


نکات:

1. برای اجرا گرفتن سوالات اول و سوالات دوم باید از مسیر های HW04/Q1/HW04_Q1.ipynb و HW04/Q2/HW04_Q2.ipynb به ترتیب استفاده کرد.
2. این تمرین با استفاده از Goggle Colab زده شده است و همان فایل های بالا به تنهایی قابل استفاده هستند. باقی فایل های مجاورشان نیز نیازی نیست، زیرا به هنگام اجرا خودشان دوباره نیز دانلود می شوند تا بتوان آن ها را در سیستم کولب استفاده کرد.
3. در برخی قسمت ها نوار های سبز رنگی مانند زیر وجود دارد.

100%  10/10 [00:31<00:00, 3.11s/it]

این نوار ها در حقیقت یک progressbar است که تعداد ایتريشن ها و سرعت پیشرفت هر ایتريشن کنارش نوشته شده است که در حالت در حال اجرا به شکل زیر است.

40%  4/10 [00:05<00:08, 1.49s/it]

4. ممکن است فایلی را در صورت سوال خواسته باشید که به علت حجم نتوانسته باشم در سایت آپلود کنم و برای همین در گیت هاب به عنوان یک فضای ابری و راحت قرار داده ام. لینک گیت هاب من به شرح زیر است:

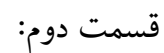
<https://github.com/MohammadRaziei/Deep-Learning-Course/tree/master/HW04>

پاسخ سوالات:

سوال اول:

قسمت اول:

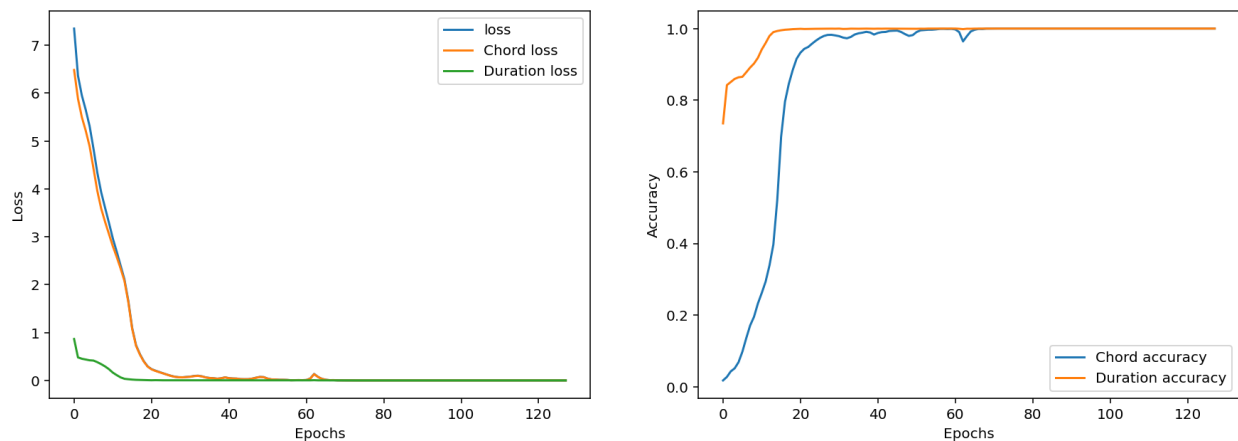
چون در این سوال ما میخواهیم موزیک مصنوعی بسازیم، بنابراین مشابه ایده شبکه های GAN در حقیقت میخواهیم توزیع احتمالی آن را یادگرفته و مدل سازی کنیم. اما هنگامی که از شبکه RNN استفاده میکنیم، بیشترین تمرکز روی یادگیری ترتیب آن ها است. بنابراین بهتر است از موسیقی هایی با ویژگی های مشابه استفاده کرد تا توزیع احتمالی آن ها مشابه یکدیگر باشند و توان پردازشی ما روی یادگیری ترتیب آن ها معطوف شود.



قسمت سوم:

The diagram illustrates a neural network architecture for predicting chord and duration. It starts with two input layers: `input_chord: InputLayer` and `input_duration: InputLayer`, both with input and output shapes of `[(None, None)]`. These feed into `embedding_chord: Embedding` and `embedding_duration: Embedding` layers, which have input shapes of `(None, None)` and output shapes of `(None, None, 64)`. The outputs of these embedding layers are concatenated in the `concat: Concatenate` layer, resulting in an input shape of `[(None, None, 64), (None, None, 64)]` and an output shape of `(None, None, 64)`. This is followed by a `gru: GRU` layer with input shape `(None, None, 64)` and output shape `(None, None, 256)`. The output of the GRU layer feeds into a `simple_rnn: SimpleRNN` layer with input shape `(None, None, 256)` and output shape `(None, 128)`. This is followed by a `dense: Dense` layer with input shape `(None, 128)` and output shape `(None, 64)`. The output of the dense layer feeds into a `dropout: Dropout` layer with input and output shapes of `(None, 64)`. Finally, the output of the dropout layer is split into two output layers: `output_chord: Dense` with input shape `(None, 64)` and output shape `(None, 3048)`, and `output_duration: Dense` with input shape `(None, 64)` and output shape `(None, 18)`.

قسمت چہارم:



قسمت پنجم:

این قسمت باید پکیجی را در کنار music21 نصب میکردم که مشکل داشت برای گوش دادن به این قسمت از موزیک پلیر سیستم خودم که pot-player است استفاده کردم. مشکلی این بود که انگار همش داشت تکرار میشد و به طرز خیلی مصنوعی، تناوبی شده بود.

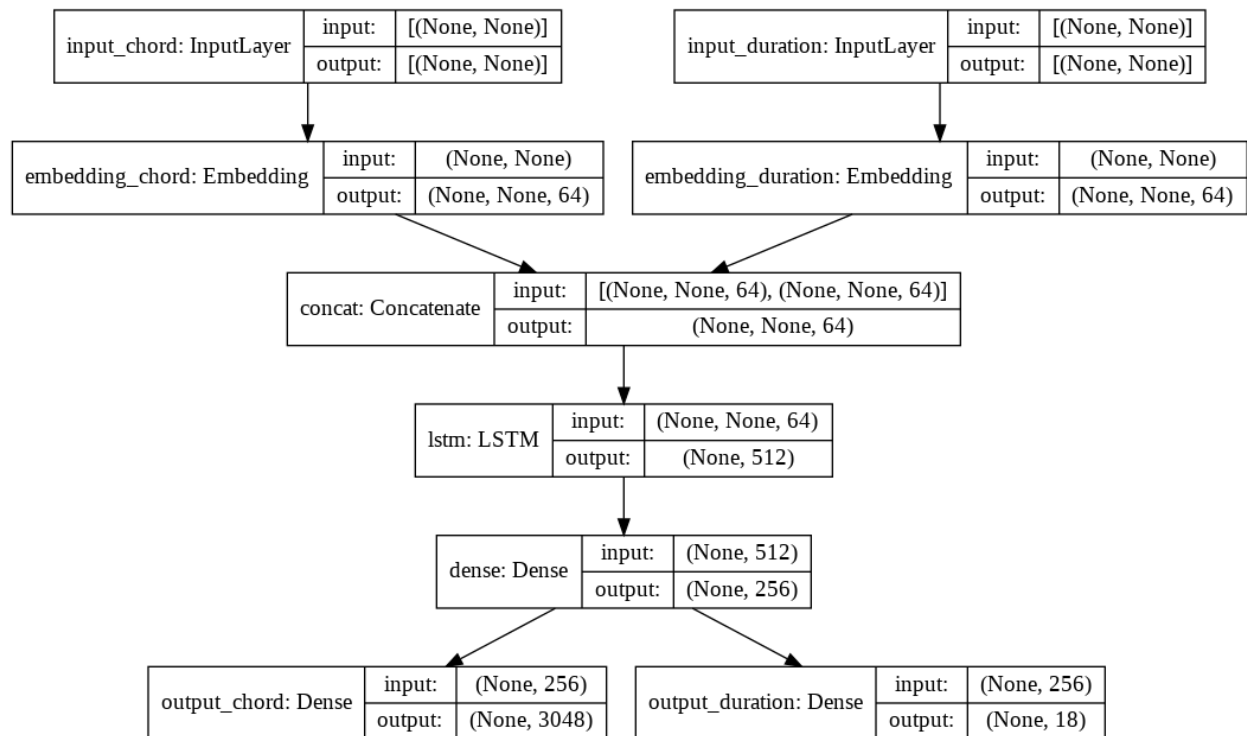
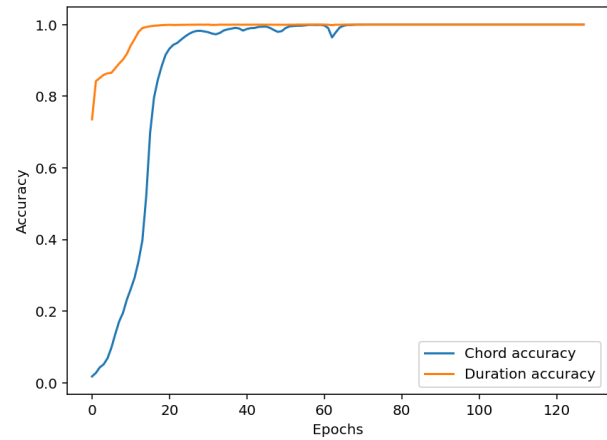
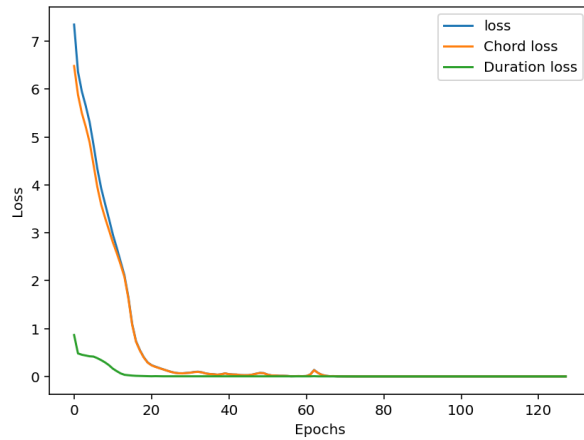
قسمت ششم:

برای حل این قسمت یا باید داده ها را با توزیع متفاوت تر نیز انتخاب کرد به طوری که خیلی هم توزیع داده ها پراکنده نشود یعنی همچنان عمومیت داده ها توزیعشان یکی باشد یا اصلا از این روش استفاده نکرد و از بلوک lstm استفاده کرد.

قسمت هفتم:

استفاده از بلوک lstm باعث بهتر شدن کیفیت خروجی گیتار ها شده است و اون مشکل نوسانی بودن هم حل شده است.

همچنین از نمودار هایشان مشخص است که خیلی سریع تر روی داده ها فیت شده است.



سوال دوم:

قسمت اول:

لطفا کد های `prepare_data.py` و `visual_dataset.py` و نحوه حل این سوال را ملاحظه کنید.

قسمت دوم:

در این سوال در ابتدا تلاش شده است تا تمامی کلاس ها از تمام تصاویر جمع آوری شود. حتی داده های نگاتیو هم لود شده اند که به عنوان کلاس صفر مورد استفاده قرار گرفته است.

برای حل مشکل عدم تعادل چهار راه به ذهن می رسد که دو تکنیک از این چهار راه حل پیاده سازی شده است.

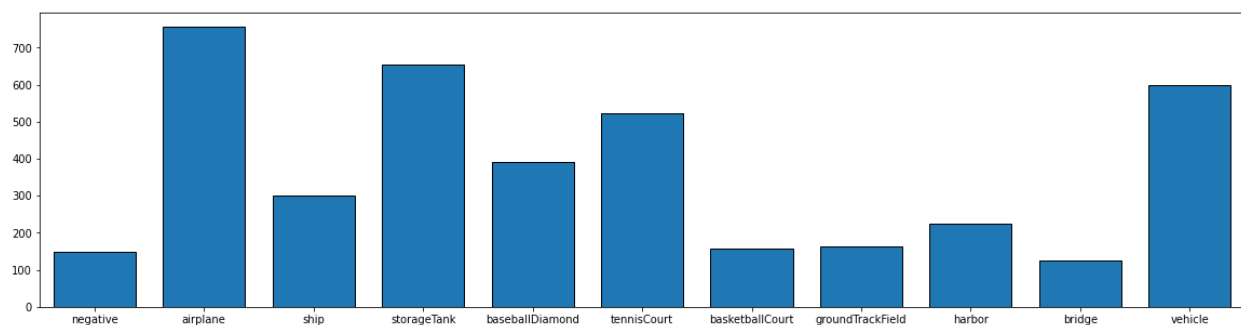
1. همه را ابتدا با توجه به کلاسی هستند به صورت رندم shuffle کرد، سپس به تعداد داده های مینیمم کلاس های موجود را انتخاب میکنیم. چون یک بار shuffle کردیم میتوانیم همان تعداد را از ابتدای لیست استخراج کنیم.

2. روش دومی که من آن را پیاده سازی کردم، مشابه روش قبلی است اما چون در روش بالا باید یک بار تمام داده ها ذخیره شوند و در مرحله بعدی انتخاب یا حذف شوند، من یک تابع توزیع احتمالی را با توجه به تعداد کلاس های موجود بدست آوردم و با استفاده از آن، هنگام تشکیل دیتا ست، روی هر عکس تصمیم میگیریم که به دیتا ست اضافه اش کنیم یا نکنیم. یعنی احتمالاتی تر از قبلی است و در حافظه و پردازش بسیار صرفه جویی می شود ولی لزوماً تعداد داده ها کاملاً یکسان نمی شوند زیرا تعدادشان نیز تصادفی است، اما با توجه به سیاست حذفی که صورت گرفته است بسیار نزدیک هم می باشد که جزییات آن را جلو تر توضیح میدهم.

3. استفاده از یک ضریب برای هر کلاس که در هنگام آموزش خودش رو باهاش وفق بده.

4. استفاده از دیتا آگمنتیشن. یعنی یک سقف قرار میدهم و سعی میکنیم با آگمنت کردن داده ها به آن برسیم.

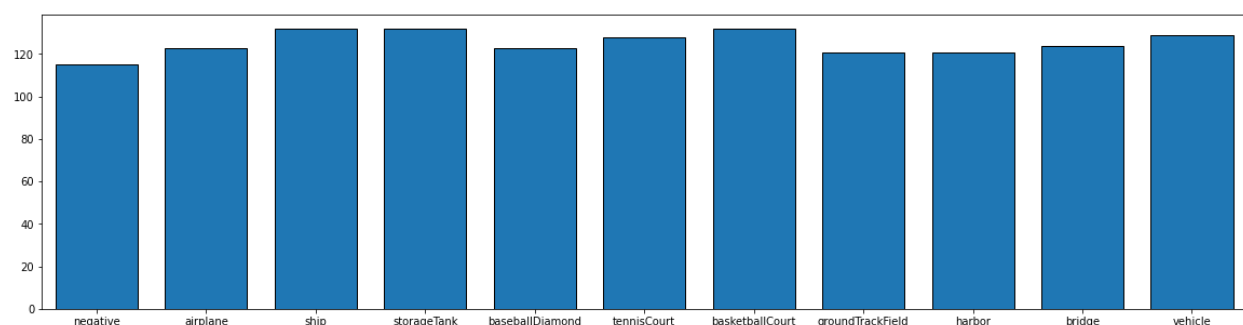
در ابتدا صرفاً از روی فایل های txt موجود تعداد تک تک کلاس ها را در می آوریم که مطابق زیر غیر بالانس است. تعداد هر کدام از این کلاس ها G_i می نامیم.



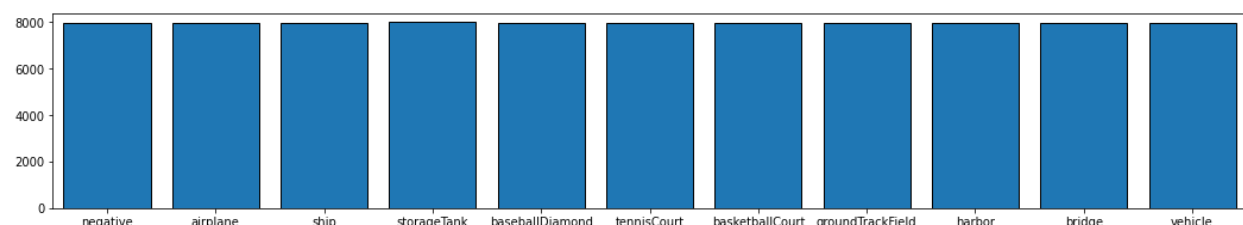
بعدش کمترین تعداد را C_{min} می‌نامیم. یعنی $C_{min} = \min C_i$ و با استفاده از آن یک تابع احتمال را برای حذف یا نگه داری آن نگه می‌داریم. این تابع باید کمترین داده را با احتمال صفر حذف کند با افزایش تعداد هر کلاس احتمال آن افزایش یابد. برای همین به صورت زیر تعریف کردم.

$$P_i(\text{removal}) = \frac{C_i - C_{min}}{C_i}$$

مشخص است تابع بالا تمام خواص مطلوب را دارد. سپس قبل از ذخیره سازی دادگان آن‌ها حذف می‌کنم. این حالت که بسیار در حافظه و پردازش صرفه جویی کرده به نمودار زیر منتهی می‌شود.

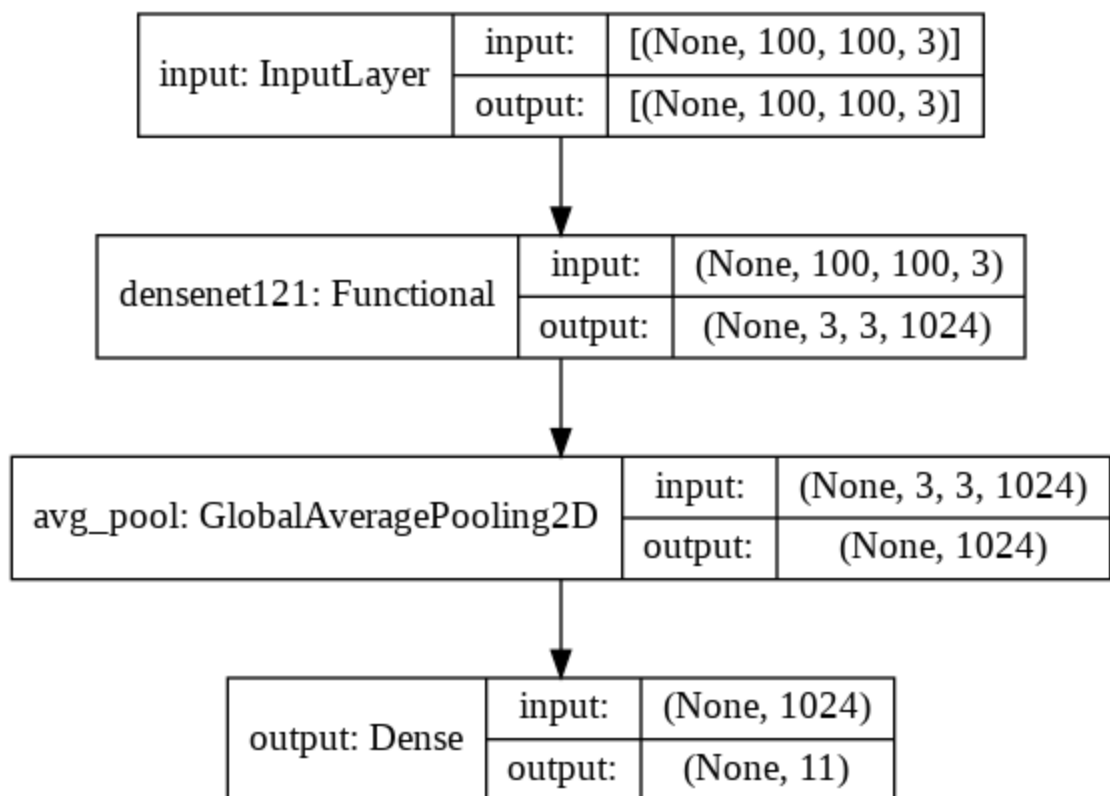


همچنین روش آگمنتیشن داده‌ها به تعداد 5 برابر ماکسیم آن‌ها که حدودا 8000 تاست، می‌رساند. باقیمانده آن‌ها را خودش ایجاد می‌کند.



قسمت سوم:

برای این قسمت از تکنیک‌های ترنسفر لرنینگ استفاده کردم و شبکه densenet121 را فریز کردم و در ساختار زیر قرار دادم.



دقت عملکردی این روش در حالت های تست و ترین و ولیدیشن برای داده های بالانس با روش احتمالاتی 98.6 است و برای روش آگمنت شده، 97 است.

قسمت چهارم:

برای این کار یک کلاسی نوشتم که از ابزار های keras ارثبری میکند و در فایل transform.py قرار دادم. دو تصویر نیز برای نشان دادن قابلیت های این تصویر از سایتی که لینکش را در آن فایل قرار دادم دانلود کردم تا عملکرد های مختلف آگمنت کردن را نشان بدهم و بعد روی تصاویر اعمالش کردم. همچنین چون از کولب استفاده کردم ممکنه رمش در این قسمت پر بشه و کرش کند. برای این مشکل چنین بار داده ها را در فایل سیو کردم و رم را پاک کردم و دوباره آن ها را بارگذاری کردم.

قسمت پنجم:

نمودار این قسمت را یادم رفت رسم کنم ولی داده هاش به صورت زیر است.

```
Epoch 1/20
56/56 - 16s - loss: 0.3018 - accuracy: 0.3715 - val_loss: 0.1851 -
val_accuracy: 0.7692
Epoch 2/20
```

56/56 - 2s - loss: 0.1403 - accuracy: 0.8788 - val_loss: 0.1114 -
val_accuracy: 0.9095
Epoch 3/20
56/56 - 2s - loss: 0.0881 - accuracy: 0.9592 - val_loss: 0.0787 -
val_accuracy: 0.9593
Epoch 4/20
56/56 - 2s - loss: 0.0638 - accuracy: 0.9773 - val_loss: 0.0608 -
val_accuracy: 0.9683
Epoch 5/20
56/56 - 2s - loss: 0.0498 - accuracy: 0.9830 - val_loss: 0.0521 -
val_accuracy: 0.9683
Epoch 6/20
56/56 - 2s - loss: 0.0413 - accuracy: 0.9898 - val_loss: 0.0451 -
val_accuracy: 0.9729
Epoch 7/20
56/56 - 2s - loss: 0.0349 - accuracy: 0.9853 - val_loss: 0.0400 -
val_accuracy: 0.9819
Epoch 8/20
56/56 - 2s - loss: 0.0303 - accuracy: 0.9909 - val_loss: 0.0360 -
val_accuracy: 0.9819
Epoch 9/20
56/56 - 2s - loss: 0.0267 - accuracy: 0.9921 - val_loss: 0.0332 -
val_accuracy: 0.9819
Epoch 10/20
56/56 - 2s - loss: 0.0237 - accuracy: 0.9898 - val_loss: 0.0311 -
val_accuracy: 0.9864
Epoch 11/20
56/56 - 2s - loss: 0.0215 - accuracy: 0.9921 - val_loss: 0.0294 -
val_accuracy: 0.9819
Epoch 12/20
56/56 - 2s - loss: 0.0194 - accuracy: 0.9921 - val_loss: 0.0281 -
val_accuracy: 0.9864
Epoch 13/20
56/56 - 2s - loss: 0.0176 - accuracy: 0.9921 - val_loss: 0.0266 -
val_accuracy: 0.9819
Epoch 14/20
56/56 - 2s - loss: 0.0163 - accuracy: 0.9921 - val_loss: 0.0256 -
val_accuracy: 0.9864
Epoch 15/20
56/56 - 2s - loss: 0.0149 - accuracy: 0.9932 - val_loss: 0.0243 -
val_accuracy: 0.9864
Epoch 16/20
56/56 - 2s - loss: 0.0138 - accuracy: 0.9943 - val_loss: 0.0237 -
val_accuracy: 0.9864
Epoch 17/20
56/56 - 2s - loss: 0.0128 - accuracy: 0.9955 - val_loss: 0.0227 -
val_accuracy: 0.9864
Epoch 18/20
56/56 - 2s - loss: 0.0119 - accuracy: 0.9955 - val_loss: 0.0216 -
val_accuracy: 0.9774
Epoch 19/20
56/56 - 2s - loss: 0.0112 - accuracy: 0.9955 - val_loss: 0.0214 -
val_accuracy: 0.9864
Epoch 20/20

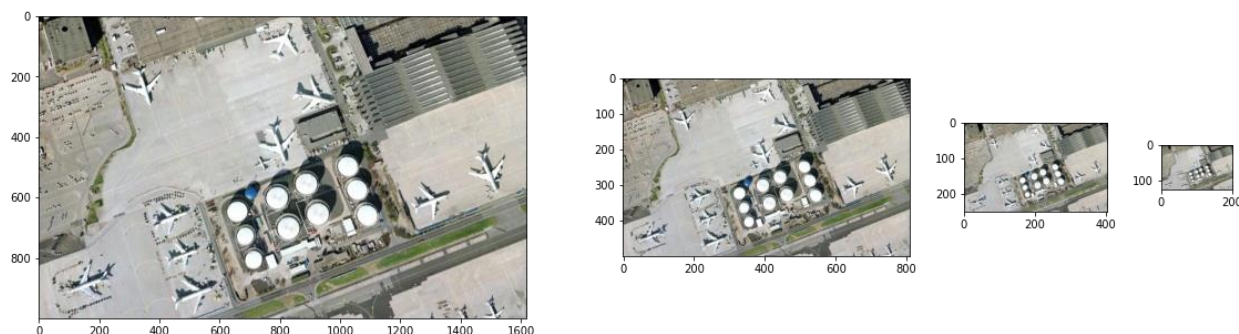

```
56/56 - 2s - loss: 0.0105 - accuracy: 0.9955 - val_loss: 0.0205 -  
val_accuracy: 0.9864
```

Saved model to disk

که نشان میدهد چقدر سریع به 98 درصد رسیده است. این که برخی لیبیل ها اشتباه است هم نشان میدهد که مدلم دقیقا روی اون داده های اشتباه کار نکرده و احتمالا به خوبی تونسته ترین شود.

قسمت ششم:

من با استفاده از تکنیک pyramid تصاویر زیر را در ابعاد خواسته شده در اوردم



حالا گفتم پنجره من 100 در 100 است و این طول را برابر 0.8 کوچکترین عرض تصاویر pyramid در نظر گرفتم. یعنی کوچکترین تصویر برام بزرگترین زاویه دید رو میسازه... برای همین اون نسبت را در نظر گرفتم و بعد پنجره رو روی تمام اون تصاویر لغزوندم ولی خوب عمل نکرد! که تو زیر میبینیم. این خوب عمل نکردن برام بسیار عجیب بود چون من دقت بالایی روی داده های تست داشتم.

