



## تمرین شماره 2

درس یادگیری عمیق

استاد درس: دکتر فاطمی زاده

نام: محمد سینا حسن نیا

شماره دانشجویی: 96108515

### سوال 3

الف) در ورژن شماره 1 رابطه به صورت زیر می باشد که  $C$  مربوط به کلاس می باشد.

$$c = 300 \rightarrow S \times S \times (C + B^*5)$$

بنابراین اگر از اعداد مقاله استفاده کنیم خواهیم داشت :

$$c = 300 \rightarrow 7 \times 7 \times (300 + 2^*5) \rightarrow 7 \times 7 \times 310$$

برای ورژن شماره 3 نیز داریم :

$$c = 300 \rightarrow N \times N \times (3^*(4 + 1 + 300)) \rightarrow N \times N \times 915$$

ب) در الگوریتم Yolo-v1 برای هر گرید یا سل فقط دو باکس برای یافتن هر شی تخمین زده می شود؛ یعنی در هر گرید را حداقل دو باکس برای یافتن اشیا بررسی می کند؛ به همین دلیل، اگر تراکم اشیا در یک ناحیه تصویر بیشتر باشد، الگوریتم از یافتن همه اشیا باز می ماند و حتی ممکن است با توجه به تراکم بالای اشیا و ترکیب شدن ویژگی های آنها در یک بخش، نتواند هیچ یک از اشیا را به درستی تشخیص بدهد.

در الگوریتم Yolo-v2 به جای دو باکس برای هر گرید از ۵ باکس استفاده شده است و به این صورت می توان مشکل تراکم زیاد اشیا را تا حد زیادی حل کرد.

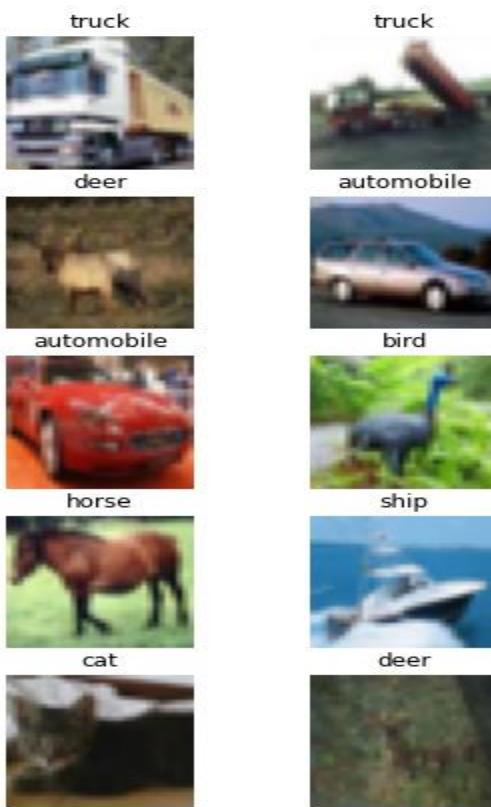
همچنین باکس های موجود در Yolo-v1 فقط از طریق داده های آموزشی، اصلاح و بهینه می شوند، اما در ورژن دوم از ایده اطلاعات پیشین یا anchor box ها استفاده شده است.

ج) در الگوریتم Yolo-v3 به جای دو باکس برای هر گرید از ۳ باکس استفاده شده است. همچنین، این باکس ها ابعاد مختلفی دارند؛ و به شکل های مختلف و به صورت باکس های  $1 \times 1$ ,  $2 \times 1$ ,  $1 \times 2$  هستند؛ در نتیجه اشیایی که نزدیک به هم هستند و همچنین اندازه های کاملاً متفاوتی دارند، را می توان بهتر تشخیص داد. در واقع باکس ها می توانند شکل های متنوع تری را پیش بینی کنند.

## سوال 1

### مقدمه

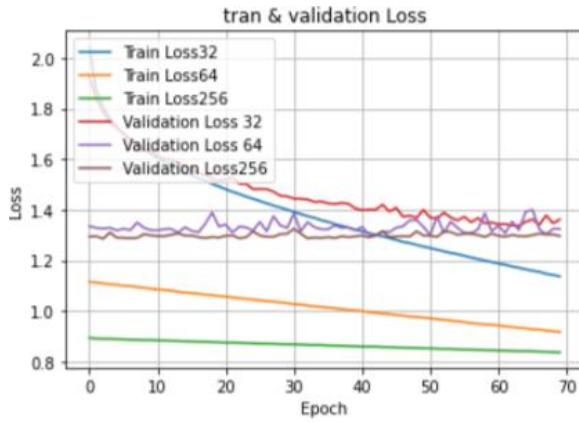
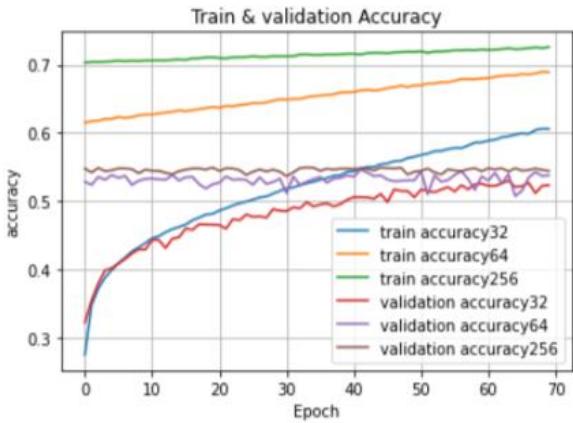
دیتاست **CIFAR-10** شامل 60 هزار تصویر رنگی است که در 10 کلاس دسته بندی شده است . به طور معمول از 50 هزار تصویر آن به عنوان مجموعه داده آموزشی استفاده می‌کنند و 10 هزار تصویر را به عنوان داده تست استفاده می‌کنند. 10 تصویر ابتدای آن همراه با نام شی به صورت زیر می‌باشد:



همچنان با استفاده از آزمون و خطا دریافتیم با 70 ایپاک و نرخ یادگیری ۱۰٪ نتایج بهتری بدست میاوریم و در مدل های خود این ۲ مقدار را ست کردیم.

**(الف)** **batch size** یکی از هایپرپارامترهای مهم شبکه های عمیق می‌باشد. **batch size** بزرگ امکان افزایش سرعت محاسباتی از طریق موازی بودن GPU ها را فراهم می کند با این حال **batch size** بزرگ قدرت تعمیم پذیری شبکه را پایین می آورد. برای بهینه سازی توابع

محدب یک کشمکش بین مزایای Batch size کوچکتر و بزرگتر وجود دارد. از یک طرف، استفاده از Batch ای برابر با کل مجموعه داده، همگرایی با بهینه مطلق تابع هدف را تضمین می کند. با این حال، این به قیمت همگرایی کندرت به آن نقطه بهینه است. از سوی دیگر استفاده از Batch ای با سایز کوچک تر نشان داده شده است که همگرایی سریع تری به پاسخ های مناسب دارد. این بخاطر آن است که اندازه دسته های کوچک تر به مدل اجازه می دهد پیش از دیدن همه داده ها شروع به یادگیری کند. نقطه ضعف استفاده از اندازه دسته کوچکتر این است که این مدل تضمینی برای همگرایی به بهینه مطلق ندارد. در بهینه سازی غیر محدب که امروزه در یادگیری عمیق با آن بیشتر موجه هستیم به طور تجربی مشاهده شده است که Batch size کوچک تر نه تنها باعث سریع تر شدن فرآیند آموزش می شود بلکه قابلیت generalization به شبکه را نیز می دهد. دلیل generalization بهتر به وجود نویز در آموزش Batch size کوچک نسبت داده می شود. از آنجایی که شبکه های عصبی به شدت مستعد overfit شدن هستند، ایده این است که استفاده از Batch size کوچک، که هر Batch نمایشی نویزدار از کل مجموعه داده است، باعث نوعی پویایی می شود. با این وجود برای دیتاست های مختلف عدد Batch size بهینه می تواند متفاوت باشد. بنابراین برای این دیتاست به ازای 3 سایز 32 و 64 و 256 BATCH شبکه را آموزش می دهیم. نتایج به شکل زیر می باشد. همانطور که دیده می شود شبکه با اندازه 256 در پارامتر دقیق هم در ولیدیشن و هم در تست نسبت به دو شبکه دیگر بهتر عمل کرده است. برای LOSS نیز هم در آموزش هم و هم در ولیدیشن با اندازه 256 loss Batch epoch داریم. از لحاظ سرعت هنگامی که اندازه batch برابر 32 است سرع شدن یک epoch نسبت به دو حالت دیگر کمتر است و زمان بیشتری طول می کشد تا یک epoch طی شود. دلیل آن این است که در هر epoch تعداد دفعات update شدن بیشتر می شود و در نتیجه با 32batch در هر epoch سرعت کمتری نسبت به 64 و 256 داریم. اما در سوی دیگر این Update شدن زیاد منجر به این می شود که شبکه با سرعت بیشتری به پایداری برسد و Converge کند



همچنین برای اندازه Batch های مختلف داریم:

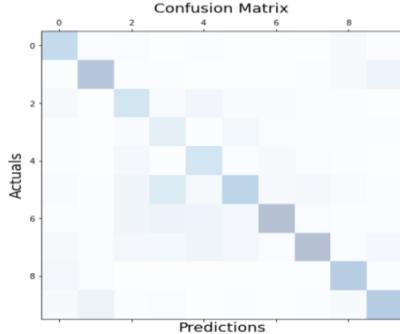
: Batch size 32

خطا و دقت بر روی داده های تست:

```
313/313 [=====] - 2s 6ms/step - loss: 1.3613 - accuracy: 0.5235
```

ماتریس آشیتگی برای داده های تست:

```
tf.Tensor(
[[470 15 35 11 26 5 3 12 76 15]
 [ 33 663 17 21 13 10 11 18 80 164]
 [ 88 8 54 10 52 10 50 51 32 19 10]
 [ 15 32 54 10 52 10 50 51 32 19 24]
 [ 31 7 102 32 364 44 63 40 29 7]
 [ 57 21 147 299 86 523 73 92 49 25]
 [ 40 23 150 182 178 100 700 35 24 29]
 [ 88 37 113 104 153 116 41 708 42 93]
 [111 34 9 13 11 11 11 5 598 30]
 [ 63 180 16 37 10 20 9 28 66 603]], shape=(10, 10), dtype=int32)
```



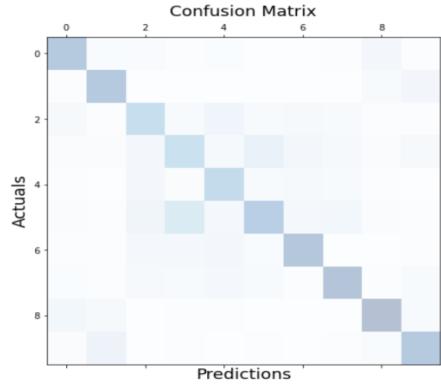
: Batch size 64

خطا و دقت بر روی داده های تست:

```
313/313 [=====] - 1s 4ms/step - loss: 1.3237 - accuracy: 0.5381
```

## ماتریس آشتفتگی برای داده‌های تست:

```
tf.Tensor(  
[[585 47 52 19 33 11 6 25 101 31]  
[ 24 581 10 9 4 4 7 11 56 119]  
[ 79 15 419 64 145 58 71 55 23 16]  
[ 39 40 107 384 69 197 106 76 38 66]  
[ 28 7 104 43 433 59 79 57 22 7]  
[ 35 26 138 286 91 531 89 114 36 31]  
[ 21 20 93 93 105 48 602 13 9 21]  
[ 32 27 67 62 94 64 17 617 16 63]  
[114 68 18 20 20 13 12 12 654 62]  
[ 43 169 9 20 6 15 11 20 45 584]], shape=(10, 10), dtype=int32)
```



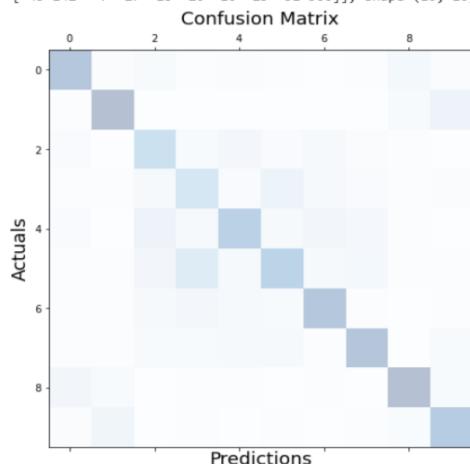
: Batch size 256

## خطا و دقت بر روی داده‌های تست:

```
313/313 [=====] - 1s 4ms/step - loss: 1.2959 - accuracy: 0.5446
```

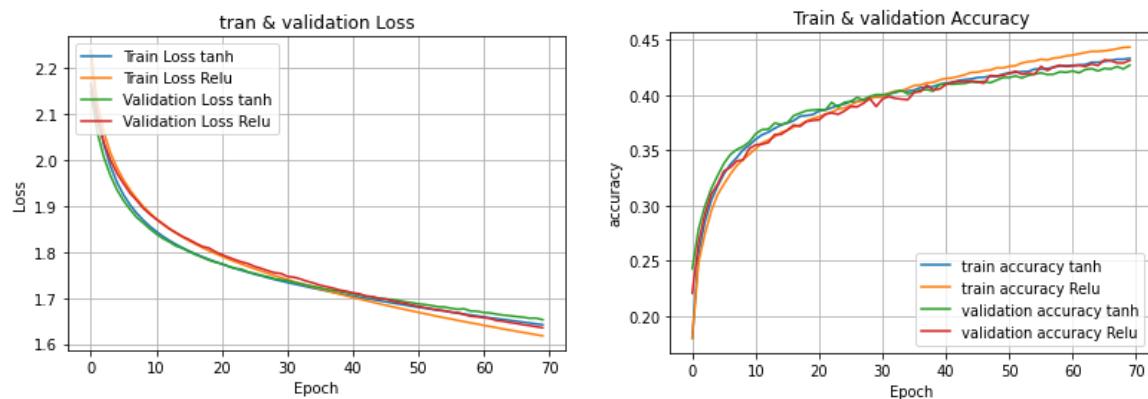
## ماتریس آشتفتگی برای داده‌های تست:

```
tf.Tensor(  
[[603 38 63 24 37 17 8 31 98 36]  
[ 30 656 15 14 6 5 11 12 65 172]  
[ 55 12 382 62 107 53 68 41 21 10]  
[ 25 21 82 327 52 179 76 55 24 42]  
[ 55 10 174 79 524 83 118 94 31 15]  
[ 23 18 118 271 67 500 67 98 30 22]  
[ 19 22 86 107 86 57 607 16 5 20]  
[ 25 23 60 66 84 71 18 619 13 59]  
[122 58 13 23 24 15 17 11 662 58]  
[ 43 142 7 27 13 20 10 23 51 566]], shape=(10, 10), dtype=int32)
```



ب) برای بررسی توابع RELU و Tanh می‌دانیم Tanh یک تابع فعالسازی است که مقداری بین -1 تا 1 دارد. مزیت استفاده از آن این است که در Tanh مقادیر منفی به صورت قوی به منفی map می‌شوند و ورودی‌های صفر به حوالی صفر نزدیک می‌شوند. ازویژگی‌های می‌توان گفت این تابع مشتق پذیر است. همچنین این تابع صعودی است در حالی که مشتق آن صعودی نیست. اضعف‌های این تابع می‌توان گفت Tanh همانند سیگموید می‌تواند دچار gradient vanishing شود. تابع Relu مقداری از صفر تا بینهایت دارد. این تابع و مشتق آن هر دو صعودی است. مشکل آن این است که تمام مقادیر منفی به صفر تبدیل می‌شود که این توانایی مدل را برای برازش و آموزش صحیح از داده‌ها کاهش می‌دهد. همچنین مزیت آن این است که دچار gradient vanishing نمی‌شود. نتایج به دست آمده برای شبکه‌ها با استفاده از 2

در در لایه مخفی و همچنین برای مدل با دو Relu در لایه مخفی به شرح زیر می‌باشد:



با توجه به مشکلات هر کدام از توابع که در بالا بیان شد به هر حال همانطور که ملاحظه می‌شود شبکه ترین شده با ReLU عملکرد بهتری نسبت به شبکه ترین شده با Tanh دارد.

همچنین برای دو شبکه ترین شده داریم:

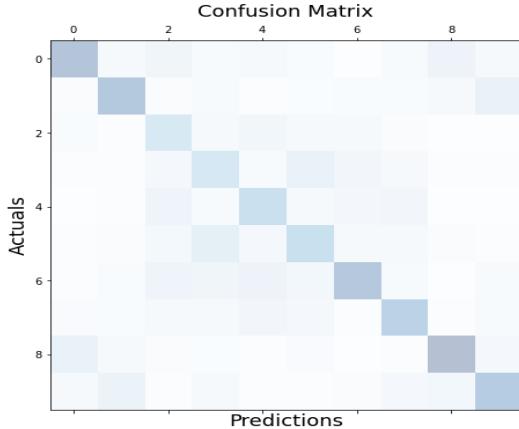
### : Using Tanh

خطا و دقت بر روی داده‌های تست:

```
313/313 [=====] - 1s 3ms/step - loss: 1.6528 - accuracy: 0.4271
```

## ماتریس آشتفتگی برای داده‌های تست:

```
tff.Tensor(
[[ 537   63  125   56   74   47   14   59  150   71]
 [ 42  510   40   55   26   34   43   47   69  172]
 [ 43  17  264   65  118   77   77  42   12  16]
 [ 26  32   80  273   58  167  107   67   29  30]
 [ 10  20  137   53  337   72   94  186   10  14]
 [ 20  35  88  203   79  342   71   69   35  25]
 [ 29  44  138  127   152  101  513   59   8  52]
 [ 49  44  64  64  106   85   28  440   19  53]
 [173  74   39   33   27  50   14   29  568   86]
 [ 71 161   25   71   23   25   39  82  100  487]], shape=(10, 10), dtype=int32)
```



## : Using Relu

خطا و دقیق روی داده‌های تست:

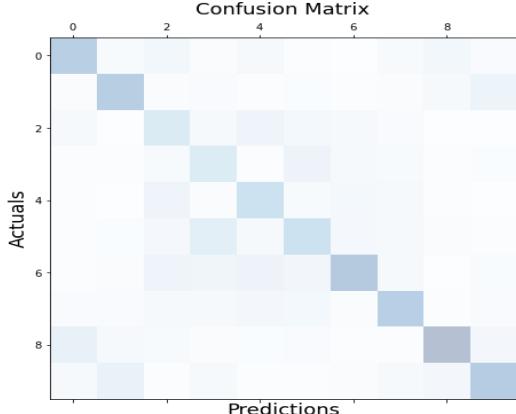
313/313 [=====] - 1s 3ms/step - loss: 1.6352 - accuracy: 0.4315

## ماتریس آشتفتگی، برای داده‌های تست:

```

tf.Tensor(
[[489 55 104 39 65 30 11 58 104 52]
 [ 35 499 36 49 26 34 28 42 74 165]
 [ 68 14 259 79 144 92 75 50 13 7]
 [ 19 32 60 255 32 158 67 54 22 34]
 [ 21 13 151 40 339 61 92 71 17 10]
 [ 22 34 87 218 73 343 84 69 38 26]
 [ 27 41 147 131 163 112 535 64 14 44]
 [ 52 49 73 73 100 93 43 487 22 53]
 [197 80 57 41 34 50 24 30 598 98]
 [ 70 183 26 75 28 27 41 75 98 511]], shape=(10, 10), dtype=int32)

```

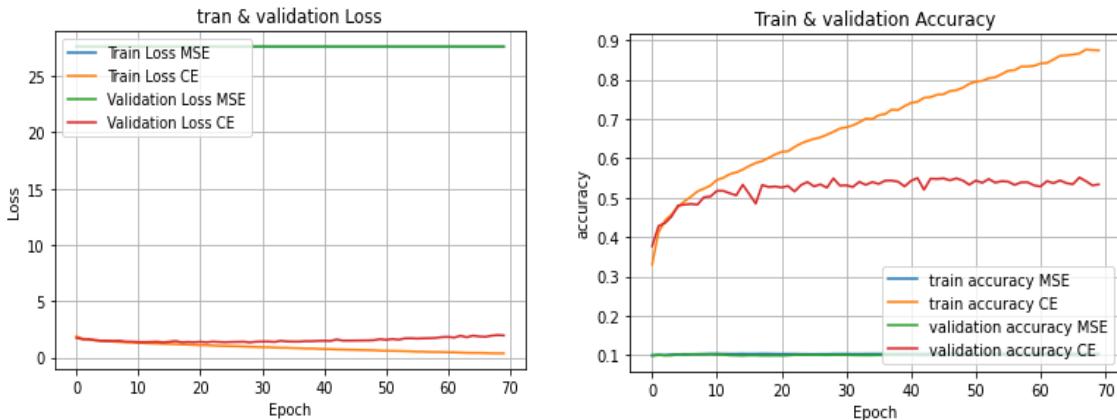


ج) تابع هزینه‌های cross entropy و mse را در نظر گرفتیم میدانیم که به صورت کلی در مسائل طبقه بندی و regression در MSE بهتر جواب می‌دهد. در اینجا نیز به همین صورت بود و نتایج CE به مراتب بهتر و دقیق‌تر بودند. دلیل ریاضی آن این است که CE احتمال هست و به هر کلاس بازه‌ای از ۰ تا ۱ را اختصاص می‌دهد در حالی که MSE هیچ مرزی برای مقادیر خود ندارد. دو دلیل عمدی که MSE یک انتخاب بد برای classification است این است که اولاً استفاده از MSE به آن معناست که ما فرض کردیم که دیتای ما از یک توزیع نرمال تولید شده است در حالی که دیتا در واقعیت این گونه نیست. دوماً تابع MSE یک تابع غیر محدب برای classification می‌باشد. همچنین به عنوان یک مثال اگر بخواهیم برای نامناسب بودن MSE برای classification بگوییم فرض کنیم برای یک مسئله ۶ کلاسه داشته باشیم:

- True probabilities = [1, 0, 0, 0, 0, 0]
- Case 1:** Predicted probabilities = [0.2, 0.16, 0.16, 0.16, 0.16, 0.16]
- Case 2:** Predicted probabilities = [0.4, 0.5, 0.1, 0, 0, 0]

MSE برای case1 برابر 0.128 و برای case2 0.1033 می‌باشد. اگرچه case1 به صورت صحیح پیش‌بینی می‌کند loss برای case1 از case2 بیشتر است!

نتایج استفاده از CE و MSE به شرح زیر می‌باشد:



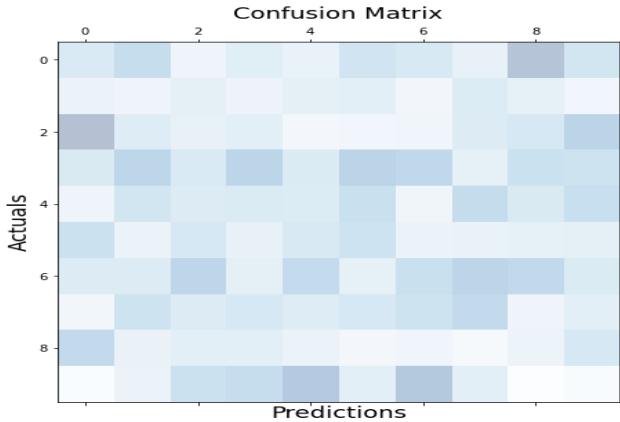
همانطور که ملاحظه می‌شود بهوضوح استفاده از CE برای طبقه بندی بهتر است.  
: Using MSE

خطا و دقت بر روی داده‌های تست:

```
313/313 [=====] - 1s 4ms/step - loss: 27.6103 - accuracy: 0.0999
```

## ماتریس آشتفتگی برای داده‌های تست:

```
tf.Tensor(  
[[102 137 67 93 74 116 105 80 191 113]  
[ 73 67 86 69 86 87 56 98 82 60]  
[199 95 79 88 51 60 63 99 108 156]  
[103 151 102 153 101 156 146 83 126 121]  
[ 68 112 97 101 98 128 62 138 103 132]  
[125 72 106 78 104 120 73 76 83 85]  
[ 97 97 152 86 140 84 129 153 144 100]  
[ 58 122 97 108 96 107 122 143 67 90]  
[142 75 89 88 72 52 63 43 71 106]  
[ 33 72 125 136 178 98 181 87 25 37]], shape=(10, 10), dtype=int32)
```



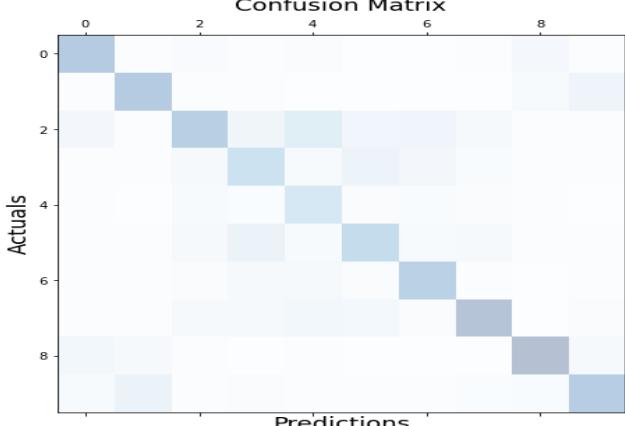
: Using CE

## خطا و دقت بر روی داده‌های تست:

```
313/313 [=====] - 1s 4ms/step - loss: 1.9506 - accuracy: 0.5333
```

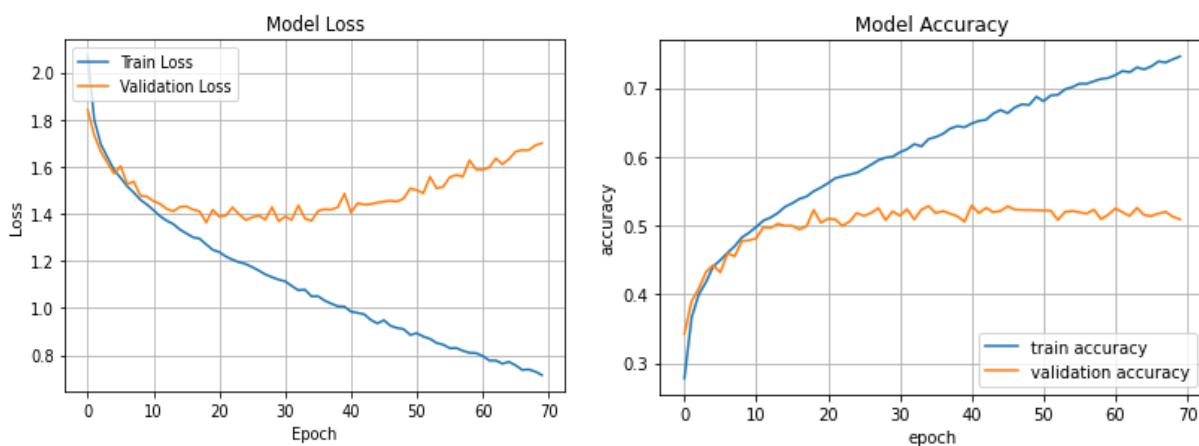
## ماتریس آشتفتگی برای داده‌های تست:

```
tf.Tensor(  
[[591 31 53 18 37 14 9 27 90 30]  
[ 25 594 15 15 8 4 14 7 63 169]  
[109 31 554 146 264 139 156 83 30 24]  
[ 20 24 87 383 67 183 108 47 27 31]  
[ 21 6 66 35 320 38 47 41 19 11]  
[ 24 15 69 194 67 445 65 78 20 20]  
[ 15 16 45 73 79 45 538 16 7 19]  
[ 21 24 79 84 118 104 36 656 14 45]  
[116 72 16 14 25 9 11 11 680 79]  
[ 58 187 16 38 15 19 16 34 50 572]], shape=(10, 10), dtype=int32)
```



همچنین از Adam optimizer نیز استفاده شده است که نتایج آن در زیر آمده است. مشکل Adam این است که بعضی اوقات در همگرایی مشکل دارد و ممکن است هیچ‌گاه همگرا نشود. هر چند Adam از SGD بسیار سریع‌تر است اما stability الگوریتم SGD از Adam بهتر است. همانطور که در شکل زیر مشاهده می‌کنید Adam با 70 ایپاک دچار overfit می‌شود در حالی که SGD با شرایط مشابه با 70 ایپاک دچار overfit نشد. همچنین از دیگر مشکلات Adam می‌توان به این اشاره کرد که :

- جواب‌هایی را پیدا می‌کند که نسبت به SGD بدتر تعمیم می‌یابند.
- معمولاً حتی زمانی که loss train ADAM به های مشابه یا کمتر می‌رسد عملکرد تست بدتر است.
- اگر چه ADAM معمولاً پیشرفت اولیه سریع‌تری دارد اما عملکرد آن‌ها به سرعت در داده validation ثابت می‌شود.



خطا و دقت بر روی داده‌های تست:

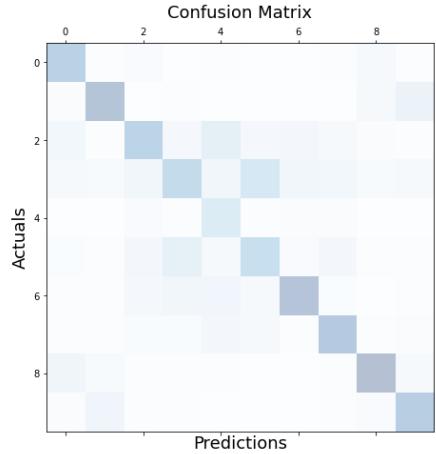
```
313/313 [=====] - 1s 4ms/step - loss: 1.6991 - accuracy: 0.5089
```

ماتریس آشفتگی برای داده‌های تست:

```

tf.Tensor(
[[508 23 50 12 28 10 4 25 71 31]
 [ 43 616 13 21 4 6 11 8 68 179]
 [110 25 486 88 227 86 104 70 35 22]
 [ 65 55 121 428 120 307 130 111 62 70]
 [ 12 6 51 21 276 24 41 42 11 7]
 [ 33 15 107 223 74 399 52 100 24 22]
 [ 24 24 98 128 135 82 617 33 10 27]
 [ 24 28 47 46 105 65 23 571 17 36]
 [142 64 16 19 22 8 7 17 649 67]
 [ 39 144 19 22 9 13 11 23 53 539]], shape=(10, 10), dtype=int32)

```



همراه با SGD در واقع به این دلیل استفاده می شود که باعث سرعت بخشیدن و همگرایی سریع تر از طریق درجهات صحیح شدن بردارهای گرادیان می شود. همچنین در مقایسه با Momentum + SGD همگرایی بهتری دارد. درواقع مسائلی وجود دارد که Momentum + SGD در آن همگرا می شود ولی Adam دچار مشکل می شود.

۵) با استفاده از پارامترهای زیر بهترین نتیجه حاصل شده است :

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 1024)	3146752
dense_1 (Dense)	(None, 128)	131200
dense_2 (Dense)	(None, 10)	1290
<hr/>		
Total params: 3,279,242		
Trainable params: 3,279,242		
Non-trainable params: 0		

Epoch = 20

Learning rate = 0.01

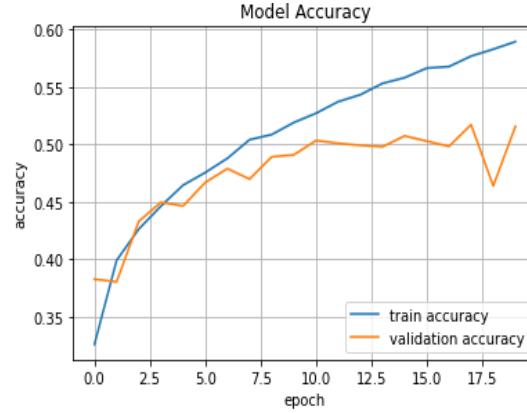
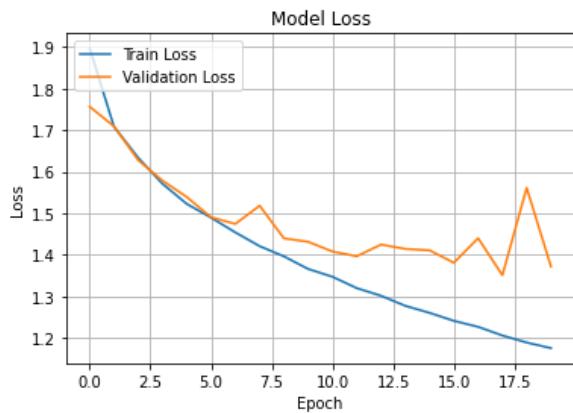
Batch size = 256

Activision Function: Relu in 2 hidden layer

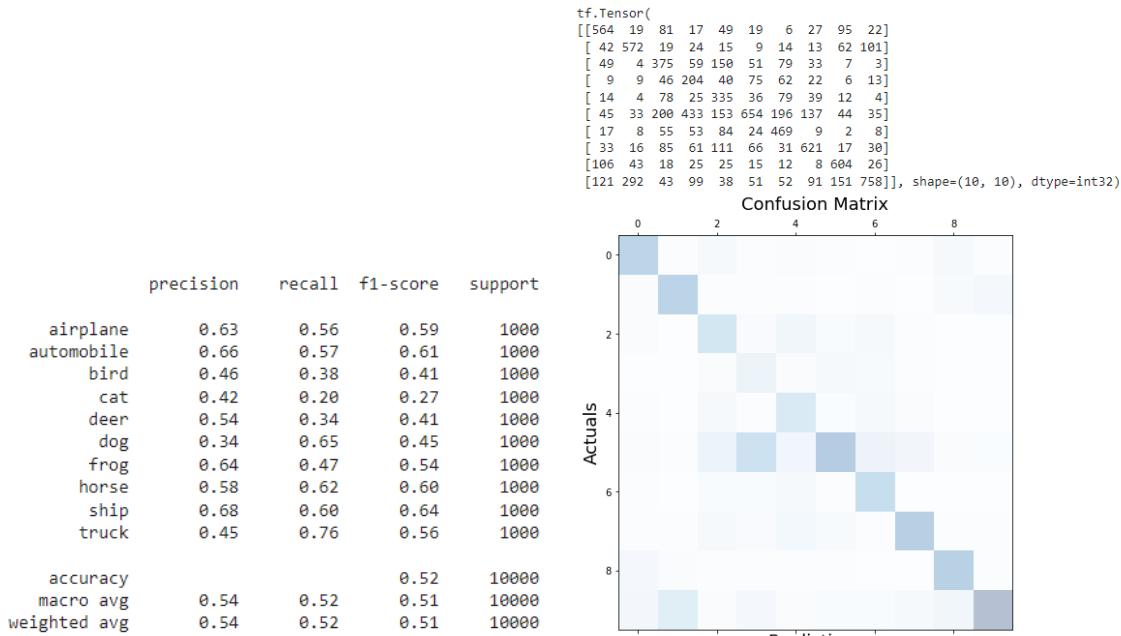
Loss function: CE

Optimizing Algorythm: SGD

برای این مدل داریم :



313/313 [=====] - 1s 4ms/step - loss: 1.3717 - accuracy: 0.5156

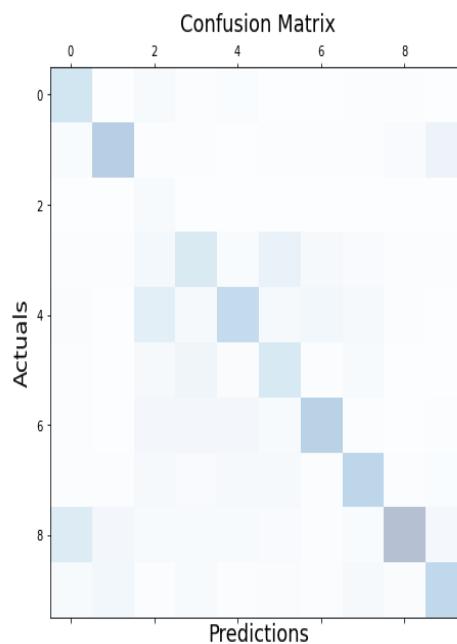


وو ز) به این نوع از دیتا ها دیتاهای imbalanced گفته می شود در واقع در این دیتاهای توزیع کلاس ها imbalance است. بسیاری از الگوریتم ها classification دقیقی برای کلاس غیر

معمولی دارند. در واقع برای این که بگوییم چگونه باعث اختلال می شود باید گفت که اکثر الگوریتم های یادگیری ماشینی فرض می کنند که داده ها به طور مساوی توزیع شده اند. بنابراین هنگامی که ما یک عدم تعادل کلاسی داریم، طبقه بندی کننده یادگیری ماشینی تمایل بیشتری به سمت طبقه اکثریت دارد و باعث طبقه بندی بد کلاس اقلیت می شود. روش هی مختلفی برای این مشکل وجود دارد مانند : , resampling, oversampling, cluster the abundant class

... در اینجا ما از resample کردن استفاده می کنیم. ابتدا مقداری از داده های کلاس Airplane و Bird را حذف می کنیم و شبکه را آموزش می دهیم نتیجه به شکل زیر می شود :

```
tf.Tensor(
[[405  7  71  21  36  11  6  16  26  9]
 [ 54 671  33  30   9  14  18  20  59 212]
 [  7  0  71  4  6  7  4  3  0  1]
 [ 22 15 117 362 57 231 87 59 20 27]
 [ 46 12 296 90 540 100 139 90 13 12]
 [ 17  7  97 168 45 377 33 66 10 12]
 [ 22 12 123 131 127 76 632 24  2 15]
 [ 29 18  83 60 88 84 22 584 13 37]
 [334 122 76 68 66 58 29 51 812 108]
 [ 64 136 33 66 26 42 30 87 45 567]], shape=(10, 10), dtype=int32)
```

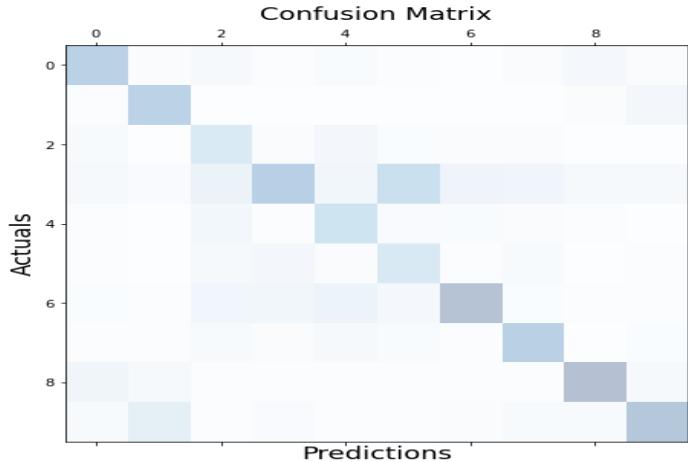


همانطور که می بینیم عملکرد روی هر دو کلاس و به خصوص کلاس bird پایین می آید. حال با استفاده از resampling مشکل imbalanced بودن را از بین می برمی داریم:

```

tf.Tensor(
[[541  37  73  20  45  20   3  40  96  43]
 [ 21 527   9  10   5   6   6  10  36 105]
 [ 58   6 301  37 105  33  39  39  11   6]
 [ 79  55 189 554 134 402 159 156  67  74]
 [ 29   2 114  29 364  50  46  44  19   6]
 [ 17   9  71 112  34 308  21  58  10  17]
 [ 33  27 140 132 185  89 661  33  13  20]
 [ 22  22  65  34  77  48  17 548   8  32]
 [143  81  21  19  26  20  14  13 682  66]
 [ 57 234  17  53  25  24  34  59  58 631]], shape=(10, 10), dtype=int32)

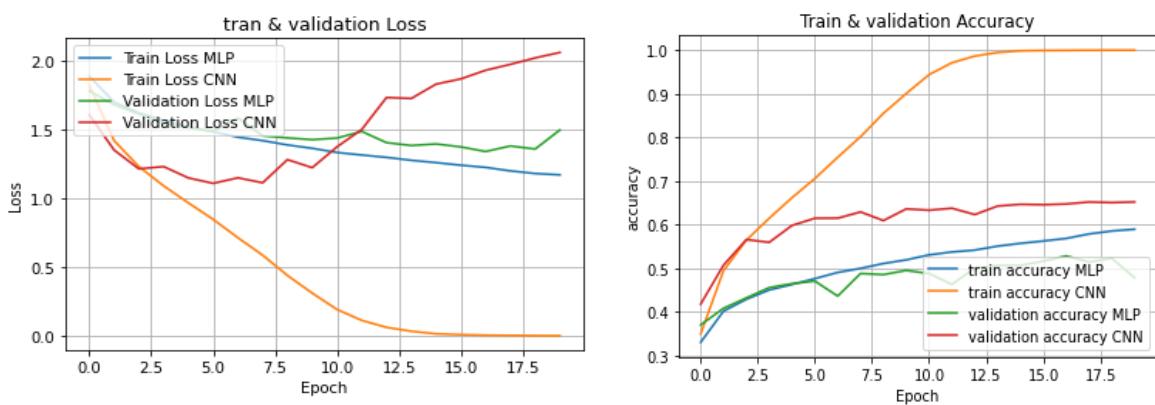
```



همانطور که می بینیم عملکرد روی هر دو کلاس و به شدت کلاس Bird تغییر می کند و خوب می شود زیرا توزیع تقریبا یکسان شده است.

## استفاده از CNN

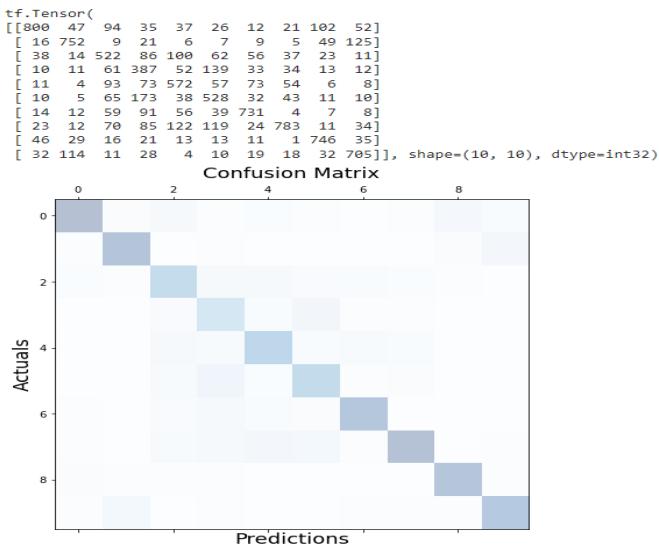
**الف)** به بهترین شبکه به دست آمده از قبل لایه های کانولوشنی اضافه می کنیم و شبکه را پیاده سازی می نماییم. برای مقایسه شبکه MLP و شبکه CNN داریم:



همانطور که مشاهده می شود شبکه CNN به مراتب عملکرد بهتری نسبت به شبکه MLP هم در خطای و هم در دقت دارد (حتی دقت داده CNN validation که یعنی آن را ندیده است از دقت

شبکه MLP بالاتر است. که این در واقع به دلیل پیچیده شدن شبکه و استفاده از پارامترهای بیشتر می‌باشد. و نشان می‌دهد که در واقع لایه‌های کانولوشنی تاثیر مناسبی دارد. حال برای آن که خطای نهایی شبکه روی داده تست در حالت قبل از overfit را به دست آوریم داریم:

```
313/313 [=====] - 2s 7ms/step - loss: 1.4715 - accuracy: 0.6526
```

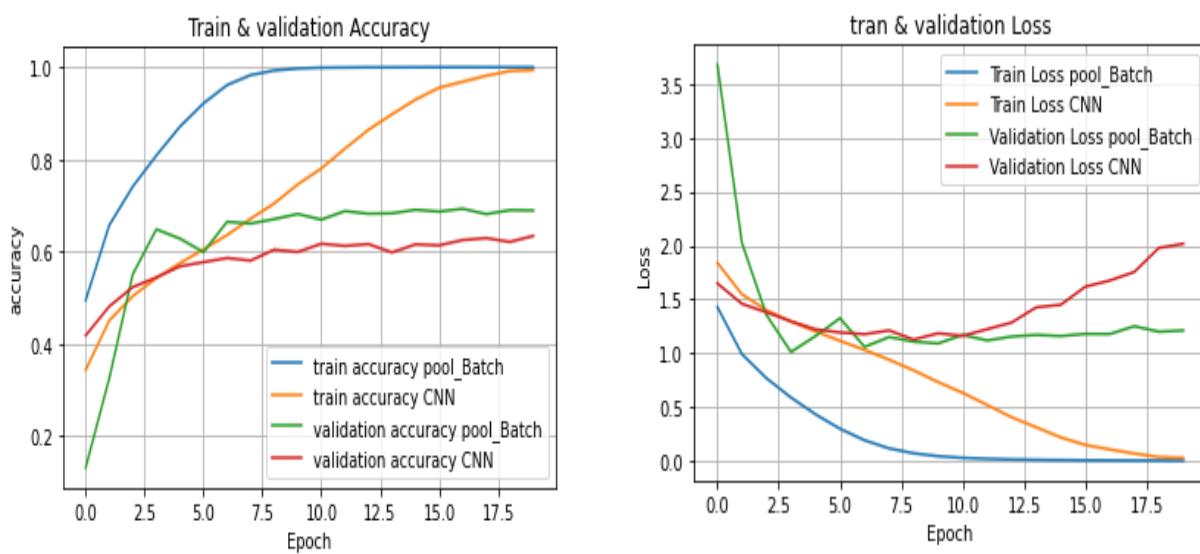


**ب)** در ابتدا لایه‌های Pooling و batch normalization را توضیح می‌دهیم. بنابراین داریم:

**Batch normalization:** نرمال سازی یک ابزار پیش پردازش داده است که برای آوردن داده‌های عددی به یک اسکیل معمول بدون تغییر شکل آن‌ها استفاده می‌شود. به طور کلی، زمانی که داده‌ها را به یک ماشین یا الگوریتم یادگیری عمیق وارد می‌کنیم، تمایل داریم مقادیر را به یک مقیاس متعادل تغییر دهیم. دلیل نرمال سازی ما تا حدی این است که اطمینان حاصل کنیم که مدل ما می‌تواند به طور مناسب تعمیم یابد. حال که به نرمال سازی دسته‌ای (Batch) فرآیندی است برای سریع‌تر و پایدارتر کردن شبکه‌های عصبی از طریق افزودن لایه‌های اضافی در یک شبکه عصبی عمیق. لایه جدید عملیات استانداردسازی و نرمال‌سازی را روی ورودی لایه‌ای که از لایه قبلی می‌آید انجام می‌دهد. اما دلیل اصطلاح Batch در نرمال سازی دسته‌ای Batch چیست؟ یک شبکه عصبی معمولی با استفاده از مجموعه‌ای از داده‌های ورودی به نام Batch آموزش داده می‌شود. به طور مشابه، فرآیند نرمال‌سازی در نرمال سازی Batch به صورت دسته

ای انجام می شود، نه با یک ورودی. از فواید استفاده از لایه BN این است که سرعت آموزش شبکه افزایش داده می شود و همچنین مشکل covariate shift داخلی را حل می کند. از این طریق، اطمینان حاصل می کنیم که ورودی برای هر لایه حول میانگین و انحراف معیار استاندارد، یکسان توزیع می شود. یک لایه BN تابع LOSS را smooth می کند که به نوبه خود با بهینه سازی پارامترهای مدل سرعت آموزش مدل را بهبود می بخشد.

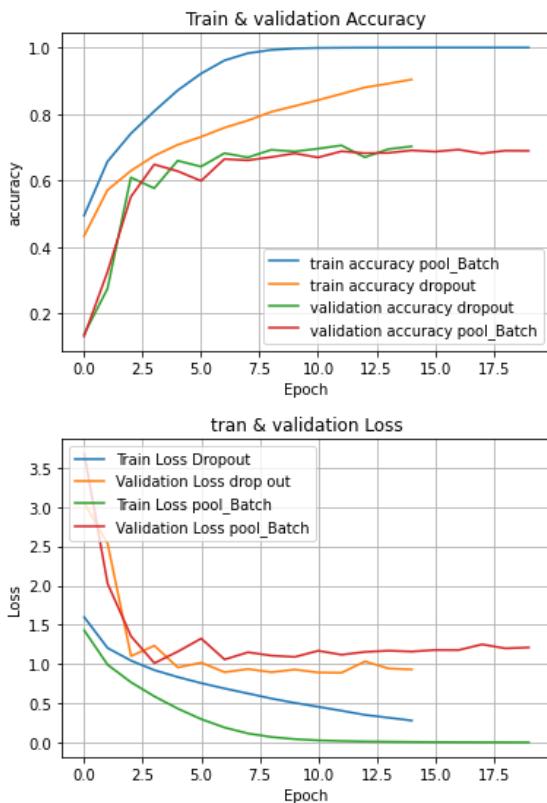
عملیات Pooling شامل لغزش یک فیلتر 2 بعدی بر روی هر کanal از Feature Pooling و خلاصه کردن ویگزهای موجود در منطقه تحت پوشش فیلتر است. لایه های map معمولاً به دو دلیل استفاده می شوند که اولی این است که ابعاد feature map ها را با آن کاهش می دهند و در نتیجه پارامترهای شبکه و محاسبات را کاهش می دهد. دومین دلیل آن است که لایه های pooling ویژگی های موجود در یک منطقه از feature map ایجاد شده توسط یک لایه کانولوشن را خلاصه می کند در نتیجه عملیات بیشتر بر روی ویژگی های خلاصه شده به جای ویژگی های دقیقاً موقعیت یافته (تولید شده توسط لایه کانولوشن) انجام می شود. این باعث می شود که مدل نسبت به تغییرات در موقعیت ویژگی ها در تصویر ورودی مقاوم تر باشد.



همانطور که دیده می شود شبکه سریع تر شده و سریع تر به همگرایی می رسد. همچنین دقت validation آن افزایش و خطای آن مقداری کاهش یافته است که تایید کننده صحبت های بالا

است. برای مقایسه کلی نیز شبکه سریع تر شده و به طور مثال train loss ان خیلی سریع پایین آمده است . همچنین دقت validation آن افزایش یافته است.

**ج)** شبکه‌های عمیق به دلیل پیچیدگی زیادی که دارند به شدت مستعد overfit کردن می‌باشند. از این رو روش‌های زیادی برای جلوگیری از overfit به وجود آمده است. در این روش‌ها که به شاخه regularization معروفند با انجام یک سری تغییرات سعی می‌شود از overfit کردن شبکه جلوگیری شود تا شبکه قابلیت تعمیم پذیری بیشتری داشته باشد. drop out یکی از روش‌های regularization می‌باشد که در واقع همان‌طور که در بالا نیز بیان شد از آن استفاده می‌شود تا از overfit جلوگیری و قدرت تعمیم پذیری شبکه افزایش یابد. فرم رگولاریزیشن کردن آن نیز مشابه L2\_regularization می‌باشد. نتایج به دست آمده پس از اعمال این الگوریتم عبارتست از:



همان‌طور که در بالا می‌بینیم استفاده از drop out باعث شده خطای ولیدیشن شبکه با drop out کاهش یابد و به بیانی دیگر تعمیم پذیرتر باشد.

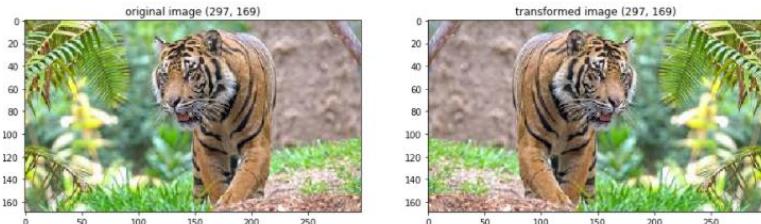
## سوال 2

الف) **Data Augmentation** تکنیک‌هایی هستند که برای افزایش تعداد داده‌ها با افزودن کپی‌هایی از داده‌های موجود که کمی تغییر یافته اند استفاده می‌شوند. این روش به عنوان یک **regulizer** عمل می‌کند و در نتیجه کمک می‌کند تا هنگام آموزش، یک مدل کمتر دچار **overfitting** شویم. همچنین علت اهمیت آن می‌تواند این باشد که اگر دیتاست غنی و کافی باشد مدل ما عملکرد بهتری خواهد داشت و دقیق‌تر خواهد بود. علاوه بر آن، برای مدل‌های یادگیری ماشین، جمع‌آوری و برچسب‌گذاری داده‌ها می‌تواند فرآیندهای طاقت‌فرسا و پرهزینه باشد. **Data Augmentation** به شرکت‌ها این امکان را می‌دهد که این هزینه‌های عملیاتی را کاهش دهند. همچنین از این تکنیک می‌توان برای حل مسئله **imbalance** بودن نیز استفاده کرد. بنابراین در صورت هر کدام از موارد بالا با استفاده از تبدیل‌های مختلفی که در زیر معرفی می‌کنیم تصاویر تغییر یافته شده تولید می‌کنیم و بر حسب نیاز‌هایی که در بالا بیان شد از آن‌ها استفاده می‌کنیم.

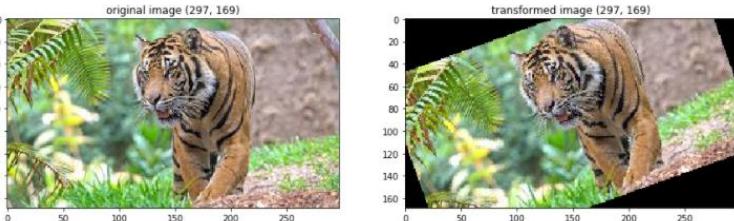
تکنیک‌هایی که برای **Data augmentation** استفاده می‌شوند به طور کلی می‌تواند شامل **Rotation, Scaling, Translation, Shearing, Mirroring,...** می‌باشد. یعنی داریم :

- Standard Transform

- **Flip** = the image is flipped horizontally or vertically



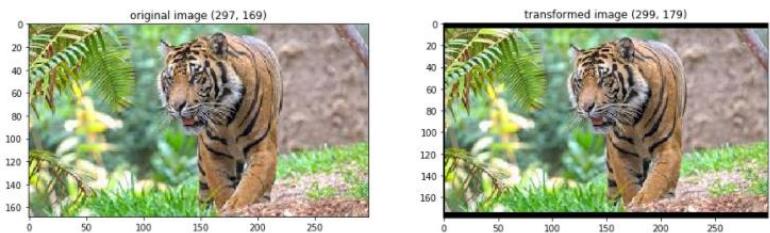
- **Rotation** = The image is rotated randomly in rotation.



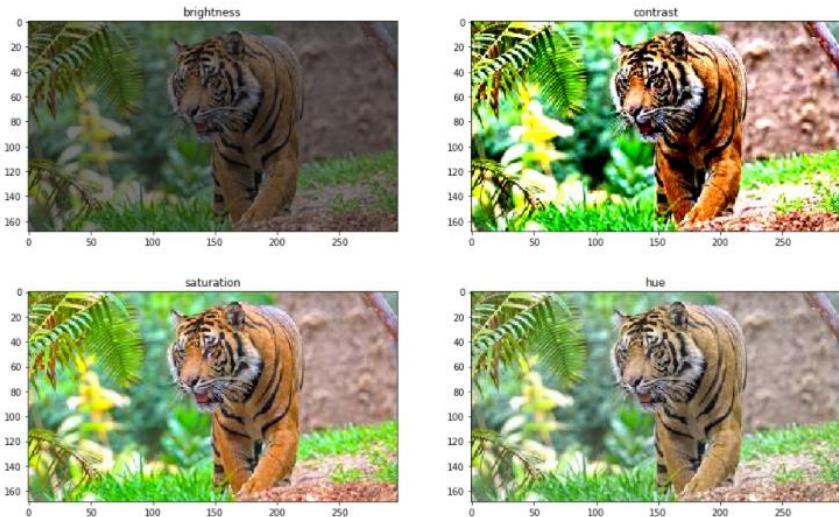
- **Scaling** = the image is resized to the given size e.g. the width of the image can be doubled.



- Padding = the image is padded with a given value on all sides.



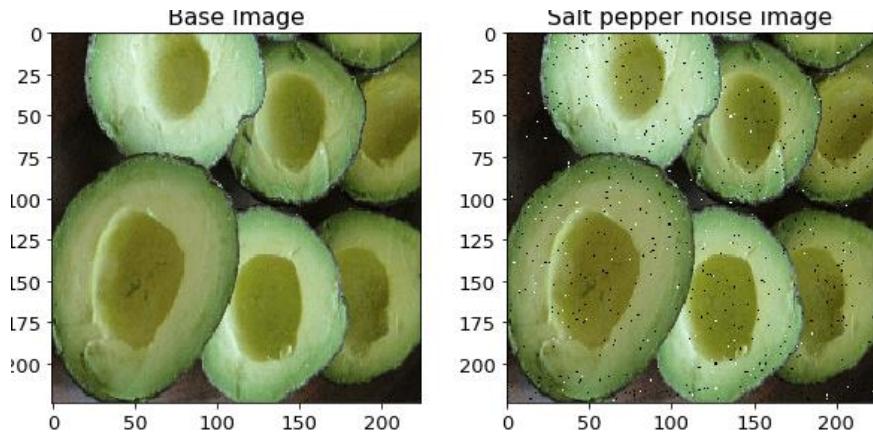
- Contrast/Brightness/illumination/color/... change



- Crop = a portion of the image is selected



- Additive// Multiplicative Noise = For blurry images, adding noise on the image can be useful.



- Translation = The image is shifted into various areas along the x-axis or y-axis



- For Audio : Audio data augmentation methods include cropping out a portion of data, noise injection, shifting time, speed tuning changing pitch, mixing background noise and masking frequency.
- Advanced Approach
  - Adversarial training/Adversarial machine learning: It generates adversarial examples which disrupt a machine learning model and injects them into dataset to train.
  - Reinforcement learning models train software agents to reach attain their goals and make decisions in a virtual environment.

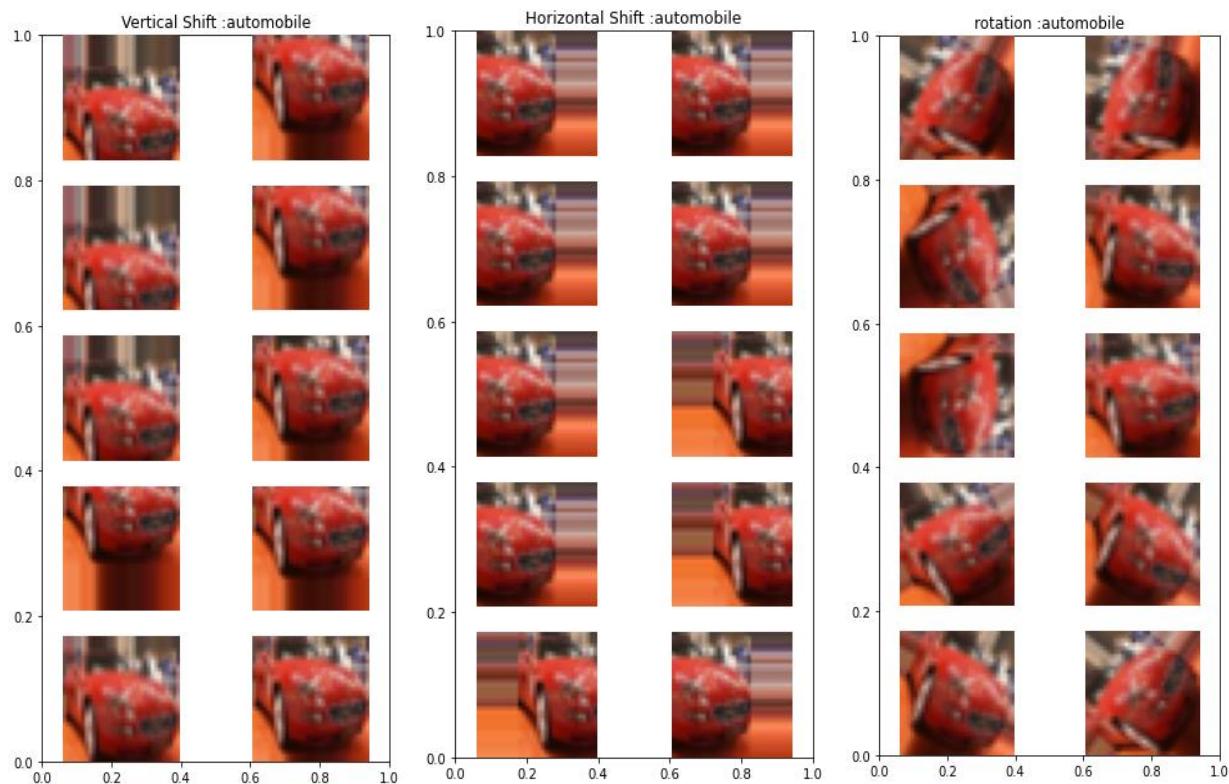
**Data Augmentation** معمولاً بر مجموعه آموزشی انجام می شود زیرا به تعمیم و استحکام مدل کمک می کند. این تکنیک را همچنین می توان هنگام پیش‌بینی با یک مدل فیت شده نیز به کار برد تا به مدل اجازه دهد برای چندین نسخه مختلف از هر تصویر در مجموعه داده تست پیش‌بینی کند. پیش‌بینی‌های روی تصاویر **Augment** شده می‌تواند به طور مثال میانگین گرفته شود، که می‌تواند منجر به عملکرد پیش‌بینی بهتری شود. در واقع این تکنیک TTA نام دارد. به طور خلاصه این تکنیک شامل ایجاد چندین نسخه **augment** شده از هر تصویر در تست دیتا، وداداشتن مدل برای پیش‌بینی هر یک، سپس بازگرداندن مجموعه‌ای از پیش‌بینی‌هاست. ( در واقع هدف طبقه‌بندی هر چه صحیح‌تر یک داده است)

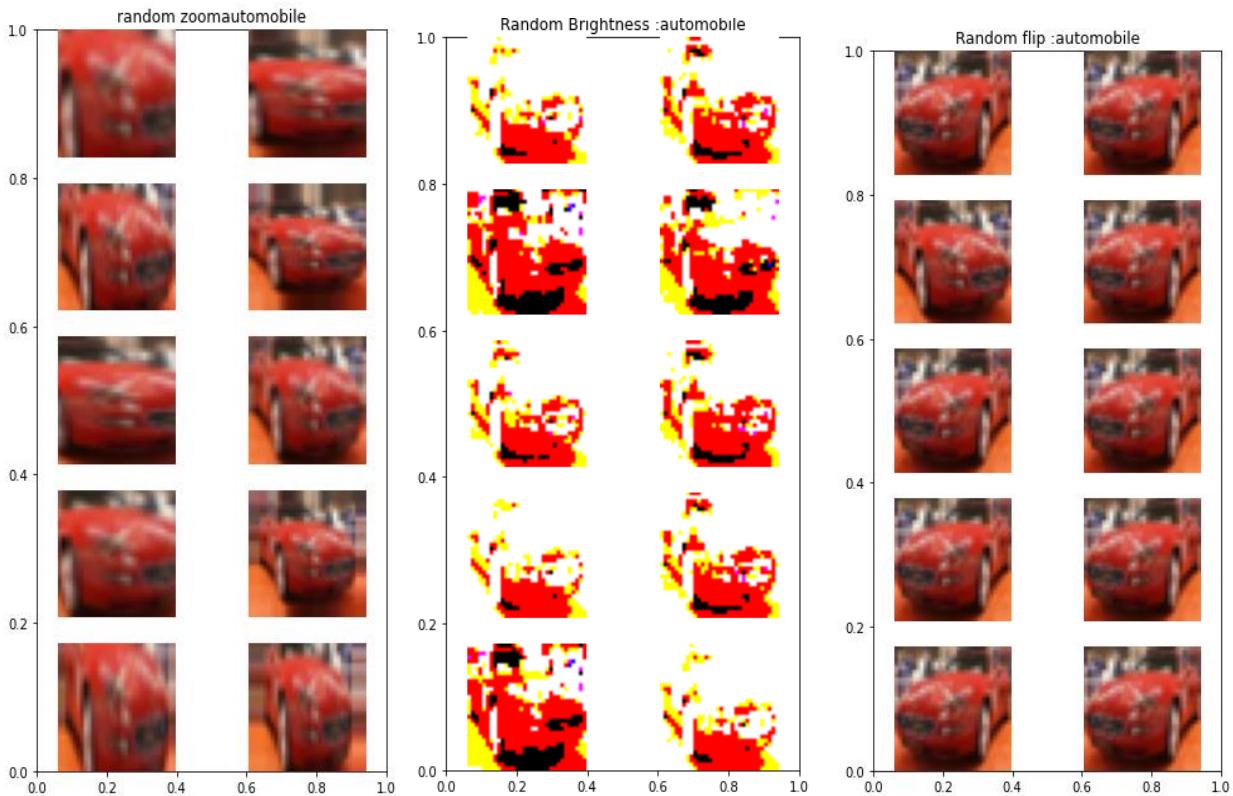
**ب)** با استفاده از تبدیل های مختلف تصویر هار را به دست آورديم. برای آن که اثرات مشعوذ باشد از هر تبدیل 10 تصویر تولید کردیم . برای تصویر original داریم:



برای تبدیل های مختلف داریم:

Transforms : shift,rotation,zoom,flip,brightness





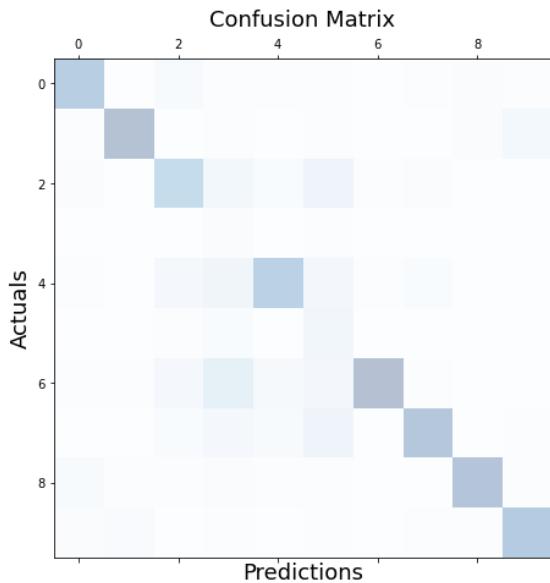
**ج)** با این کار شبکه imbalance می شود و در کلاس هایی که داده کم است دچار مشکل می شود. در واقع همانطور که در قسمت الف نیز در این مورد توضیح دادیم با این کار توزیع کلاس ها به هم می خورد . در واقع برای این که بگوییم چگونه باعث اختلال می شود باید گفت که اکثر الگوریتم های یادگیری ماشینی فرض می کنند که داده ها به طور مساوی توزیع شده اند. بنابراین هنگامی که ما یک عدم تعادل کلاسی داریم، طبقه بندی کننده یادگیری ماشینی تمایل بیشتری به سمت طبقه اکثربیت دارد و باعث طبقه بندی بد کلاس اقلیت می شود. داریم :

پس از آموزش شبکه با این مجموعه داده داریم :

```

tf.Tensor(
[[744  5  70  36  21  18   5  15  53  22]
 [ 27 879  6  38   7  28   8   8  46 127]
 [ 55  4 581 151  62 220  31  49   9   5]
 [  0  0  0  42   1  26   0   0   0   1]
 [ 22  1 115 189 705 133  38  58   6   4]
 [  0  1  14  59   3 167   1   4   0   0]
 [ 17  16 124 296  98 142 895  21   9   8]
 [ 11  3  59 116  81 220   5  827   4  24]
 [ 79  24  19  53  18  29  12   4 845  33]
 [ 45  67  12  20   4  17   5  14  28 776]], shape=(10, 10), dtype=int32)

```

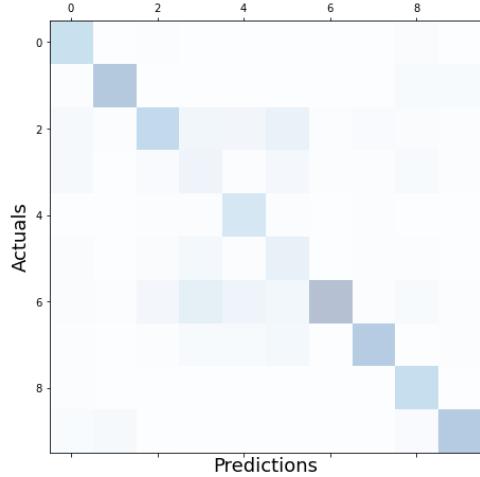


همانطور که دیده می شود در کلاس گربه و سگ یعنی کلاس های 3 و 5 عملکرد شبکه بسیار بد است. این از همان هدم توزیع یکسان داده ها ناشی می شود . دلیل آن را نیز در بالا گفتیم. یکی از راه ها برای جبران این کار استفاده از **data augmentation** می باشد. در قسمت بعدی همین کار را انجام می دهیم.

**(۵)** همان گونه که در زیر دیدع می شود کلاس های 3 و 5 که در حالت قبل در وضعیت خوبی قرار نداشتند اکنون وضعیتشان بهتر شده است . این به این دلیل است که توانستیم با استفاده از **balance data augmentation** دیتاست را کنیم.

```
tf.Tensor(  
[[556   3   30   4   8   2   2   2   53  14]  
[ 37 831   4  12   4   3   1   2  74  82]  
[ 89 17 625 173 164 261 34 72 48 17]  
[106   3  66 229  37 127 17 15 78 15]  
[ 9   2  35 39 453 35 10 33   3  5]  
[ 46   3  46 132 21 281  4  38 20  2]  
[ 51 33 151 319 231 153 926 38 77 25]  
[ 19   6  33 74 74 131  6 797   3 33]  
[ 26   5  2   4   4   3   0   0 576  4]  
[ 61 97   8 14   4   4   0   3  68 803]], shape=(10, 10), dtype=int32)
```

Confusion Matrix



### سوال 3

**الف)** در ابتدا به معرفی معماری شبکه و ایده ای که پشت آن قرار دارد می پردازیم:

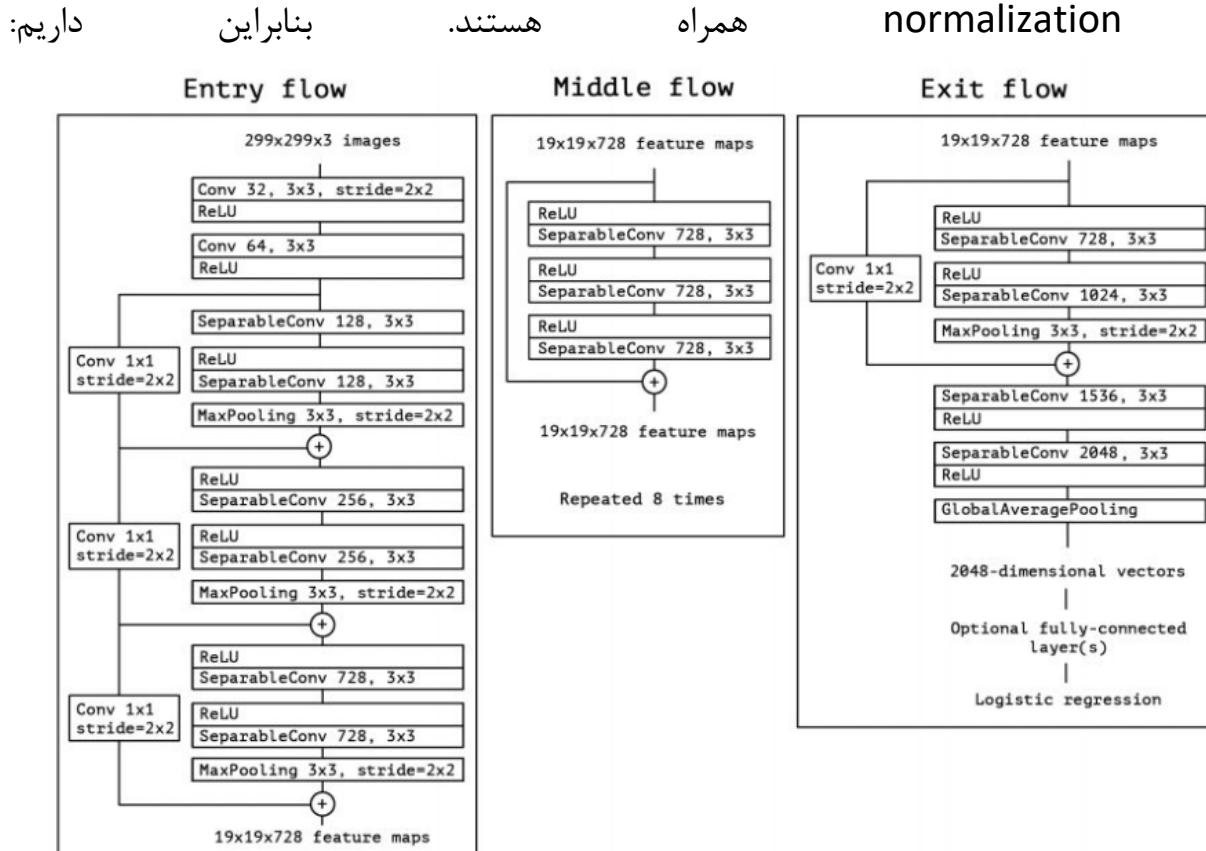
یک معماری برای شبکه عصبی کانولوشنی عمیق است که شامل کانولوشن‌های جدапذیر عمقی<sup>۱</sup> می‌باشد. Xception در واقع با الهام از Inception و توسط محققان گوگل ساخته شده است. در inception کانولوشن‌های  $1 \times 1$  برای فشرده‌سازی ورودی اصلی استفاده شد و از هر یک از این فضاهای ورودی، ما از انواع فیلترهای مختلف در هر یک از فضای عمق استفاده کردیم. Xception به نوعی معکوس است یعنی ابتدا فیلترهارا روی هر یک نقشه‌های عمق اعمال می‌کند و در نهایت فضای ورودی را با استفاده از کانولوشن  $1 \times 1$  با اعمال آن در عمق فشرده سازی می‌کند. البته Inception، Xception یک تفاوت دیگر نیز دارند و آن وجود یا عدم وجود غیرخطی بودن پس از عملیات اول است. در مدل Inception، هر دو عملیات با یک لایه غیر خطی ReLU همراه می‌شوند، با این حال Xception هیچ گونه غیرخطی را معرفی نمی‌کند.

با استفاده از Xception framework گوگل پیاده‌سازی شده است.  
(GPU:NVIDIA K80)

معماری Xception مطابق شکل زیر می‌باشد. داده‌ها ابتدا به entry flow می‌روند، سپس به middle flow که 8 بار تکرار شده است می‌روند و در نهایت به exit flow می‌روند. همچنین Batch باشد که تمام لایه‌های کانولوشن و کانولوشن جدآپذیر عمیقی با

---

<sup>۱</sup> Depthwise Separable Convolution



این معماری در اکثر چالش‌های طبقه بندی کلاسیک عملکرد بهتری نسبت به VGG-16 و InceptionV3 و Resnet داشته است.

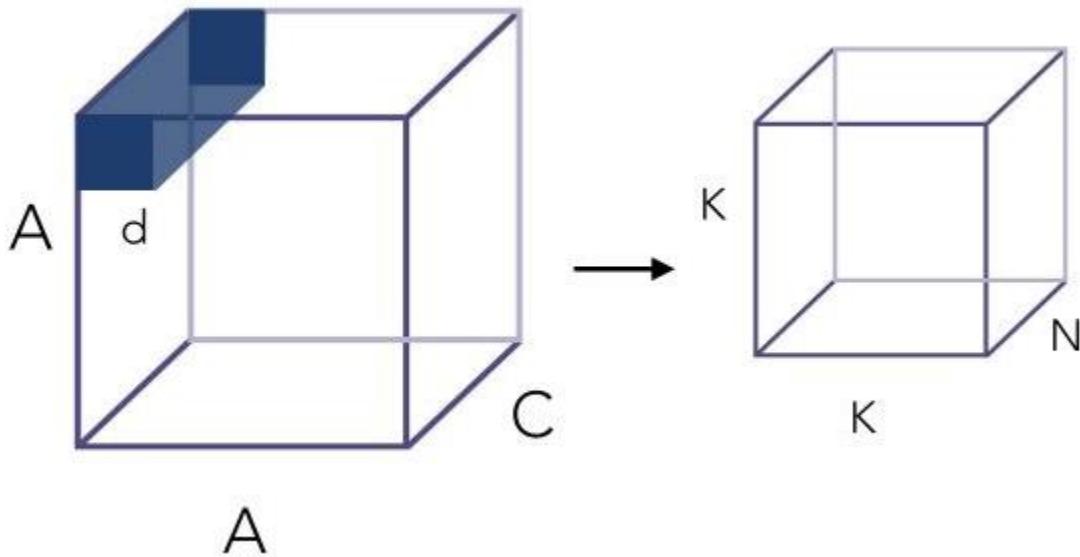
اگر بخواهیم بگوییم که Xception چگونه عمل می کند باید گفت این معماری یک معماری بهینه است که بر دو اصل پابرجاست:

- Depthwise Separable Convolution
- Shortcuts between Convolution blocks as in ResNet

ابتدا در مورد Depthwise Separable Convolution صحبت می کنیم:

Depthwise Separable Convolution جایگزینی برای کانولوشن‌های کلاسیک هستند که قرار است از نظر زمان محاسبه بسیار کارآمدتر باشند. کانولوشن کلاسیک به صورت جدی یک عملگر

هزینه‌بر می‌باشد. اگر فرض کنیم که تصویر ما بعد  $A^*A^*C$  داشته باشد که  $C$  تعداد کانال‌های آن می‌باشد و اگر روی آن یک کرنل  $d^*d$  اعمال کنیم آن گاه برای تعداد عملیات خواهیم داشت:



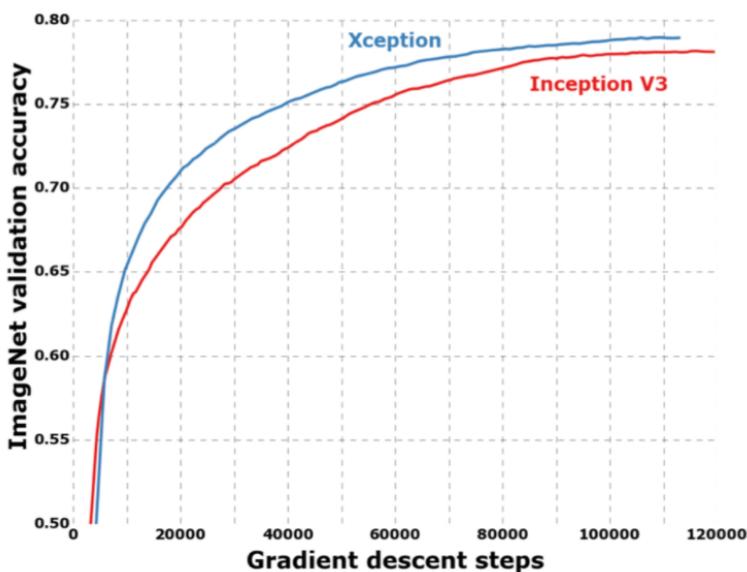
$$\# \text{Operation for } N \text{ kernel} = K^2 \times d^2 \times C \times N$$

برای فائق اومدن برا cost سنگین این نوع کانولوشن‌ها معرفی شده است. در این نوع کانولوشن در گام نخست به جای اعمال کانولوشن با سایز  $\times d$  ما کانولوشن‌هایی با سایز  $d \times 1$  اعمال می‌کنیم. به عبارت دیگر ما روی تمامی کانال‌ها کانولوشن را اعمال نمی‌کنیم. بنابراین یک  $K \times K \times C$  volume واقع، تا کنون، ما عملیات کانولوشن را فقط برای 1 فیلتر انجام دادیم، نه برای  $N$  تا از آن‌ها. حال باید گام دوم را انجام دهیم که عنوان آن pointwise convolution می‌باشد. برای نسخه‌های دیگر این شبکه می‌توان از شبکه L2MXception نام برد. در واقع در اینجا یک بهبود داده شده معرفی شد که ترم رگولاریزیشن L2-norm and mean را ترکیب می‌کند و تفاوتی در شبکه ارائه شده نیست. در واقع مورد گفته شده در بالا باعث بهبود عملکرد در دقت validation می‌شود.

برای مقایسه آن با سایر مدل‌ها داریم جدول زیر نشان می‌دهد که Xception از هر مدلی بر دیتاست ImageNet عملکرد بهتری دارد.

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	<b>0.790</b>	<b>0.945</b>

همچنین Xception نسبت به Inception دقیق‌تری را داده ولیدیشن در دیتاست imagenet دارد.



با این حال علی‌رغم این که دو ساختار تقریباً سایزی مشابه هم دارند Xception از Inception کندر است . برای آن که روش‌تر شود در جدول زیر تعداد پارامترها و همچنین گام‌های آموزشی بر ثانیه ( gradient update ) در زیر آمده است:

Table 3. Size and training speed comparison.

	Parameter count	Steps/second
Inception V3	23,626,728	31
Xception	22,855,952	28

در واقع همچنین جدول بالا نشان می دهد که عملکرد بهتر Xception نسبت به از افزایش حجم پارامترها و شبکه نیست بلکه به دلیل استفاده بهینه از پارامترهای شبکه در شبکه Xception است.

همانطور که مشخص است سایز ورودی  $299 \times 299 \times 3$  می باشد . برای preprocess باید سایز تصویر را مطابق ورودی سسیتم نماییم. همچنین باید پیکسل ها بین -1 و 1 اسکیل شوند که در سایت kerass نیز این نکته ذکر شده است.

`xception_preprocess_input() will scale input pixels between -1 and 1.`

خروجی مدل را می توان  $2048$  dimensional vector global average pooling به دست آمده است. البته باید این نکته را ذکر کرد که همانطور که در مدل نیز گفته شده است می توان یک سری لایه FC نیز به شبکه اضافه کرد مثلا اگر بخواهیم از این شبکه برای imagenet استفاده کنیم می توان یک لایه FC  $1000$  سلولی را به آن اضافه کنیم. معنای خروجی آن این می تواند باشد که می توان با اتصال لایه های مختلف به انتهای آن از آن برای task های مختلفی استفاده کرد. در ادامه نیز کاربردهای این را می بینیم.

**ب و ج)** با استفاده از transfer learning شبکه را پیاده سازی کردیم. سپس یک آناناس که از کلاس های قابل طبقه بندی برای این شبکه است را به شبکه دادیم و نتیجه مطابق شکل زیر شد. (3 دسته به ترتیب با بیشترین احتمال پیش بینی شده در خروجی نشان داده شده است که همانطور که میبینیم شبکه به طور خوبی آناناس بودن شبکه را تشخیص داده است:

```
[[('n07753275', 'pineapple', 0.94270843), ('n07747607', 'orange', 0.0010595159), ('n07753592', 'banana', 0.0008359188)]
pineapple is : True
orange is : False
banana is : False
```

همچنین برای آن که از این جلوگیری کنیم که همیشه بزرگ ترین عدد تشخیص داده شده را به عنوان لیبل بگیریم یک ترشولد نیز می گذاریم تا تعیین کند که این عدد بزرگ تا حد خوبی قابل استناد است. موردی که می خواهیم از آن جلوگیری کنیم در واقع می تواند این باشد که مثلا سه عدد بزرگ  $0.00001$  و  $0.0000001$  باشد. اگر ما ترشولد قرار ندهیم

عدد خروجی `argmax` را به عنوان لیبل در نظر میگیرد در صورتی که با این اعداد نمی توان گفت لیبل چیست.

۵) راه حل ما قرار دادن ترشولد است تا از این گونه موارد جلوگیری کنیم. دلیل آن نیز در بالا مفصل تر بیان شده است. برای مثال می دانیم کلاس کیک در `Imagenet` نیست. یک تصویر کیک به شبکه دادیم خروجی به صورت زیر بوده است :

```
[[(n07836838', 'chocolate_sauce', 0.23674652), ('n07614500', 'ice_cream', 0.03872641), ('n07871810', 'meat_loaf', 0.03390128)]]
chocolate_sauce is : False
ice_cream is : False
meat_loaf is : False
```

اگر ترشولد نداشتیم خروجی بستنی لیبل زده می شد در صورتی که کیک بوده است. با قرار دادن ترشولد همانطور که مشاهده می شود سه کلاس احتمال بالای داده شده `False` زده شده است یعنی از میزان حد نسباب ترشول م اعبور نکرده و قابل استناد نیست که بر اساس آن کیک را بستنی لیبل بزنیم!

۶) دیتابست `food101` را دانلود و با استفاده از دستوراتی که به طور کلی از `os` است آن را مهیاً استفاده کردیم. از این دیتابست دو کلاس `Ice cream` و `pizza` را استفاده می کنیم زیرا می دانیم این تصاویر برای `Imagenet` قابل شناسایی است و جزو کلاس های آن می باشد. در ادامه نیز با استفاده از `transfer learning` و البته با `include top=False` به عنوان آرگومان آن با استفاده از مدل مناسب خطای `loss` برای آموزش و ارزیابی را به دست آوردیم. (مدل توسط مجموعه داده آموزش داده شده است. نتایج مطابق شکل زیر می باشد:

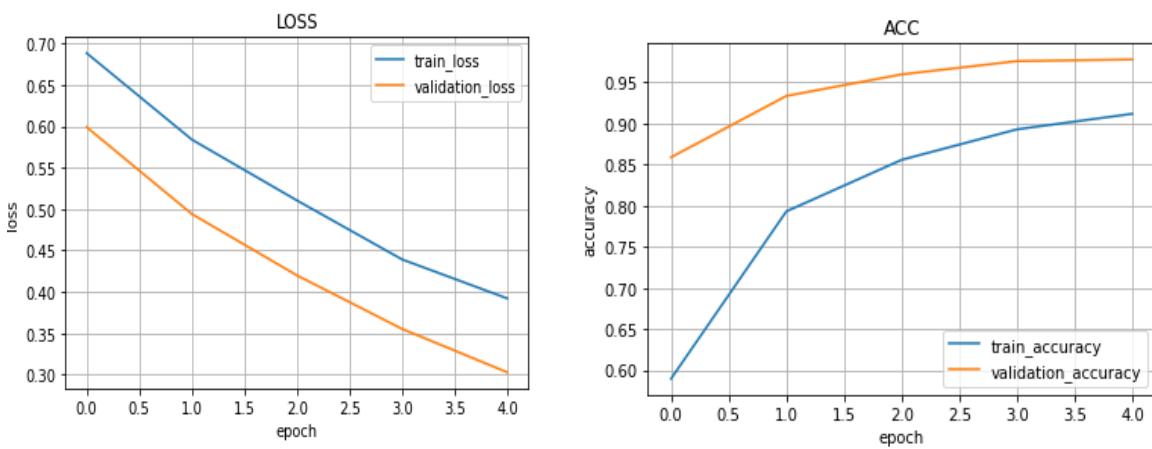
نمودارها نشان می دهند که دقت مدل با `Epoch` ها افزایش یافته `LOSS` کاهش یافته است .

دقت اعتبارسنجی در بسیاری از دوره ها از دقت آموزشی بالاتر بوده است

این می تواند به چند دلیل باشد:

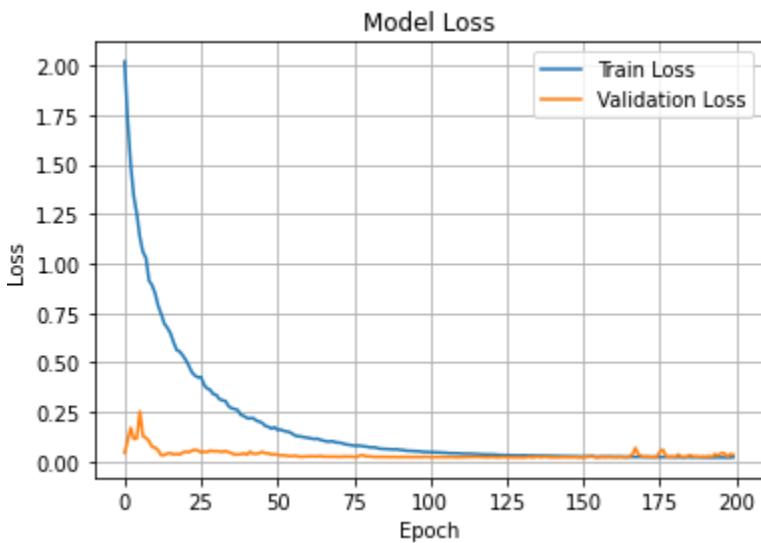
- ما از یک مدل از پیش آموزش دیده شده در `ImageNet` استفاده کردیم که حاوی داده هایی از کلاس های مختلف است

- استفاده از dropout می تواند به دقت اعتبار سنجی بالاتر منجر شود.(درانجا استفاده کردیم)



سوال 4

برای این سوال ابتدا دیتا را preprocess کردیم و دیتاست را لود کردیم. یعنی ابتدا دیتاست را آپلود کردیم و سپس از طریق دستورات os آدرس هر image را ساختیم در مرحله بعد نوبت به این رسید که تصاویر را بخوانیم. بعد از ساختن تصاویر برای Ny را مطابق آن چه در مقاله گفته شده بود ساختیم و سپس از روی Ny بردار y را به دست آوردیم. سپس مطابق آن چه در صورت سوال گفته شده است دیتا را به سه قسمت آموزش، اعتبارسنجی و تست با نسبت های 70 و 15 و 15 تقسیم کردیم. سپس شبکه را تشکیل دادیم و مدل داده شده را ترین کردیم. برای loss و valid loss :



و همچنین testloss داریم:

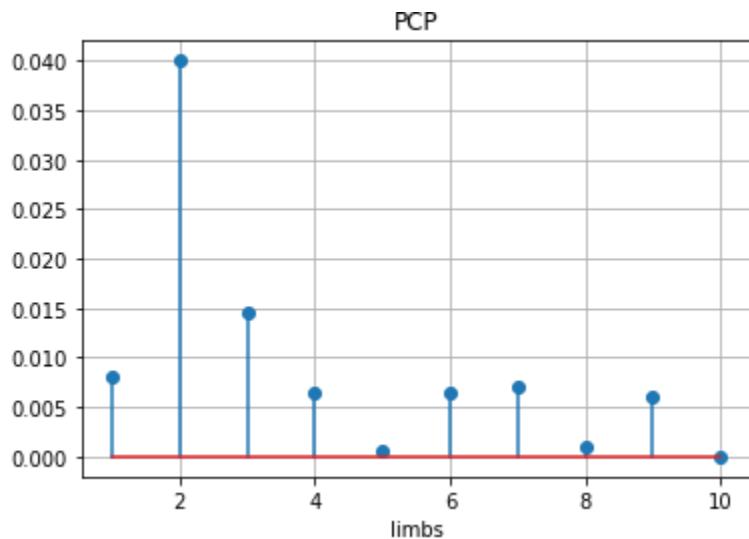
```
10/10 [=====] - 2s 92ms/step - loss: 0.0346
0.0345836877822876
```

همچنین باید توجه داشته باشیم که چون نرمالیزه می کنیم و ورودی ما نرمالیزه هست خروجی را نیز باید با اعمال عکس برگردانیم تا بتوانیم محاسبات نهایی را انجام دهیم البته معیار دوم بر اساس نرمالیزه سنجیده می شود.

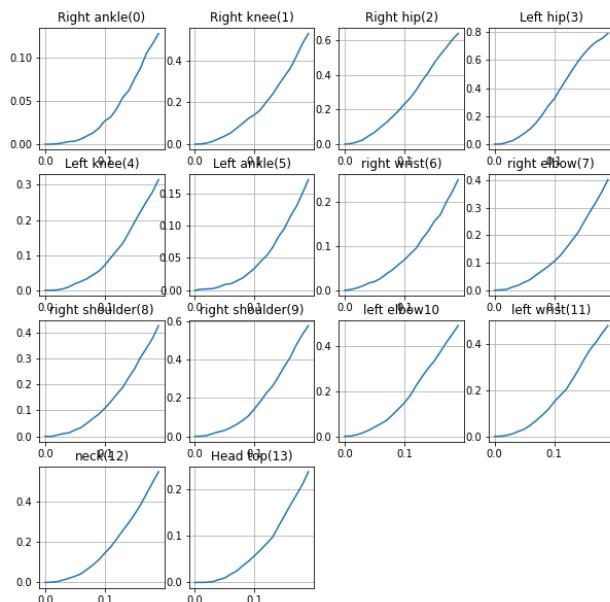
برای معیار PCP توضیح مختصر آن است که عدد های پریدیکت شده برای یک اندام (شامل 2 مفصل) را مشاهده می کنیم و فاصله آن ها تا نقاط اصلی و متناظر را محاسبه و در نهایت جمع می کنیم اگر عدد به دست آمده کمتر از 0.5 برابر طول اندام مورد نظر(واقعی) باشد آن گاه مورد

قبول در غیر این صورت قابل قبول نیست. برای PDJ اما بحث در مورد مفاصل است و نه اندام.

برای نمودارهای بخش PDJ داریم :



برای PDJ داریم:



همچنین برای چند تصویر نیز داریم :

