاری شدند

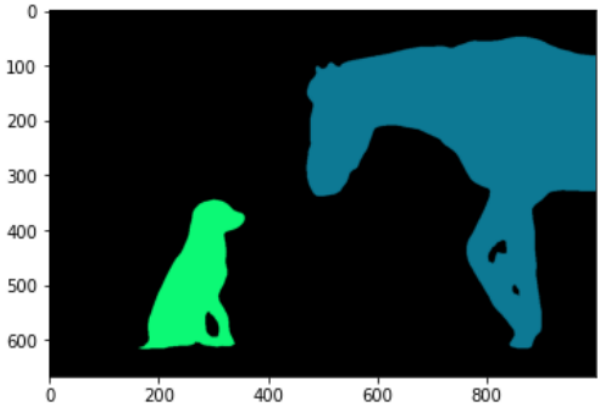
در نهایت نتایج به صورت زیر هستند



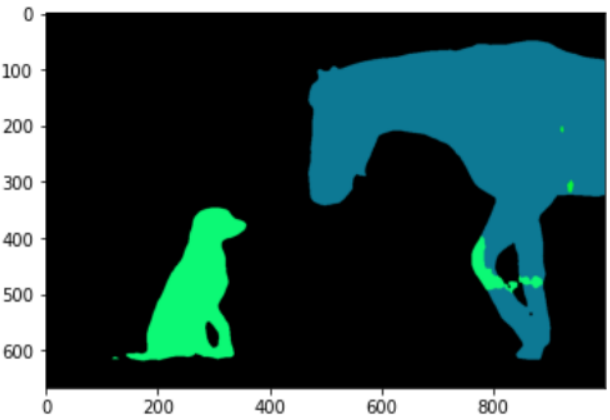

شکل ۴۲: نتایج پردازش روی عکس ماشین با مدل DeepLab




شکل ۴۳: نتایج پردازش روی عکس ماشین با مدل FCN




شکل ۴۴: نتایج پردازش روی عکس اسب و سگ با مدل DeepLab




شکل ۴۵: نتایج پردازش روی عکس اسب و سگ با مدل FCN

همانطور که در شکل های ۴۲ تا ۴۵ مشاهده می شود می بینیم که مدل DeepLab عملکرد بهتری از مدل FCN دارد

# سوال ۴ – تشخیص اشیا

الف)

در YOLOv1 هر grid صرفا میتواند یک شی را تشخیص دهد و در دیتا هایی که اشیا در آن overlap دارند دچار مشکل میشود.

هم چنین YOLOv1 نمیتواند bounding box هایی با ابعاد نامتعارف یا ابعادی که در آن training set شبکه نبوده است ایجاد کند.

در YOLOv2 برای حل این مشکلات چند راهکار درست شده است :

High resolution classification :

در ۱۰ ایپاک اول از تصاویر با رزولوشن بهتر استفاده شده است که Map را ۴ درصد بهبود میدهد

Batch normalization :

این کار باعث بهبود ۲ درصدی Map میشود

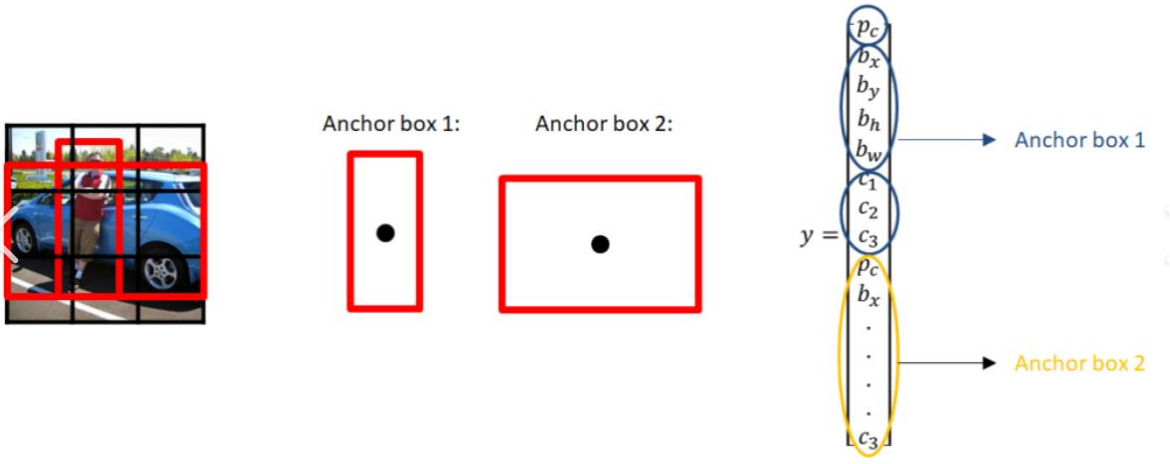
Direct location prediction:

در yolov1 / bounding box میتواند مقادیر بیشتر از ۱ و کمتر از صفر اختیار کند ولی در yolov2 اینگونه نیست.

Conv with anchor boxes :

برای اینکه در هر grid بتوان بیشتر از یک شی را تشخیص داد و overlapping و مشکلات آن را برطرف کرد است.

در حالت قبل هر شی در training به یک grid و یک نقطه وسط نسبت داده میشد اما الان علاوه بر این دو مورد یک anchor box نیز اضافه میشود.




برای این هدف اعمالی روی شبکه انجام شده است که شامل موارد زیر است :

۱.کاهش سایز تصویر ورودی به طوری که feature map ها هم سایز با grid باشند.

۲.برداشتن لایه های fully-connected

۳. انتقال کلاس بندی از grid cell به bounding box

ب)

در yolov4 از مفهوم bag of freebies( تکنیک هایی که باعث افزایش در عملکرد مدل بدون افزایش هزینه استنباط میشود) و مفهوم bag of specials ( تکنیک هایی که دقت را افزایش میدهد ولی هزینه های محاسبات را نیز افزایشس میدهد) استفاده شده است. و مدت زمان training بیشتر میشود .

### Bag of Freebies (BOF)

1. Data augmentation techniques: Cutmix (Cut and mix multiple images containing objects that we want to detect), Mixup(Random mixing of images), Cutout, Mosaic data augmentation.
2. Bounding box regression loss: Experimentation of different types of bounding box regression types. Example: MSE, IoU, CIoU, DIoU.
3. Regularization: Different types of regularization techniques like Dropout, DropPath, Spatial dropout, DropBlock.
4. Normalization: Introduced the cross mini-batch normalization which has proven to increase accuracy. Along with techniques like Iteration-batch normalization and GPU normalization.

### Bag of Specials BOS

1. Spatial attention modules (SAM): Generates feature maps by utilizing the inter-spacial feature relationship. Help in increasing accuracy but increase the training times.
2. Non-max suppression(NMS): In the case of objects that are grouped together we get multiple bounding boxes as predictions. Non-max suppression reduces false/excess boxes.
3. Non-linear activation functions: Different types of activation functions were tested with the YOLOv4 model. Example ReLU, SELU, Leaky, Swish, Mish.
4. Skip-Connections like weighted residual connections(WRC) or cross-stage partial connections(CSP).

در yolov5 میتوان به یادگیری خود به خود anchor box ها و mosaic data augmentation اشاره کرد.

به همان دقت yolov4 یا بیشتر از آن دست میباد اما تفاوتش در این است که توان کمتری برای محاسبات استفاده میکند. در نتیجه سرعت بهتری باید داشته باشد.

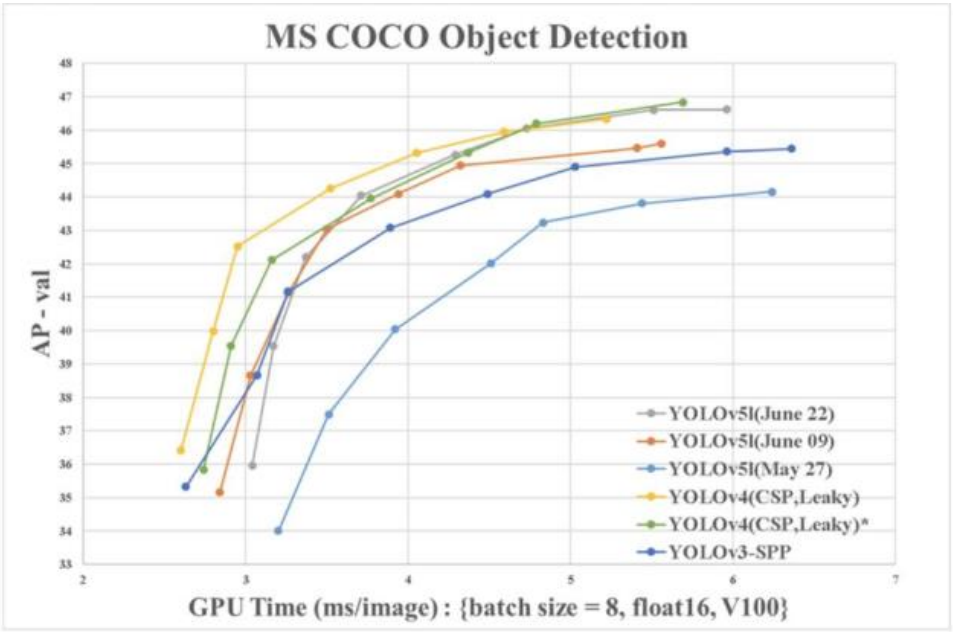
یکی دیگر از بهبود های آن استفاده از cross stage partial connection (csp) در back bone هست.

فریم ورک راحت تری برای تست و آموزش دارد .

هم چنین باید توجه داشت حافظه مورد نیاز برای ذخیره وزن ها برای ورژن YOLOV4 بیشتر از YOLOV5 است.

به طور کلی برای سرعت بیشتر و دقت قابل قبول و را آسان تر از YOLOV5 استفاده میشود. و برای تسک هایی با دقت ملزوم بالا از YOLOV4 در darknet استفاده میشود.

| | YOLOv3 | YOLOv4 | YOLOv5 |
|---|---|---|---|
| Neural Network Type | Fully convolution | Fully convolution | Fully convolution |
| Backbone Feature Extractor | Darknet-53 | CSPDarknet53 | CSPDarknet53 |
| Loss Function | Binary cross entropy | Binary cross entropy | Binary cross entropy and Logits loss function |
| Neck | FPN | SSP and PANet | PANet |
| Head | YOLO layer | YOLO layer | YOLO layer |




شکل ۴۶

پ)

شبکه های یک مرحله ای :

در این شبکه ها دو عمل localization , classification با هم اتفاق میافتد که باعث میشود سرعت این نوع شبکه ها از سرعت شبکه های دو مرحله ایی بیشتر باشد.

این نوع شبکه ها برای object هایی که شکل غیر عادی دارند یا وقتی تعداد زیادی شکل کوچک در تصویر داریم به خوبی شبکه های دو مرحله ایی عمل نمیکنند.

مثال :

Yolo , SSD,Retinanet

شبکه های دو مرحله ایی :

در این نوع شبکه ها دو عمل classification , localization از هم جدا هستند . شبکه ابتدا ناحیه ایی که شی در آن است را تشخیص داده و سپس آن را کلاس بندی میکند.

دقت این نوع شبکه از دقت شبکه های یک مرحله ایی بیشتر است و اما سرعت آن ها کمتر است .

معمولا به روش end to end نمیتوان آن ها را آموزش داد

مثال :

R-CNN،Faster R-CNN ،Mask R-CNN وG-RCNN

ت)

برای پیاده سازی این بخش از مراحل مختلفی تشکیل شده است که شامل موارد زیر است :

## 1.import libraries and install dependencies:

در این مرحله باید مخزن github YOLOV5 را clone کنیم.

پس از این کار میبینیم پوشه yolov5 به فایل های google colab اضافه میشود / سپس کتابخانه pytorch و تعدادی method را به محیط کار اضافه میکنیم.

## 2.load /download/unzip the dataset(roboflow)

فایل های .txt / data.yaml / valid/ train در صورت سوال به ما داده شده اند :

فایل های valid , train شامل تصویر ها و label های تصاویر هستند .

فایل data.yaml که در آن لیبل ها و تعداد آن ها موجود است.

فایل . txt دارای اطلاعات preprocess هایی است که روی این داده ها انجام شده است.

با train.py با فایل train مدل را آموزش خواهیم داد و با استفاده از detectمحتویات valid را به مدل میدهیم و نتایج را تست میکنیم.

سپس با استفاده از فایل data.yaml لیبل ها و تعداد کلاس ها را میابیم :

```
nc: 6
names: ['blue', 'green', 'red', 'vline', 'white', 'yellow']
```

میبینیم که ۶ کلاس داریم که شامل قرمز و آبی و سبز و خط و سفید و زرد است.

## 3. Train

برای این دستور پارامتر هایی در نظر گرفته شده است که در پایین توضیح داده شده اند:

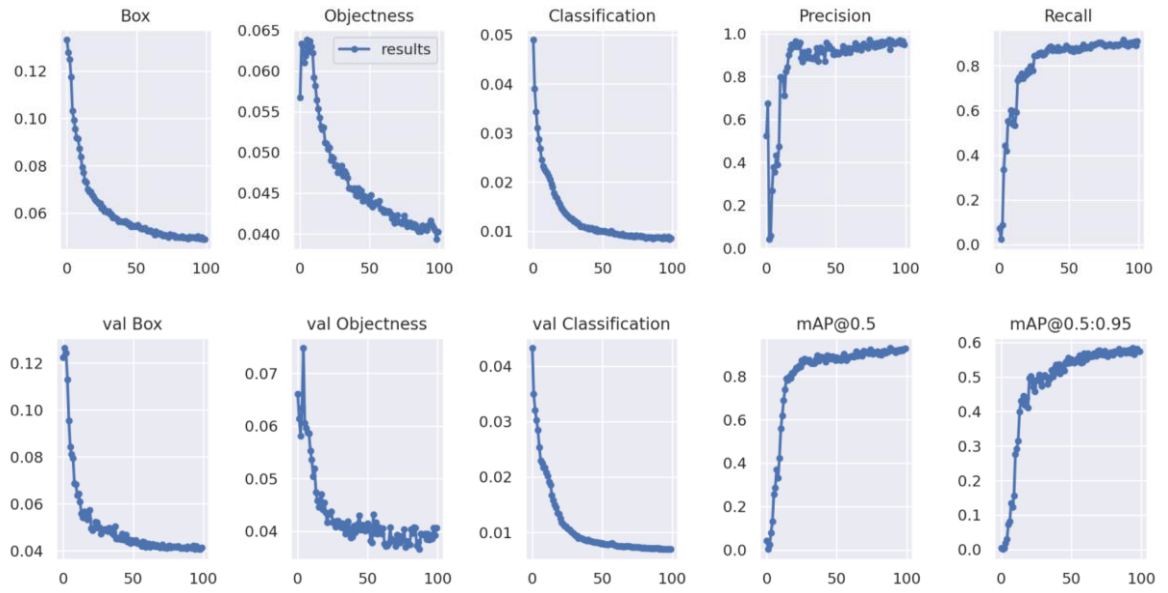
۱. batch (سایز batch در پروژه ۱۶ است)// ۲. Epoch ( که در پروژه ۱۵۰ است)

۳. img ( سایز تصویر است که ۴۲۰ است)// ۴. data ( مسیر data.yaml ) //۵ . cfg ( configuration مدل را برای ما مشخص میکند )

۶. weights ( برای مشخصات وزن ها هست) // ۷. name ( نام نتیجه مدل)

۸. no – save (نقطه نهایی ( check point) را ذخیره میکند.

۹. cache ( تصاویر برای یادگیری سریع تر )

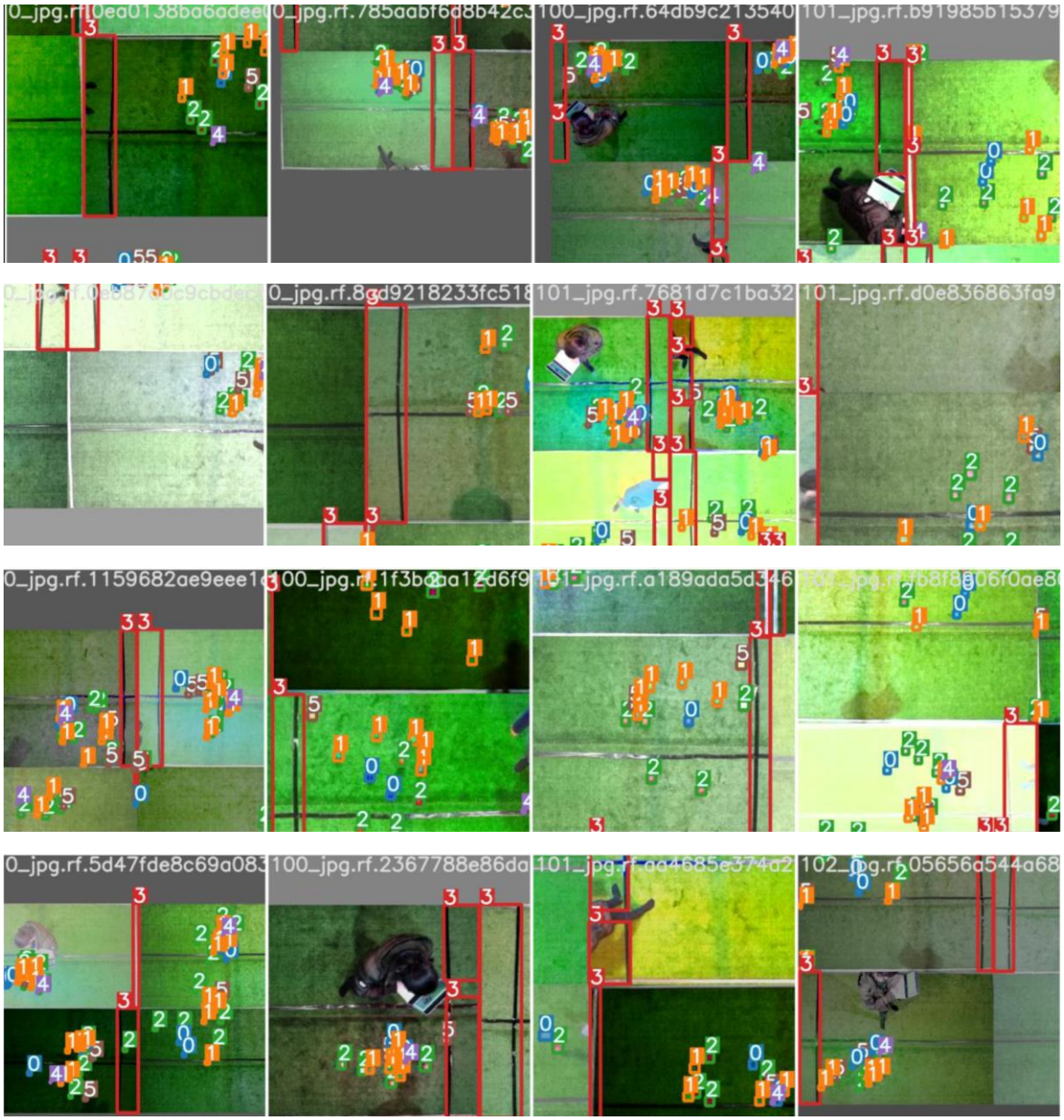



شکل ۴۷ - نتایج نموداری به سبک ابتدایی مدل با تمرین در 150ایپاک

همانطور که میبینیم مقدار mAP0.5 برابر  با ۰,۹۴۱ و mAP0.5:0.95 برابر با ۰,۵۹ است.

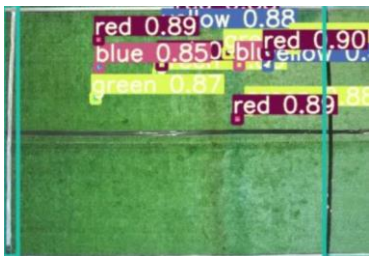
نتایج برای مدل های تمرین:

شکل ۴۸ - نتایج تشخیص مدل



شکل ۴۹- نتایج تشخیص مدل

بعد از تمرین داده های آموزش داده های پوشه validرا ارزیابی کردیم و تشخیص objectرا روی آن بررسی کردیم. نتیجه یک نمونه در شکل زیر آمده است.




شکل ۵۰ - نتیجه تشخیص مدل به ازای داده ارزیابی