بسمه تعالی

دانشکده مهندسی کامپیوتر

یادگیری عمیق

نام استاد: دکتر محمدی

تمرین سیزدهم

آرمان حیدری

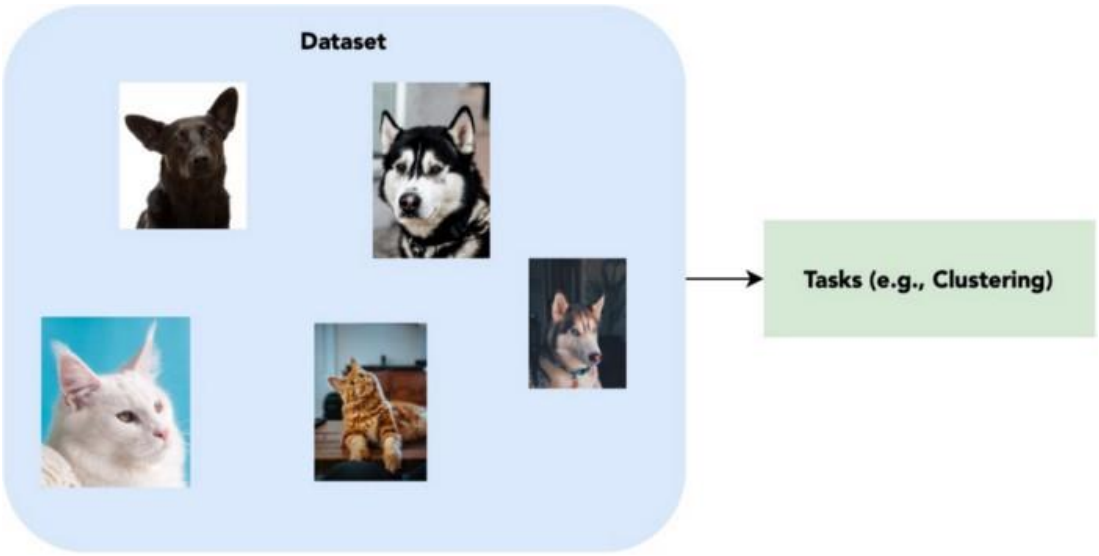
شماره دانشجویی: ۹۷۵۲۱۲۵۲

آذر ۱۴۰۰

فهرست

پاسخ سوال اول

- Unsupervised learning: نوعی از یادگیری بدون ناظر است.به طور کلی یعنی داده های موجود در دیتاست آموزشی، هیچگونه label ای ندارند. در این شرایط جوابی کاملا درست وجود ندارد و سعی میکنیم داده ها را cluster کنیم. مثلا عکس زیر را میبینیم :



در این نوع از یادگیری، نمیدانیم که سگ ها و گربه ها کدام هستند ولی مدل باید یاد بگیرد که دو گروه داده داریم و هر کدام شامل کدام یک از این دسته ها هستند. (به دو دسته cluster کند و خودش label بزند)

- Self-supervised learning: زیرمجموعه ای از حالت قبل است چون باز هم دیتاست ما label ندارد. که به عنوان self supervision نیز شناخته می شود، یک راه حل نوظهور برای چالش ناشی از برچسب گذاری داده ها است. یادگیری خود نظارتی یک رویکرد یادگیری ماشینی است که در آن مدل با استفاده از یک بخش از داده ها برای پیش بینی بخشی دیگر و تولید برچسب ها به طور دقیق، خود را آموزش می دهد. در پایان، این روش یادگیری یک مسئله یادگیری بدون نظارت را به یک مشکل تحت نظارت تبدیل می کند. در زیر نمونه ای از خروجی یادگیری خود نظارت شده است:

(a) Input context (b) Output

روش‌های یادگیری خود نظارتی برای یادگیری ویژگی‌های عمومی از داده‌های بدون برچسب در مقیاس بزرگ پیشنهاد شده‌اند.

- Representation learning: یادگیری بازنمایی حوزه ای از تحقیقات است که بر نحوه یادگیری نمایش های عددی و فشرده برای دیتا تمرکز دارد. این دیتاها اغلب ویدئو، متن، صدا و تصویر هستند. هدف این تحقیق استفاده از این نمایش‌ها برای کارهای دیگر مانند جستجو برای اطلاعات است.
یک مثال شناخته شده در این مورد هنگام جستجوی ویدیوها در یوتیوب است: کاربر کلمات کلیدی متنی را ارائه می دهد که در نتیجه یوتیوب مجموعه ای از ویدیوها را که شبیه به آن کلمات هستند را برمی گرداند.
در ادبیات بینایی کامپیوتر، بازنمایی ها با آموزش یک مدل یادگیری عمیق برای تبدیل ورودی خام به یک بردار عددی آموخته می شوند. هنگام جاسازی فیلم، صدا یا متن، بردارهای عددی اغلب برای حفظ روابط زمانی چند بعدی هستند. روش هایی که محققان این مدل ها را آموزش می دهند به شدت متفاوت است.

با توجه به توضیحات داده شده یادگیری بازنمایی، می تواند شامل داده هایی با برچسب و یا بدون برچسب باشد. به همین خاطر اشتراکی با حالات قبلی دارد و بخشی از آن هم می تواند برای دیتا های supervised باشد. مانند پیش آموزش دادن مدل روی دیتاست imagenet که متشکل از ۱۰۰۰ دسته کلاس تصاویر مختلف است.

منابع

https://research.aimultiple.com/self-supervised-learning/

https://towardsdatascience.com/supervised-semi-supervised-unsupervised-and-self-supervised-learning-7fa79aa9247c

https://towardsdatascience.com/tagged/representation-learning

اسلاید جلسه ۲۵ درس

پاسخ سوال دوم

الف)

معیار utility را بعنوان سودمندی self-supervised تعریف کرده که برای صرفه جویی در استفاده از داده های دارای label میباشد. یعنی برای دستیابی به همان دقت بدون self-supervision، چند برچسب دیگر لازم است. اگر میزان دقت مدل self-supervised با همان تعداد برچسب با دقت مدل برابر شود، U(n)برابر با صفر می شود. اگر هیچ تعداد برچسبی وجود نداشته باشد که که دقت این دو مدل باهم برابر شود، U(n) به بی نهایت میل می کند.

$$U(n) = \frac{n1}{n} - 1$$

که اگر a(n) میزان دقت مدلی است که از ابتدا با n برچسب آموزش داده شده است. و $a_{ft}(n)$ میزان دقت مدلی است که fine tune شده باشد. و n1 را تعداد برچسب هایی تعریف میکنیم که لازم است تا $a(n1) = a_{ft}(n)$.

ب)

Object classification: مدل را آموزش داده تا بین ده کلاس ShapeNet که برای ارائه داده‌های مصنوعی استفاده می‌شوند بتواند تفکیک کند. این تصاویر تنها شامل یک شی هستند و توزیع یکسانی میان داده ۱۰ کلاس وجود دارد. عملکرد با میزان دقت سنجیده شده است.

Object Pose Estimation: برای این تسک دوباره از داده هایی استفاده میشود که یک شی دارند که در وسط تصویر میباشد. برای این تسک در این مقاله گفته شده است که که pose آبجکت داخل تصویر را به ۵ تا قسمت تبدیل کرده و با استفاده از آن classifier را آموزش میدهیم. یکی از دلایل برای قاب بندی مسئله به این شکل، در نظر گرفتن تقارن چرخشی موجود در برخی از دسته بندی های ShapeNet است. پنج حالت انتخاب شده به گونه ای انتخاب میشوند که جهت گیری در امتداد آن محور چرخشی نادیده گرفته می شود . توزیع نمونه ها در هر ۵ دسته به طور مساوی میباشد. این مدل همچنان برای تخمین pose نیاز دارد که مدل ویژگی‌های مربوط به درک سه بعدی را استخراج کند اما مقدار سختی و پیچیدگی راهنگام supervising و evaluating را کاهش میدهد. برای اموزش مدل از تابع خطا cross entropy استفاده شده و دقت دسته بندی را در نهایت گزارش میکنند.

Semantic segmentation: تصاویر با چندین شی ترکیب می شوند. در این تسک هدف آموزش مدلی که بتواند ماسک های دقیق و دارای رزولوشن بالا پیش بینی کند نیست، بلکه ماسک ها رزولوشن بسیار درشت تری نسبت به تصویر ورودی دارند. از تابع ضرر cross entropy برای هر پیکسل استفاده شده است.

Depth Estimation: همچون تسک قبل، در تصاویر این تسک نیز چندین شی وجود دارند. برای آموزش مدلی که بتواند عمق اشیا را تشخیص دهد از L1 loss استفاده شده و برای گزارش میزان دقت، از درصد پیش بینی هایی که در نسبت معینی از عمق ground truth قرار گرفته اند استفاده می شود.

پ)

از ۴ روش مختلف در این مقاله استفاده شده است:

Variational autoencoder (VAE): یک پایه استاندارد و تعیین شده برای نگاشت تصاویر به یک فضا با ابعاد کمتر است. این ایده بسیار ساده است و شامل تنظیم یک encoder و decoder به عنوان شبکه های عصبی و یادگیری بهترین طرح encode-decode با استفاده از یک فرآیند بهینه سازی تکراری است.

Rotation: این شبکه وظیفه دارد پیش بینی کند که آیا یک تصویر ۰، ۹۰، ۱۸۰ یا ۲۷۰ درجه چرخیده است.

Contrastive Multiview Coding: در این روش تصویر را بر اساس کانال های آن اسپلیت میکنیم. مثلا اگر تصویر در فضای Lab باشد آن را به L و ab اسپلیت می کنیم. سپس آنها را از دو نیمه شبکه عبور میدهیم و embeddingهای خروجی باهم و در تضاد با embedding های تصاویر دیگر مقایسه می شوند.

Augmented Multiscale Deep InfoMax: مشابه روش قبل، این روش یک مدل را نیز از طریق کدگذاری متضاد آموزش می دهد. به جای مقایسه بین کانال‌های تصویر، نمایش‌های دو نسخه تقویت‌شده یک تصویر و همچنین نمایش‌های تولید شده در لایه‌های میانی شبکه را مقایسه می‌کند.

پاسخ سوال سوم

پیاده سازی این سوال در فایل HW13.ipynb تکمیل شده است. (لینک گوگل کولب)

در سلول اولی که باید پیاده سازی کنیم، جملات را به برداری از کلمات تبدیل میکنیم. (با استفاده از دیکشنری ورودی wrd_idx)

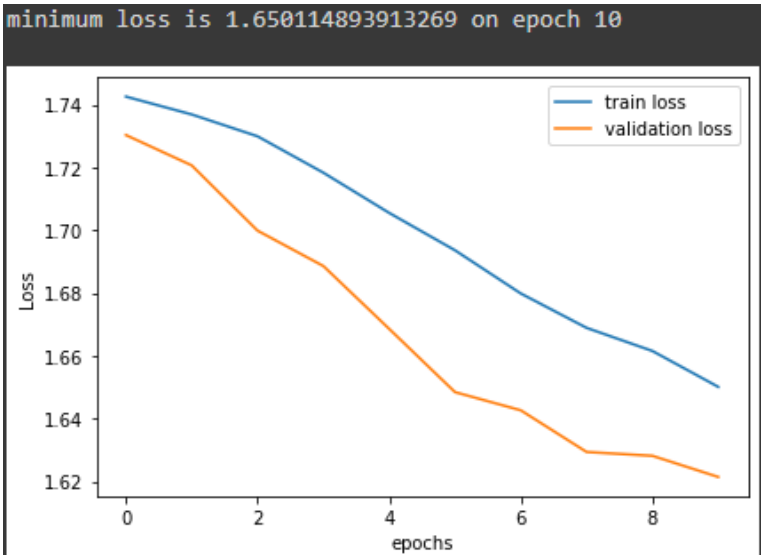
سلول های دوم و سوم ساده هستند و فقط مربوط به پلات کردن تابع ضرر و دقت، به همراه پیدا کردن مینیمم و ماکسیمم برای آن هاست.

و آخرین سلول هم باید مدلی که در سلول های بعدی summary و شکل آن را میبینیم پیاده سازی کرده ایم.

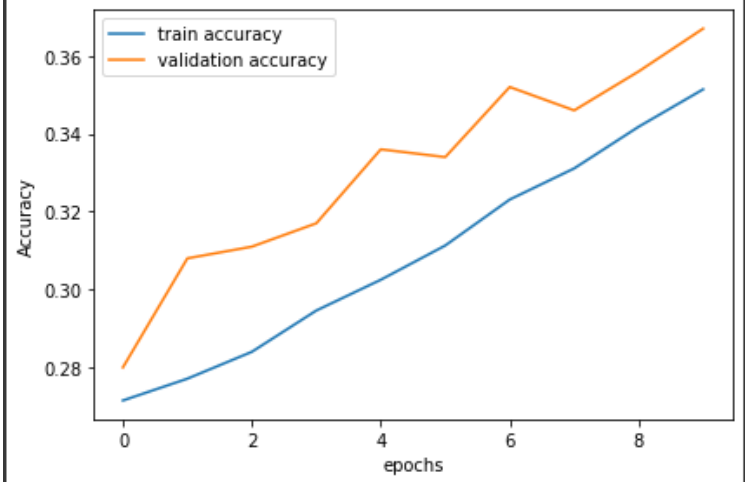
خروجی آموزش مدل به این صورت بود:

```
Epoch 1/10
  1/313 [..............................] - ETA: 16s - loss: 1.6621 - accuracy: 0.4062/usr/local/lib/python3.7/dist-packages/tensorflow
  "Even though the `tf.config.experimental_run_functions_eagerly` "
313/313 [==============================] - 16s 52ms/step - loss: 1.7425 - accuracy: 0.2715 - val_loss: 1.7303 - val_accuracy: 0.2800
Epoch 2/10
313/313 [==============================] - 16s 51ms/step - loss: 1.7368 - accuracy: 0.2771 - val_loss: 1.7206 - val_accuracy: 0.3080
Epoch 3/10
313/313 [==============================] - 16s 52ms/step - loss: 1.7299 - accuracy: 0.2840 - val_loss: 1.6998 - val_accuracy: 0.3110
Epoch 4/10
313/313 [==============================] - 17s 56ms/step - loss: 1.7183 - accuracy: 0.2946 - val_loss: 1.6886 - val_accuracy: 0.3170
Epoch 5/10
313/313 [==============================] - 16s 52ms/step - loss: 1.7055 - accuracy: 0.3025 - val_loss: 1.6686 - val_accuracy: 0.3360
Epoch 6/10
313/313 [==============================] - 16s 52ms/step - loss: 1.6936 - accuracy: 0.3113 - val_loss: 1.6484 - val_accuracy: 0.3340
Epoch 7/10
313/313 [==============================] - 16s 51ms/step - loss: 1.6798 - accuracy: 0.3231 - val_loss: 1.6427 - val_accuracy: 0.3520
Epoch 8/10
313/313 [==============================] - 16s 51ms/step - loss: 1.6689 - accuracy: 0.3311 - val_loss: 1.6294 - val_accuracy: 0.3460
Epoch 9/10
313/313 [==============================] - 16s 51ms/step - loss: 1.6615 - accuracy: 0.3418 - val_loss: 1.6282 - val_accuracy: 0.3560
Epoch 10/10
313/313 [==============================] - 16s 51ms/step - loss: 1.6501 - accuracy: 0.3514 - val_loss: 1.6214 - val_accuracy: 0.3670
```

و نمودار های به دست آمده از روند آموزش:



```
maximum accuracy is 0.3513999879360199 on epoch 10
```



چون ۱۰ ایپوک مقدار زیادی نیست مدل underfit شده است. البته از روند نمودارها مشخص است که با ادامه دادن آموزش میتوان به نتایج بهتری دست یافت. چون علاوه بر صعودی بودن دقت و نزولی بودن تابع ضرر، نتایج داده ارزیابی هم همراه داده آموزشی پیشرفت میکنند و این نشان از overfit نشدن مدل است. و میبینیم که کمترین loss و بیشترین accuracy هم توسط تابع محاسبه شده و در epoch آخری که اجرا کرده ایم بوده است. که همین نشان از روند خوب مدل هم دارد.

یک نمونه تست هم پس از آموزش مدل که به صورت دستی وارد کرده ام و نتیجه به این صورت بود:

```
--------------------------------------------------------------------------------
Custom User Queries (Make sure there are spaces before each word)
--------------------------------------------------------------------------------
Please input a story
Mary went to the bathroom . Sandra moved to the garden .
Please input a query
Where is Mary ?
Result
Mary went to the bathroom . Sandra moved to the garden . Where is Mary ? | Prediction: garden
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops.py:4527: UserWar
  "Even though the `tf.config.experimental_run_functions_eagerly` "
```

به خوبی پس از گرفتن سوال و جملات، جواب که ‘garden’ است را تشخیص داده است.

به طور کلی این مدل سعی می کند با دریافت چند جمله به عنوان یک قصه آن را یاد بگیرد. بعد سوالی را مربوط به آن جملات پاسخ می دهد. اول لایه امبدینگ story و لایه امبدینگ query در هم ضرب داخلی می شوند و سپس امبدینگ story را به آن اضافه میکنیم. از یک لایه LSTM هم برای یادگیری بهتر ترتیبی استفاده میکنیم. چون میدانیم لایه های بازگشتی در پردازش متون کاربرد زیادی دارند.

از معایب این مدل میتوان به این اشاره کرد که فقط همان ۲۱ کلمه موجود در دیکشنری را میشناسد. و در سلول آخر که باید خودمان داده وارد کنیم، حتما باید از آن کلمات استفاده کنیم. چون در غیر اینصورت نمیتواند جمله را به برداری از اعداد تبدیل کند. همینطور این که فقط نوع خاصی از داده ها که با space جدا شده اند و بین story های مختلف حتما . باشد را میتواند تشخیص بدهد.

مزیت های مهم آن هم یکی تعداد نسبتا کم پارامترهای قابل آموزش به دلیل پیچیده نبودن شبکه است. و در نتیجه زمان آموزش چندان طولانی نیست. از sequential های موازی استفاده میکند و در نتیجه خیلی خوب سوال و جواب را باهم ترکیب کرده است.

برای استفاده از این مدل در کارهای واقعی، باید انعطاف آن را بیشتر کنیم. یعنی در برابر نویز هایی مثل نبود space بین کلمات مختلف و حتی نبودن . بین جملات آن را robust کنیم. استفاده از دیتاست واقعی تر مثلا متون موجود در وبسایت ها میتواند بسیار مفید باشد. این مدل میتواند با تغییراتی، به مدل خوبی برای پرسش و پاسخ تبدیل شود. مثلا حذف stop word ها در ابتدای پروسس تغییر مفیدی است.

همچنین ایرادی که بالاتر ذکر شد هم باید رفع کنیم. مثلا استفاده از سیستم های word2vec آماده که کتابخانه های آن هم (nltk) موجود است، میتواند مفید باشد. که بتوانیم هر کلمه ای را داخل داده های آموزشی و تست بیاوریم و مدل کار بکند. همچنین قاعدتا آموزش مدل باید طولانی تر باشد و تا جایی که به بیشترین دقت روی داده های تست برسد باید آموزش را ادامه دهیم.

پاسخ سوال چهارم

پیاده سازی این سوال در فایل HW13.ipynb انجام شده است (لینک گوگل کولب). که در سه قسمت مختلف part A مربوط به بخش الف، part B مربوط به بخش ب و part C مربوط به بخش پ است.

الف)

مدلی که در این قسمت و قسمت های بعدی استفاده کرده ایم، شبکه mobilenet-v2 است. چون میخواستم به اندازه کافی ظرفیت یادگیری داشته باشد و چندان هم سنگین نباشد. این مدل را به عنوان backbone استفاده میکنیم و در هر بخش لایه های پس از آن کمی تفاوت دارد. برای این قسمت از یک لایه flatten و سپس یک لایه dense با ۱۰ نورون و تابع فعالسازی softmax جهت تصمیمگیری استفاده میکنیم.

البته باید دقت کنیم که قبل از دادن y_train و y_test به شبکه با استفاده از تابع to_categorical موجود در تنسورفلو، آن ها را به فرم one hot در آورده ایم.

نتایج این بخش با اجرا فقط روی ۲۰۰ داده آموزشی برچسب دار و batch_size=64 و epochs=100:

```
Epoch 89/100
4/4 [==============================] - 0s 38ms/step - loss: 0.2535 - accuracy: 0.9500
Epoch 90/100
4/4 [==============================] - 0s 38ms/step - loss: 0.2144 - accuracy: 0.9400
Epoch 91/100
4/4 [==============================] - 0s 39ms/step - loss: 0.2560 - accuracy: 0.9250
Epoch 92/100
4/4 [==============================] - 0s 39ms/step - loss: 0.2616 - accuracy: 0.9300
Epoch 93/100
4/4 [==============================] - 0s 38ms/step - loss: 0.2230 - accuracy: 0.9200
Epoch 94/100
4/4 [==============================] - 0s 40ms/step - loss: 0.2530 - accuracy: 0.9300
Epoch 95/100
4/4 [==============================] - 0s 37ms/step - loss: 0.1972 - accuracy: 0.9450
Epoch 96/100
4/4 [==============================] - 0s 38ms/step - loss: 0.2104 - accuracy: 0.9200
Epoch 97/100
4/4 [==============================] - 0s 38ms/step - loss: 0.3329 - accuracy: 0.9100
Epoch 98/100
4/4 [==============================] - 0s 43ms/step - loss: 0.3357 - accuracy: 0.9200
```

که مطابق انتظار شبکه با این ظرفیت به سادگی روی داده های کم fitشده است. اما وقتی روی ۱۰۰۰۰ تست آن را evaluate میکنیم، میبینیم که به دقت بسیار بدی میرسیم که نشان از overfit کامل دارد.

```
313/313 [==============================] - 5s 16ms/step - loss: 2.3085 - accuracy: 0.1000
but accuracy is just 10.000000149011612 % on test data!
```

که با توجه به کم بودن داده آموزشی و پیچیدگی مسئله ی ۱۰ کلاسه کاملا طبیعی است.

ب)

در این بخش ابتدا دیتاستی با استفاده از ۴۹۸۰۰ داده بدون برچسب میسازیم. هر کدام را به صورت رندوم با استفاده از cv2.rotate()، ۰ یا ۹۰ یا ۱۸۰ یا ۲۷۰ درجه میچرخانیم. و برچسب را 0 به معنی چرخش ۰ درجه، 1 به معنی چرخش ۹۰ درجه، 2 به معنی چرخش ۲۷۰ درجه و 3 به معنی چرخش ۱۸۰ درجه قرار میدهیم.

بعد مدلی شامل موبایلنت ورژن ۲ + یک لایه flatten + یک لایه dense با ۴ نورون برای تصمیم گیری بین چرخش ۰ یا ۹۰ یا ۱۸۰ یا ۲۷۰ درجه تصویر طراحی میکنیم. با learning rate=0.01 و ثابت ماندن سایر هایپرپارامترها (و فقط با 50 ایپاک چون کمی آموزش زمانبر بود) این بار روی دیتاستی که ساختیم آن را آموزش میدهیم و به این نتایج میرسیم:

```
Epoch 38/50
779/779 [==============================] - 32s 41ms/step - loss: 0.4126 - accuracy: 0.8425
Epoch 39/50
779/779 [==============================] - 32s 41ms/step - loss: 0.4057 - accuracy: 0.8453
Epoch 40/50
779/779 [==============================] - 32s 41ms/step - loss: 0.3929 - accuracy: 0.8506
Epoch 41/50
779/779 [==============================] - 32s 41ms/step - loss: 0.3866 - accuracy: 0.8523
Epoch 42/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3771 - accuracy: 0.8572
Epoch 43/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3735 - accuracy: 0.8574
Epoch 44/50
779/779 [==============================] - 32s 41ms/step - loss: 0.3681 - accuracy: 0.8603
Epoch 45/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3687 - accuracy: 0.8583
Epoch 46/50
779/779 [==============================] - 32s 41ms/step - loss: 0.3565 - accuracy: 0.8642
Epoch 47/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3461 - accuracy: 0.8692
Epoch 48/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3386 - accuracy: 0.8719
Epoch 49/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3328 - accuracy: 0.8733
Epoch 50/50
779/779 [==============================] - 31s 40ms/step - loss: 0.3379 - accuracy: 0.8729
```

مدل کم کم به خوبی آموزش دیده است و به دقت قابل قبولی میرسد. حالا دقیقا همین وزن ها را نگه میداریم و فقط لایه آخر را با لایه جدیدی که به جای ۴ نورون، ۱۰ نورون دارد قرار میدهیم. و آن را روی ۲۰۰ داده برچسب دار آموزش میدهیم.(با همان هایپرپارامترهای قسمت الف، و تفاوت با قسمت قبل این است که learning rate = 0.001 میگیریم چون وزن ها تا حد خوبی به دست آمده اند. تا مقایسه دقیق باشد):

```
4/4 [==============================] - 0s 39ms/step - loss: 1.2487 - accuracy: 0.7100
Epoch 88/100
4/4 [==============================] - 0s 39ms/step - loss: 1.2494 - accuracy: 0.7100
Epoch 89/100
4/4 [==============================] - 0s 39ms/step - loss: 1.2268 - accuracy: 0.7200
Epoch 90/100
4/4 [==============================] - 0s 38ms/step - loss: 1.2163 - accuracy: 0.7200
Epoch 91/100
4/4 [==============================] - 0s 38ms/step - loss: 1.1928 - accuracy: 0.7450
Epoch 92/100
4/4 [==============================] - 0s 39ms/step - loss: 1.1904 - accuracy: 0.7300
Epoch 93/100
4/4 [==============================] - 0s 39ms/step - loss: 1.1806 - accuracy: 0.7400
Epoch 94/100
4/4 [==============================] - 0s 41ms/step - loss: 1.1577 - accuracy: 0.7100
Epoch 95/100
4/4 [==============================] - 0s 39ms/step - loss: 1.1246 - accuracy: 0.7450
Epoch 96/100
4/4 [==============================] - 0s 40ms/step - loss: 1.1332 - accuracy: 0.7350
Epoch 97/100
4/4 [==============================] - 0s 39ms/step - loss: 1.0974 - accuracy: 0.7550
Epoch 98/100
4/4 [==============================] - 0s 42ms/step - loss: 1.0765 - accuracy: 0.7650
Epoch 99/100
4/4 [==============================] - 0s 40ms/step - loss: 1.0744 - accuracy: 0.7450
Epoch 100/100
4/4 [==============================] - 0s 40ms/step - loss: 1.0666 - accuracy: 0.7350
```

و در نهایت این مدل را روی داده تست ارزیابی میکنیم:

```
313/313 [==============================] - 6s 18ms/step - loss: 2.7833 - accuracy: 0.1925
result is a little better. the accuracy on test is 19.249999523162842 %
```

که میبینیم به دقت تقریبا دو برابر رسیده ایم. و این نمونه ای از self-supervised learning بود که توانستیم دقت مدل را بدون استفاده از داده های برچسب دار بیشتری، دو برابر کنیم. با آموزش بیشتر هر کدام از این شبکه ها احتمالا به نتایج بهتری هم میرسیدیم. یا با زیاد کردن لایه های dense برای تصمیم گیری بهتر هم احتمالا همینطور میشد.

پ)

بازهم اولین مرحله ساختن دیتای مورد نیاز است. ابتدا تمام x_train و x_unlabeld را با هم concatenate کرده ایم و all_x_train را با shape=(50000,32,32,3) ساخته ایم. و همچنین y_train را هم مطابق صورت سوال با آرایه ای تماما صفر به اندازه داده های بدون برچسب concatenate کرده ایم و all_y_train

را با shape = (50000,1) ساخته ایم. سپس برای این که label دار بودن ۲۰۰ داده اول و بدون برچسب بودن بقیه باعث خطایی در مدل ها نشود، از تابع shuffle موجود در sklearn استفاده میکنیم و همه را به هم میریزیم.

حالا مشابه قسمت قبل که برای 49800 داده، به صورت رندوم چرخاندیم و برچسب میزان درجه چرخیدن را زدیم، این بار برای تمام ۵۰۰۰۰ داده میکنیم و در نتیجه y_train_rotate را با shape=(50000,1) و x_train_rotate با همان all_x_train shape ساخته میشوند. و دقیقا کار مشابه را برای ۱۰۰۰ داده تست هم انجام می دهیم.

بعد تمام برچسب ها را بازهم با to_categorical به فرم one hot در آورده ایم.

مدلی را این بار هم با موبایلنت ورژن دوم + یک لایه flatten + دو خروجی (مطابق لینک داده شده در سوال) که یکی لایه dense با ۱۰ نورون برای تعیین کلاس تصویر ورودی و یکی لایه dense با ۴ نورون برای تعیین میزان چرخش عکس است و هر دو طبیعتا تابع فعالسازی softmax دارند را طراحی میکنیم.

تابع run_with_loss_weights را تعریف میکنیم که شامل مراحل compile کردن مدل با ورودی های داده شده به این تابع به عنوان loss_weight هر کدام از لایه های خروجی، و fit کردن مدل روی داده های ساخته شده است. به این صورت بدون تغییر هایپرپارامترها با learning rate = 0.01 میتوانیم حالات مختلف را مقایسه کنیم. Epoch را برابر ۲۰ قرار داده ایم چون کمی اجرا زمانبر میباشد. نتایج به دست آمده را در ادامه میبینیم:

۱. با ضریب تابع ضرر برای کلاسبندی = ۱ و ضریب تابع ضرر برای چرخش = ۵

```
Epoch 1/20
391/391 [==============================] - 31s 64ms/step - loss: 3.4931 -
rotation_out_loss: 0.6918 - class_out_loss: 0.0339 - rotation_out_accuracy:
0.7216 - class_out_accuracy: 0.9964 - val_loss: 55.1765 -
val_rotation_out_loss: 5.4248 - val_class_out_loss: 28.0525 -
val_rotation_out_accuracy: 0.5641 - val_class_out_accuracy: 0.1000
Epoch 2/20
391/391 [==============================] - 24s 61ms/step - loss: 3.3660 -
rotation_out_loss: 0.6668 - class_out_loss: 0.0320 - rotation_out_accuracy:
0.7347 - class_out_accuracy: 0.9964 - val_loss: 42.4800 -
val_rotation_out_loss: 4.3437 - val_class_out_loss: 20.7616 -
val_rotation_out_accuracy: 0.5833 - val_class_out_accuracy: 0.1000
Epoch 3/20
391/391 [==============================] - 24s 61ms/step - loss: 3.2604 -
rotation_out_loss: 0.6457 - class_out_loss: 0.0317 - rotation_out_accuracy:
0.7456 - class_out_accuracy: 0.9964 - val_loss: 34.9468 -
val_rotation_out_loss: 3.4336 - val_class_out_loss: 17.7788 -
val_rotation_out_accuracy: 0.5929 - val_class_out_accuracy: 0.1000
Epoch 4/20
391/391 [==============================] - 24s 62ms/step - loss: 3.1627 -
rotation_out_loss: 0.6261 - class_out_loss: 0.0325 - rotation_out_accuracy:
0.7549 - class_out_accuracy: 0.9964 - val_loss: 31.9962 -
```

```
val_rotation_out_loss:  2.9546  -  val_class_out_loss:  17.2230  -
val_rotation_out_accuracy: 0.6164 - val_class_out_accuracy: 0.1000
Epoch 5/20
391/391 [==============================] - 24s 62ms/step - loss: 3.3627 -
rotation_out_loss: 0.6658 - class_out_loss: 0.0335 - rotation_out_accuracy:
0.7357  -  class_out_accuracy:  0.9964  -  val_loss:  46.4025  -
val_rotation_out_loss:  4.7449  -  val_class_out_loss:  22.6782  -
val_rotation_out_accuracy: 0.5820 - val_class_out_accuracy: 0.1000
Epoch 6/20
391/391 [==============================] - 24s 62ms/step - loss: 3.0350 -
rotation_out_loss: 0.6007 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.7629  -  class_out_accuracy:  0.9964  -  val_loss:  35.6949  -
val_rotation_out_loss:  3.7606  -  val_class_out_loss:  16.8917  -
val_rotation_out_accuracy: 0.5848 - val_class_out_accuracy: 0.1000
Epoch 7/20
391/391 [==============================] - 24s 62ms/step - loss: 2.9218 -
rotation_out_loss: 0.5781 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.7734  -  class_out_accuracy:  0.9964  -  val_loss:  27.5629  -
val_rotation_out_loss:  2.8269  -  val_class_out_loss:  13.4282  -
val_rotation_out_accuracy: 0.6319 - val_class_out_accuracy: 0.1000
Epoch 8/20
391/391 [==============================] - 24s 61ms/step - loss: 2.8379 -
rotation_out_loss: 0.5613 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.7816  -  class_out_accuracy:  0.9964  -  val_loss:  25.2894  -
val_rotation_out_loss:  2.7810  -  val_class_out_loss:  11.3842  -
val_rotation_out_accuracy: 0.5969 - val_class_out_accuracy: 0.1000
Epoch 9/20
391/391 [==============================] - 24s 61ms/step - loss: 2.7727 -
rotation_out_loss: 0.5483 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.7842  -  class_out_accuracy:  0.9964  -  val_loss:  24.8178  -
val_rotation_out_loss:  2.4628  -  val_class_out_loss:  12.5037  -
val_rotation_out_accuracy: 0.6408 - val_class_out_accuracy: 0.1000
Epoch 10/20
391/391 [==============================] - 24s 61ms/step - loss: 2.6976 -
rotation_out_loss: 0.5333 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.7919  -  class_out_accuracy:  0.9964  -  val_loss:  24.0171  -
val_rotation_out_loss:  2.0918  -  val_class_out_loss:  13.5580  -
val_rotation_out_accuracy: 0.6451 - val_class_out_accuracy: 0.1000
Epoch 11/20
391/391 [==============================] - 24s 61ms/step - loss: 2.6429 -
rotation_out_loss: 0.5222 - class_out_loss: 0.0318 - rotation_out_accuracy:
0.7985  -  class_out_accuracy:  0.9964  -  val_loss:  26.7297  -
val_rotation_out_loss:  2.2489  -  val_class_out_loss:  15.4854  -
val_rotation_out_accuracy: 0.6427 - val_class_out_accuracy: 0.1000
Epoch 12/20
391/391 [==============================] - 24s 61ms/step - loss: 2.5601 -
rotation_out_loss: 0.5057 - class_out_loss: 0.0317 - rotation_out_accuracy:
0.8062  -  class_out_accuracy:  0.9964  -  val_loss:  21.8920  -
val_rotation_out_loss:  2.1053  -  val_class_out_loss:  11.3657  -
val_rotation_out_accuracy: 0.6303 - val_class_out_accuracy: 0.1000
Epoch 13/20
391/391 [==============================] - 24s 63ms/step - loss: 2.5039 -
rotation_out_loss: 0.4944 - class_out_loss: 0.0321 - rotation_out_accuracy:
0.8086  -  class_out_accuracy:  0.9964  -  val_loss:  27.1587  -
val_rotation_out_loss:  2.5997  -  val_class_out_loss:  14.1599  -
val_rotation_out_accuracy: 0.6163 - val_class_out_accuracy: 0.1000
Epoch 14/20
391/391 [==============================] - 24s 61ms/step - loss: 2.4379 -
rotation_out_loss: 0.4813 - class_out_loss: 0.0312 - rotation_out_accuracy:
0.8152  -  class_out_accuracy:  0.9964  -  val_loss:  24.6113  -
```

```
val_rotation_out_loss:  2.5655  -  val_class_out_loss:  11.7836  -
val_rotation_out_accuracy: 0.6233 - val_class_out_accuracy: 0.1000
Epoch 15/20
391/391 [==============================] - 24s 61ms/step - loss: 2.3658 -
rotation_out_loss: 0.4667 - class_out_loss: 0.0321 - rotation_out_accuracy:
0.8206  -  class_out_accuracy:  0.9964  -  val_loss:  19.9112  -
val_rotation_out_loss:  1.9827  -  val_class_out_loss:  9.9975  -
val_rotation_out_accuracy: 0.6495 - val_class_out_accuracy: 0.0999
Epoch 16/20
391/391 [==============================] - 24s 61ms/step - loss: 2.3115 -
rotation_out_loss: 0.4559 - class_out_loss: 0.0320 - rotation_out_accuracy:
0.8248  -  class_out_accuracy:  0.9964  -  val_loss:  23.7548  -
val_rotation_out_loss:  2.3971  -  val_class_out_loss:  11.7693  -
val_rotation_out_accuracy: 0.6282 - val_class_out_accuracy: 0.1000
Epoch 17/20
391/391 [==============================] - 24s 61ms/step - loss: 2.2217 -
rotation_out_loss: 0.4380 - class_out_loss: 0.0320 - rotation_out_accuracy:
0.8333  -  class_out_accuracy:  0.9964  -  val_loss:  21.5289  -
val_rotation_out_loss:  2.0480  -  val_class_out_loss:  11.2890  -
val_rotation_out_accuracy: 0.6605 - val_class_out_accuracy: 0.1000
Epoch 18/20
391/391 [==============================] - 24s 63ms/step - loss: 2.1700 -
rotation_out_loss: 0.4276 - class_out_loss: 0.0317 - rotation_out_accuracy:
0.8359  -  class_out_accuracy:  0.9964  -  val_loss:  27.7916  -
val_rotation_out_loss:  2.5489  -  val_class_out_loss:  15.0473  -
val_rotation_out_accuracy: 0.6270 - val_class_out_accuracy: 0.1000
Epoch 19/20
391/391 [==============================] - 24s 61ms/step - loss: 2.1005 -
rotation_out_loss: 0.4136 - class_out_loss: 0.0327 - rotation_out_accuracy:
0.8444  -  class_out_accuracy:  0.9964  -  val_loss:  24.4459  -
val_rotation_out_loss:  2.0249  -  val_class_out_loss:  14.3212  -
val_rotation_out_accuracy: 0.6615 - val_class_out_accuracy: 0.1000
Epoch 20/20
391/391 [==============================] - 24s 61ms/step - loss: 2.0384 -
rotation_out_loss: 0.4014 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.8487  -  class_out_accuracy:  0.9964  -  val_loss:  26.2456  -
val_rotation_out_loss:  3.1002  -  val_class_out_loss:  10.7449  -
val_rotation_out_accuracy: 0.5758 - val_class_out_accuracy: 0.1000
```

حالا سعی میکنیم حالتی که ضرایب تقریبا برعکس هستند را امتحان کنیم.

۲. با ضریب تابع ضرر برای کلاسبندی = ۴ و ضریب تابع ضرر برای چرخش = ۱

```
Epoch 1/20
391/391 [==============================] - 32s 66ms/step - loss: 1.7033 -
rotation_out_loss: 1.4693 - class_out_loss: 0.0585 - rotation_out_accuracy:
0.3183  -  class_out_accuracy:  0.9939  -  val_loss:  19.8074  -
val_rotation_out_loss:  1.3877  -  val_class_out_loss:  4.6049  -
val_rotation_out_accuracy: 0.2513 - val_class_out_accuracy: 0.1000
Epoch 2/20
391/391 [==============================] - 24s 61ms/step - loss: 1.6808 -
rotation_out_loss: 1.4965 - class_out_loss: 0.0461 - rotation_out_accuracy:
0.2822  -  class_out_accuracy:  0.9964  -  val_loss:  28.0089  -
val_rotation_out_loss:  1.3936  -  val_class_out_loss:  6.6538  -
val_rotation_out_accuracy: 0.2502 - val_class_out_accuracy: 0.1000
```

```
Epoch 3/20
391/391 [==============================] - 24s 63ms/step - loss: 1.6478 -
rotation_out_loss: 1.4698 - class_out_loss: 0.0445 - rotation_out_accuracy:
0.2586 - class_out_accuracy: 0.9964 - val_loss: 27.4198 -
val_rotation_out_loss: 1.3922 - val_class_out_loss: 6.5069 -
val_rotation_out_accuracy: 0.2502 - val_class_out_accuracy: 0.1000
Epoch 4/20
391/391 [==============================] - 24s 61ms/step - loss: 1.5538 -
rotation_out_loss: 1.4198 - class_out_loss: 0.0335 - rotation_out_accuracy:
0.2687 - class_out_accuracy: 0.9964 - val_loss: 34.7055 -
val_rotation_out_loss: 1.3907 - val_class_out_loss: 8.3287 -
val_rotation_out_accuracy: 0.2505 - val_class_out_accuracy: 0.1000
Epoch 5/20
391/391 [==============================] - 24s 61ms/step - loss: 1.5625 -
rotation_out_loss: 1.4074 - class_out_loss: 0.0388 - rotation_out_accuracy:
0.2629 - class_out_accuracy: 0.9963 - val_loss: 27.1328 -
val_rotation_out_loss: 1.3887 - val_class_out_loss: 6.4360 -
val_rotation_out_accuracy: 0.2502 - val_class_out_accuracy: 0.1000
Epoch 6/20
391/391 [==============================] - 24s 63ms/step - loss: 1.5124 -
rotation_out_loss: 1.3792 - class_out_loss: 0.0333 - rotation_out_accuracy:
0.2848 - class_out_accuracy: 0.9964 - val_loss: 27.6511 -
val_rotation_out_loss: 1.4279 - val_class_out_loss: 6.5558 -
val_rotation_out_accuracy: 0.2513 - val_class_out_accuracy: 0.1000
Epoch 7/20
391/391 [==============================] - 24s 63ms/step - loss: 1.4169 -
rotation_out_loss: 1.2820 - class_out_loss: 0.0337 - rotation_out_accuracy:
0.3853 - class_out_accuracy: 0.9964 - val_loss: 31.0299 -
val_rotation_out_loss: 1.3602 - val_class_out_loss: 7.4174 -
val_rotation_out_accuracy: 0.3535 - val_class_out_accuracy: 0.1000
Epoch 8/20
391/391 [==============================] - 24s 61ms/step - loss: 1.2973 -
rotation_out_loss: 1.1645 - class_out_loss: 0.0332 - rotation_out_accuracy:
0.4643 - class_out_accuracy: 0.9964 - val_loss: 32.6977 -
val_rotation_out_loss: 1.5678 - val_class_out_loss: 7.7825 -
val_rotation_out_accuracy: 0.4050 - val_class_out_accuracy: 0.1000
Epoch 9/20
391/391 [==============================] - 24s 63ms/step - loss: 1.2531 -
rotation_out_loss: 1.1207 - class_out_loss: 0.0331 - rotation_out_accuracy:
0.4987 - class_out_accuracy: 0.9964 - val_loss: 33.3295 -
val_rotation_out_loss: 1.6622 - val_class_out_loss: 7.9168 -
val_rotation_out_accuracy: 0.4269 - val_class_out_accuracy: 0.1000
Epoch 10/20
391/391 [==============================] - 24s 63ms/step - loss: 1.2156 -
rotation_out_loss: 1.0845 - class_out_loss: 0.0328 - rotation_out_accuracy:
0.5208 - class_out_accuracy: 0.9963 - val_loss: 30.2547 -
val_rotation_out_loss: 1.5674 - val_class_out_loss: 7.1718 -
val_rotation_out_accuracy: 0.4878 - val_class_out_accuracy: 0.1000
Epoch 11/20
391/391 [==============================] - 25s 63ms/step - loss: 1.1841 -
rotation_out_loss: 1.0546 - class_out_loss: 0.0324 - rotation_out_accuracy:
0.5433 - class_out_accuracy: 0.9964 - val_loss: 38.3265 -
val_rotation_out_loss: 1.6973 - val_class_out_loss: 9.1573 -
val_rotation_out_accuracy: 0.4816 - val_class_out_accuracy: 0.1000
Epoch 12/20
391/391 [==============================] - 24s 61ms/step - loss: 1.1469 -
rotation_out_loss: 1.0193 - class_out_loss: 0.0319 - rotation_out_accuracy:
0.5687 - class_out_accuracy: 0.9964 - val_loss: 36.6903 -
val_rotation_out_loss: 2.0080 - val_class_out_loss: 8.6706 -
val_rotation_out_accuracy: 0.5358 - val_class_out_accuracy: 0.1000
```

```
Epoch 13/20
391/391 [==============================] - 24s 61ms/step - loss: 1.1181 -
rotation_out_loss: 0.9892 - class_out_loss: 0.0322 - rotation_out_accuracy:
0.5888 - class_out_accuracy: 0.9964 - val_loss: 38.7685 -
val_rotation_out_loss: 2.3428 - val_class_out_loss: 9.1064 -
val_rotation_out_accuracy: 0.4986 - val_class_out_accuracy: 0.1000
Epoch 14/20
391/391 [==============================] - 24s 61ms/step - loss: 1.0944 -
rotation_out_loss: 0.9656 - class_out_loss: 0.0322 - rotation_out_accuracy:
0.6005 - class_out_accuracy: 0.9964 - val_loss: 38.8946 -
val_rotation_out_loss: 2.5587 - val_class_out_loss: 9.0840 -
val_rotation_out_accuracy: 0.5059 - val_class_out_accuracy: 0.1000
Epoch 15/20
391/391 [==============================] - 25s 63ms/step - loss: 1.0721 -
rotation_out_loss: 0.9442 - class_out_loss: 0.0320 - rotation_out_accuracy:
0.6103 - class_out_accuracy: 0.9964 - val_loss: 43.5239 -
val_rotation_out_loss: 2.9941 - val_class_out_loss: 10.1324 -
val_rotation_out_accuracy: 0.5312 - val_class_out_accuracy: 0.1000
Epoch 16/20
391/391 [==============================] - 24s 61ms/step - loss: 1.0845 -
rotation_out_loss: 0.9557 - class_out_loss: 0.0322 - rotation_out_accuracy:
0.6033 - class_out_accuracy: 0.9964 - val_loss: 55.8122 -
val_rotation_out_loss: 4.7401 - val_class_out_loss: 12.7680 -
val_rotation_out_accuracy: 0.4664 - val_class_out_accuracy: 0.1000
Epoch 17/20
391/391 [==============================] - 24s 61ms/step - loss: 1.0449 -
rotation_out_loss: 0.9153 - class_out_loss: 0.0324 - rotation_out_accuracy:
0.6245 - class_out_accuracy: 0.9964 - val_loss: 42.5208 -
val_rotation_out_loss: 3.2226 - val_class_out_loss: 9.8246 -
val_rotation_out_accuracy: 0.5368 - val_class_out_accuracy: 0.1000
Epoch 18/20
391/391 [==============================] - 24s 63ms/step - loss: 1.0228 -
rotation_out_loss: 0.8957 - class_out_loss: 0.0318 - rotation_out_accuracy:
0.6330 - class_out_accuracy: 0.9964 - val_loss: 43.4334 -
val_rotation_out_loss: 3.0589 - val_class_out_loss: 10.0936 -
val_rotation_out_accuracy: 0.5364 - val_class_out_accuracy: 0.1000
Epoch 19/20
391/391 [==============================] - 24s 63ms/step - loss: 1.0118 -
rotation_out_loss: 0.8851 - class_out_loss: 0.0317 - rotation_out_accuracy:
0.6370 - class_out_accuracy: 0.9964 - val_loss: 42.0423 -
val_rotation_out_loss: 3.2382 - val_class_out_loss: 9.7010 -
val_rotation_out_accuracy: 0.5318 - val_class_out_accuracy: 0.1000
Epoch 20/20
391/391 [==============================] - 24s 61ms/step - loss: 1.0403 -
rotation_out_loss: 0.9139 - class_out_loss: 0.0316 - rotation_out_accuracy:
0.6266 - class_out_accuracy: 0.9964 - val_loss: 52.8178 -
val_rotation_out_loss: 4.2122 - val_class_out_loss: 12.1514 -
val_rotation_out_accuracy: 0.3913 - val_class_out_accuracy: 0.1000
```

و حالا حالتی که تابع ضرر برای هر دو خروجی اثر برابری داشته باشد را میبینیم.

۳. با ضریب تابع ضرر برای کلاسبندی = ۳ و ضریب تابع ضرر برای چرخش = ۵

```
Epoch 1/20
391/391 [==============================] - 31s 64ms/step - loss: 1.6153 -
rotation_out_loss: 1.5512 - class_out_loss: 0.0641 - rotation_out_accuracy:
0.2850 - class_out_accuracy: 0.9947 - val_loss: 5.9792 -
```

```
val_rotation_out_loss:     1.4043     -     val_class_out_loss:     4.5749     -
val_rotation_out_accuracy: 0.2503 - val_class_out_accuracy: 0.1000
Epoch 2/20
391/391  [==============================] -  24s  60ms/step  -  loss:  1.3666  -
rotation_out_loss:  1.3302  -  class_out_loss:  0.0364  -  rotation_out_accuracy:
0.3552     -     class_out_accuracy:     0.9964     -     val_loss:     7.2027     -
val_rotation_out_loss:     1.4014     -     val_class_out_loss:     5.8012     -
val_rotation_out_accuracy: 0.2505 - val_class_out_accuracy: 0.1000
Epoch 3/20
391/391  [==============================] -  24s  61ms/step  -  loss:  1.1660  -
rotation_out_loss:  1.1299  -  class_out_loss:  0.0361  -  rotation_out_accuracy:
0.4969     -     class_out_accuracy:     0.9963     -     val_loss:     8.4604     -
val_rotation_out_loss:     1.3975     -     val_class_out_loss:     7.0629     -
val_rotation_out_accuracy: 0.2480 - val_class_out_accuracy: 0.1000
Epoch 4/20
391/391  [==============================] -  24s  61ms/step  -  loss:  1.0879  -
rotation_out_loss:  1.0536  -  class_out_loss:  0.0343  -  rotation_out_accuracy:
0.5502     -     class_out_accuracy:     0.9964     -     val_loss:     8.9821     -
val_rotation_out_loss:     1.4057     -     val_class_out_loss:     7.5763     -
val_rotation_out_accuracy: 0.2549 - val_class_out_accuracy: 0.1000
Epoch 5/20
391/391  [==============================] -  24s  61ms/step  -  loss:  1.0333  -
rotation_out_loss:  1.0002  -  class_out_loss:  0.0331  -  rotation_out_accuracy:
0.5794     -     class_out_accuracy:     0.9964     -     val_loss:     9.1979     -
val_rotation_out_loss:     1.5023     -     val_class_out_loss:     7.6957     -
val_rotation_out_accuracy: 0.3038 - val_class_out_accuracy: 0.1000
Epoch 6/20
391/391  [==============================] -  24s  62ms/step  -  loss:  1.0066  -
rotation_out_loss:  0.9732  -  class_out_loss:  0.0334  -  rotation_out_accuracy:
0.5969     -     class_out_accuracy:     0.9964     -     val_loss:    10.0021     -
val_rotation_out_loss:     1.4671     -     val_class_out_loss:     8.5350     -
val_rotation_out_accuracy: 0.3446 - val_class_out_accuracy: 0.1000
Epoch 7/20
391/391  [==============================] -  24s  61ms/step  -  loss:  0.9778  -
rotation_out_loss:  0.9452  -  class_out_loss:  0.0326  -  rotation_out_accuracy:
0.6085     -     class_out_accuracy:     0.9964     -     val_loss:     8.5970     -
val_rotation_out_loss:     1.1943     -     val_class_out_loss:     7.4027     -
val_rotation_out_accuracy: 0.4755 - val_class_out_accuracy: 0.1000
Epoch 8/20
391/391  [==============================] -  24s  61ms/step  -  loss:  0.9540  -
rotation_out_loss:  0.9212  -  class_out_loss:  0.0327  -  rotation_out_accuracy:
0.6210     -     class_out_accuracy:     0.9964     -     val_loss:     8.9247     -
val_rotation_out_loss:     1.0447     -     val_class_out_loss:     7.8800     -
val_rotation_out_accuracy: 0.5645 - val_class_out_accuracy: 0.1000
Epoch 9/20
391/391  [==============================] -  24s  61ms/step  -  loss:  0.9281  -
rotation_out_loss:  0.8955  -  class_out_loss:  0.0325  -  rotation_out_accuracy:
0.6301     -     class_out_accuracy:     0.9964     -     val_loss:     9.7632     -
val_rotation_out_loss:     1.4080     -     val_class_out_loss:     8.3553     -
val_rotation_out_accuracy: 0.5744 - val_class_out_accuracy: 0.1000
Epoch 10/20
391/391  [==============================] -  24s  61ms/step  -  loss:  0.9503  -
rotation_out_loss:  0.9175  -  class_out_loss:  0.0327  -  rotation_out_accuracy:
0.6228     -     class_out_accuracy:     0.9964     -     val_loss:    11.6691     -
val_rotation_out_loss:     1.9778     -     val_class_out_loss:     9.6914     -
val_rotation_out_accuracy: 0.5653 - val_class_out_accuracy: 0.1000
Epoch 11/20
391/391  [==============================] -  24s  61ms/step  -  loss:  0.9061  -
rotation_out_loss:  0.8739  -  class_out_loss:  0.0323  -  rotation_out_accuracy:
0.6452     -     class_out_accuracy:     0.9964     -     val_loss:    15.3703     -
```

```
val_rotation_out_loss:  2.8113  -  val_class_out_loss:  12.5590  -
val_rotation_out_accuracy: 0.5308 - val_class_out_accuracy: 0.1000
Epoch 12/20
391/391 [==============================] - 24s 62ms/step - loss: 0.8781 -
rotation_out_loss: 0.8461 - class_out_loss: 0.0319 - rotation_out_accuracy:
0.6543  -  class_out_accuracy:  0.9964  -  val_loss:  18.0757  -
val_rotation_out_loss:  4.4198  -  val_class_out_loss:  13.6559  -
val_rotation_out_accuracy: 0.5100 - val_class_out_accuracy: 0.1000
Epoch 13/20
391/391 [==============================] - 24s 62ms/step - loss: 0.8557 -
rotation_out_loss: 0.8236 - class_out_loss: 0.0321 - rotation_out_accuracy:
0.6650  -  class_out_accuracy:  0.9964  -  val_loss:  15.0199  -
val_rotation_out_loss:  3.4534  -  val_class_out_loss:  11.5665  -
val_rotation_out_accuracy: 0.5475 - val_class_out_accuracy: 0.1000
Epoch 14/20
391/391 [==============================] - 24s 61ms/step - loss: 0.8406 -
rotation_out_loss: 0.8087 - class_out_loss: 0.0320 - rotation_out_accuracy:
0.6717  -  class_out_accuracy:  0.9964  -  val_loss:  14.9141  -
val_rotation_out_loss:  3.4494  -  val_class_out_loss:  11.4648  -
val_rotation_out_accuracy: 0.5685 - val_class_out_accuracy: 0.1000
Epoch 15/20
391/391 [==============================] - 24s 61ms/step - loss: 0.8230 -
rotation_out_loss: 0.7913 - class_out_loss: 0.0317 - rotation_out_accuracy:
0.6803  -  class_out_accuracy:  0.9964  -  val_loss:  15.1207  -
val_rotation_out_loss:  3.5073  -  val_class_out_loss:  11.6134  -
val_rotation_out_accuracy: 0.5884 - val_class_out_accuracy: 0.1000
Epoch 16/20
391/391 [==============================] - 24s 61ms/step - loss: 0.8028 -
rotation_out_loss: 0.7712 - class_out_loss: 0.0317 - rotation_out_accuracy:
0.6879  -  class_out_accuracy:  0.9964  -  val_loss:  14.1760  -
val_rotation_out_loss:  3.9012  -  val_class_out_loss:  10.2747  -
val_rotation_out_accuracy: 0.5510 - val_class_out_accuracy: 0.1000
Epoch 17/20
391/391 [==============================] - 24s 61ms/step - loss: 0.7885 -
rotation_out_loss: 0.7572 - class_out_loss: 0.0314 - rotation_out_accuracy:
0.6930  -  class_out_accuracy:  0.9964  -  val_loss:  15.9007  -
val_rotation_out_loss:  3.7701  -  val_class_out_loss:  12.1306  -
val_rotation_out_accuracy: 0.5661 - val_class_out_accuracy: 0.1000
Epoch 18/20
391/391 [==============================] - 24s 61ms/step - loss: 0.7682 -
rotation_out_loss: 0.7370 - class_out_loss: 0.0312 - rotation_out_accuracy:
0.7009  -  class_out_accuracy:  0.9964  -  val_loss:  14.4397  -
val_rotation_out_loss:  3.2333  -  val_class_out_loss:  11.2063  -
val_rotation_out_accuracy: 0.5798 - val_class_out_accuracy: 0.1000
Epoch 19/20
391/391 [==============================] - 24s 61ms/step - loss: 0.7613 -
rotation_out_loss: 0.7298 - class_out_loss: 0.0315 - rotation_out_accuracy:
0.7075  -  class_out_accuracy:  0.9964  -  val_loss:  15.1596  -
val_rotation_out_loss:  3.3943  -  val_class_out_loss:  11.7653  -
val_rotation_out_accuracy: 0.5738 - val_class_out_accuracy: 0.1000
Epoch 20/20
391/391 [==============================] - 24s 63ms/step - loss: 0.7415 -
rotation_out_loss: 0.7100 - class_out_loss: 0.0315 - rotation_out_accuracy:
0.7157  -  class_out_accuracy:  0.9964  -  val_loss:  26.4898  -
val_rotation_out_loss:  6.0250  -  val_class_out_loss:  20.4647  -
val_rotation_out_accuracy: 0.5030 - val_class_out_accuracy: 0.1000
```

که با مقایسه این سه حالت میبینیم که کلاسبندی در زمان آموزش در هر سه مدل به راحتی به دقت بالا رسیده است و در زمان تست هم در هر سه پایین است. ولی برای rotation که آن ها را مقایسه میکنیم میبینیم که در زمان آموزش، عملکرد حالت اول بهتر بوده و در زمان ارزیابی (که مهم است) حالت اول نتیجه بهتری هم روی دقت و هم روی کاهش loss در روند آموزش مدل داشته است.

بهتر بودن عملکرد مدل در حالت اول، قابل توجیه است. چون ضریب loss را در آن حالت برای نتیجه چرخش بیشتر گرفته ایم. و میدانیم که تمام ۵۰۰۰۰ داده ما به همراه میزان چرخش آن ها داده هایی واقعی است و اگر گرادیان بیشتر در جهت بهینه کردن آن ها حرکت کند بهتر به جواب بهینه میرسد. درحالی که ۴۹۸۰۰ داده کلاسشان ۰ در نظر گرفته شده است و فقط ۲۰۰ داده واقعی داریم. در نتیجه ضریب بیشتر دادن به آن خروجی یا حتی ضریب برابر دادن، باعث نتایج نامناسبی میشود.