به نام خدا

نام: فاطمه زهرا بخشنده **استاد: دکتر مرضیه داود آبادی**

شماره دانشجویی: 98522157 **درس مبانی یادگیری عمیق**

گزارش تمرین 6:

سوال اول:

کد این سوال در فایل Q1.ipynp موجود است.

از فرمول های زیر استفاده می کنیم:

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2$$

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

$$y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j$$

برای batch normalization میانگین و انحراف معیار در هر ستون (هر کانال) را محاسبه می کنیم. نتیجه:

```
batch normalization aoutput:

 [[-0.25839036 -1.95226042 -1.32416942 -0.17220347  1.24551178]
 [-0.8613012   0.36725691  1.32416942  1.40632837 -0.92919133]
 [ 1.89486265  0.27061036  0.41380294 -0.74621505  1.19608671]
 [ 0.          0.89881297 -0.99312707 -1.32022663 -0.68206597]
 [-0.77517108  0.41558019  0.57932412  0.83231679 -0.83034119]]
```

برای layer normalization هر تصویر را جداگانه نرمال سازی می کنیم. پس میانگین و انحراف معیار در هر سطر را محاسبه می کنیم. نتیجه:

```
layer normalization aoutput:

 [[-0.87558998 -1.07958858 -1.87064621  0.11541284  0.77748801]
 [-1.27882221  0.5099897   2.95682788  0.69247703 -1.02301054]
 [ 0.56452512  0.44375727  1.29738366 -0.09442869  0.73656759]
 [-0.70277617  0.87426805 -1.26721195 -0.30427021 -0.81840843]
 [-1.22121761  0.54310591  1.59910079  0.48263551 -0.9411697 ]]
```

سوال دوم:

برت نوعی مدل زبانی unsupervised برای حل چالش‌های به‌روز در زمینهٔ NLP است. نوآوری کلیدی برت که آن را متمایز می‌سازد پیاده‌سازی آموزش دوطرفه روی معماری ترنسفورمرها برای مدل‌سازی زبان‌هاست. تلاش‌های قبل از این دنبالهٔ متون را یا از چپ به راست یا از راست به چپ بررسی می‌کردند. نتایج مقالهٔ برت نشان می‌دهد که رویکرد دوطرفه باعث می‌شود مدل طراحی‌شده ازنظر فهم context و flow کلمات، در مقایسه با مدل‌های یک‌طرفه، زبان را عمیق‌تر درک کند. در این مقاله، تکنیکی با عنوان MLMیا Masked Language Model معرفی می‌شود که آموزش دوطرفه را ممکن می‌سازد.

برت از مکانیزم توجه ترنسفورمرها استفاده می‌کند که رابطهٔ بین کلمات را در زمینه‌های مختلف یاد می‌گیرد. در ساده‌ترین شکل خود، ترنسفورمر شامل دو مکانیزم جداست: یک Encoder که متن ورودی را می‌خواند و یک Decoder که پیش‌بینی را برای تسک مشخص‌شده بیان می‌کند. از آنجا که هدف برت ساختن نوعی مدل زبانی است که متون را می‌فهمد، تنها استفاده از لایه‌های Encoder ضروری است.

برعکس مدل‌های قبلی (RNNs و LSTM) که متن ورودی را به‌ترتیب از چپ به راست یا از راست به چپ می‌خواند، لایهٔ کدگذار ترنسفورمرها دنباله‌ای از کلمات ورودی را به‌صورت یکجا می‌خواند. این خصوصیت باعث می‌شود که مدل مدنظر زمینهٔ (context) یک کلمه را بر اساس کلمه‌های نزدیکش (چپ یا راست) یاد بگیرد.

مدل هایی مانند word2vec و GloVe، word embedding را بدون در نظر گرفتن context ای که کلمات در آن ظاهر می شوند، یاد می گیرند. و این یک محدودیت است زیرا بسیاری از کلمات بسته به context استفاده از آنها معانی مختلفی را بیان می کنند. به عنوان مثال، کلماتی مانند" bank" ممکن است در یک context مالی مانند bank account ظاهر شوند یا ممکن است برای توصیف ساحل استفاده شوند. BERT بازنمایی ها را بر اساس context ای که کلمات در آن ظاهر می شوند، یاد می گیرد. در نتیجه، BERT می تواند بازنمایی های معنایی غنی تری را بیاموزد که معانی مختلفی از کلمات را بسته به context آن ها یاد می گیرد.

BERT با حل نوع خاصی از تسک self supervised که نیازی به داده های لیبل دار ندارد، بهینه می شود. یعنی در طول آموزش درصدی از token های انتخاب شده به‌طور تصادفی از جمله ورودی قبل از عبور از Transformer encoder پوشانده می‌شوند. encoder جمله ورودی را به یک سری از بردارهای embed شده نگاشت می کند. این بردارها از یک لایه softmax عبور می‌کنند که احتمالات را در کل واژگان محاسبه می‌کند تا محتمل‌ترین کلمات شانس بیشتری برای انتخاب داشته باشند.

مهم ترین ویژگی در مدل BERT نیز این است که می توان با توجه به تسک دلخواه خود آن را tune-fine کرد.

سوال سوم:

ابتدا preprocess های مورد نیاز را روی داده هاو لیبل ها انجام می دهیم. توابع مورد نیاز را پیاده سازی می کنیم. از تابع plot_result برای کشیدن نمودار های loss و accuracy پس از هر train، استفاده می‌کنیم. از تابع build_back_model برای ساختن مدل base خود استفاده می کنیم که در مدل های مختلفی که میسازیم، مشترک است. تابع block را برای اضافه کردن یک block به مدل خود استفاده می‌کنیم، تا کد یک بلاک کانولوشنی را چند دفعه تکرار نکنیم. یک بلاک، شامل یک لایه Conv2D، یک لایه Batch Normalization، یک لایه Max Pooling و در ادامه یک لایه Dropout می باشد. در مدل پایه خود، از سه بلاک استفاده می کنیم. مدل پایه به صورت زیر است:

```
Model: "back_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d (Conv2D)             (None, 30, 30, 32)        896

 batch_normalization (BatchN  (None, 30, 30, 32)       128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 10, 10, 32)       0
 )

 dropout (Dropout)           (None, 10, 10, 32)        0

 conv2d_1 (Conv2D)           (None, 8, 8, 64)          18496

 batch_normalization_1 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 64)          0

 conv2d_2 (Conv2D)           (None, 2, 2, 128)         73856

 batch_normalization_2 (Batc  (None, 2, 2, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 1, 1, 128)         0

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 1024)              132096

 batch_normalization_3 (Batc  (None, 1024)             4096
 hNormalization)

 dropout_3 (Dropout)         (None, 1024)              0

=================================================================
Total params: 230,336
Trainable params: 227,840
Non-trainable params: 2,496
```

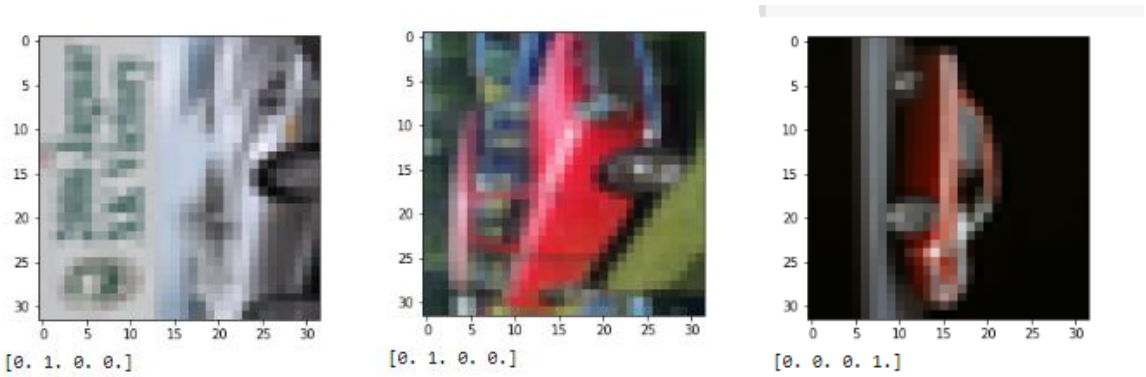
از تابع build_model برای ساختن مدل نهایی برای هر بخش استفاده می کنیم. این تابع ابتدا یک مدل شامل مدل پایه می سازد، و سپس، لایه Dense آخر را با تعداد units که ورودی می گیرد، قرار می دهد. و مدل را با بهینه ساز Adam و تابع ضرر CCE کامپایل می کند.

الف) در این بخش با استفاده از توابعی که توضیح داده شد، مدل را با 10 نورون خروجی میسازیم. مدل را روی داده های آموزشی با batch size برابر 32 و epochs=50 آموزش می دهیم. از همان داده های تست برای validation استفاده می کنیم. نتیجه:

```
- loss: 0.2945 - accuracy: 0.9100 - val_loss: 5.4223 - val_accuracy: 0.1340
```

شبکه Overfit شده است. تنها 200 داده Label دار داریم که شبکه آن را را حفظ کرده است و دقت آن روی داده train به 91 درصد می رسد اما داده validation شامل 49800 تصویر است و دقت شبکه در آن، نزدیک به حالت رندوم شده است (10 کلاس داریم، اگر شبکه کاملا رندوم پیش بینی کند دقت 10 درصد خواهد شد.)

ب) ابتدا تصاویر اصلی را در 4 زاویه به صورت شانسی Rotate می‌کنیم. و لیبل ها را نیز با to_categorical می سازیم. 4 خروجی به صورت one hot خواهیم داشت.



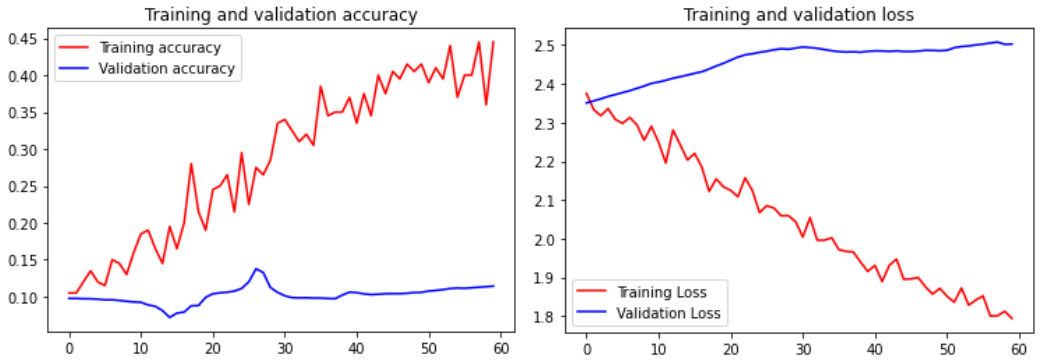
نمونه 3 دیتا را مشاهده می کنیم که دیتای اول 270 درجه چرخیده پس جزو کلاس 4 ام است. دو تصویر بعدی 90 درجه چرخیده اند پس عضو کلاس 2 هستند.

سپس با استفاده از توابعی که توضیح داده شد، مدل را با 4 نورون خروجی میسازیم. مدل را روی داده های آموزشی جدید با batch size برابر 32 و epochs=40 آموزش می دهیم. از 0.2 داده ها هم برای validation استفاده می کنیم. نتیجه:

```
loss: 0.3997 - accuracy: 0.8372 - val_loss: 0.4816 - val_accuracy: 0.8023
```

حالا از وزن های بدست آمده در تسک Self-Supervised برای تسک اصلی خود استفاده می کنیم. برای انجام اینکار مدل از پیش‌آموخته را بدون لایه آخر برمی‌داریم و با استفاده از تابع build_model به انتهای آن یک لایه Dense با 10 نورون اضافه می کنیم. learning rate را کم می کنیم و برابر 0.00005 قرار می‌دهیم. نتیجه:

```
loss: 1.9694 - accuracy: 0.3000 - val_loss: 2.2676 - val_accuracy: 0.1703
```



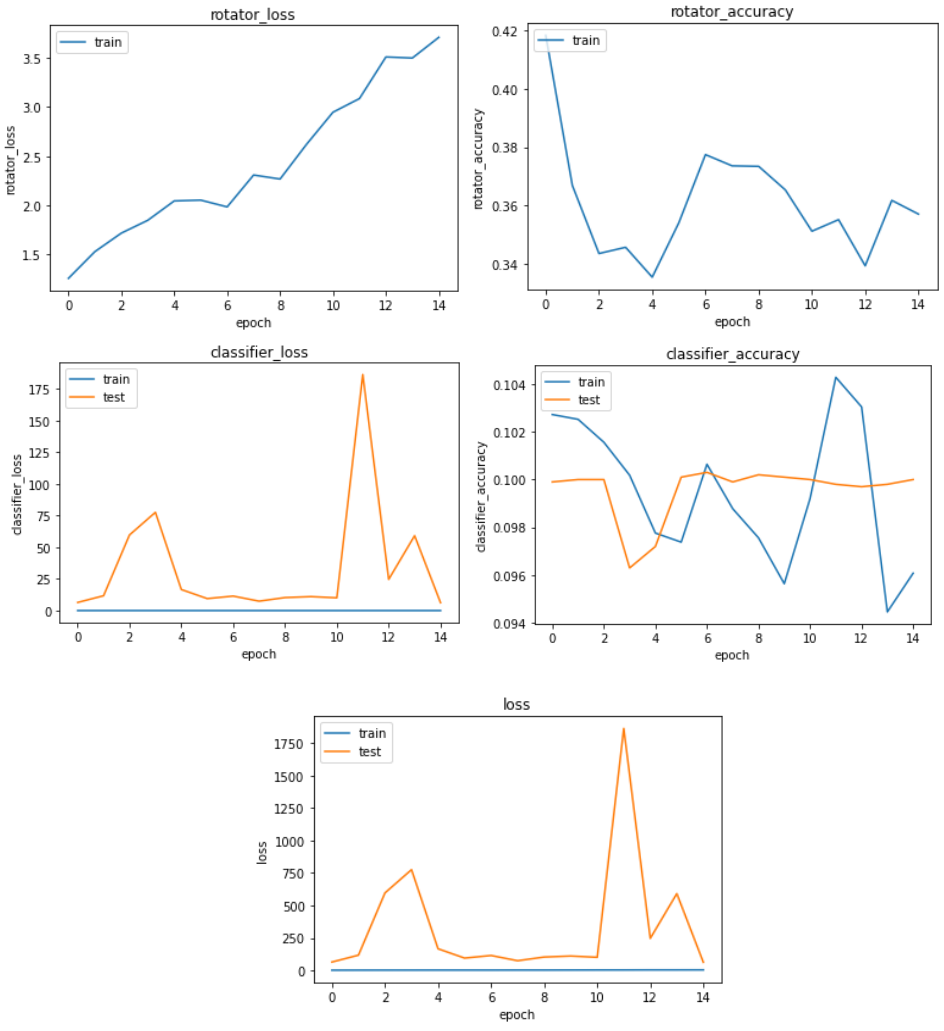
با استفاده از self supervised learning توانستیم دقت validation را افزایش دهیم. اگر مدت آموزش را افزایش می دادیم، می‌توانستیم به دقت بالاتری نیز برسیم اما چون learning rate را مقدار کمی انتخاب کردیم این فرایند بیشتر طول می کشد.

پ) در این بخش تابع build_two_out_model را پیاده سازی می کنیم که ابتدا مدل پایه را با ستفاده از build_back_model می سازد، و دو خروجی classifier و rotator به آن اضافه می کند.

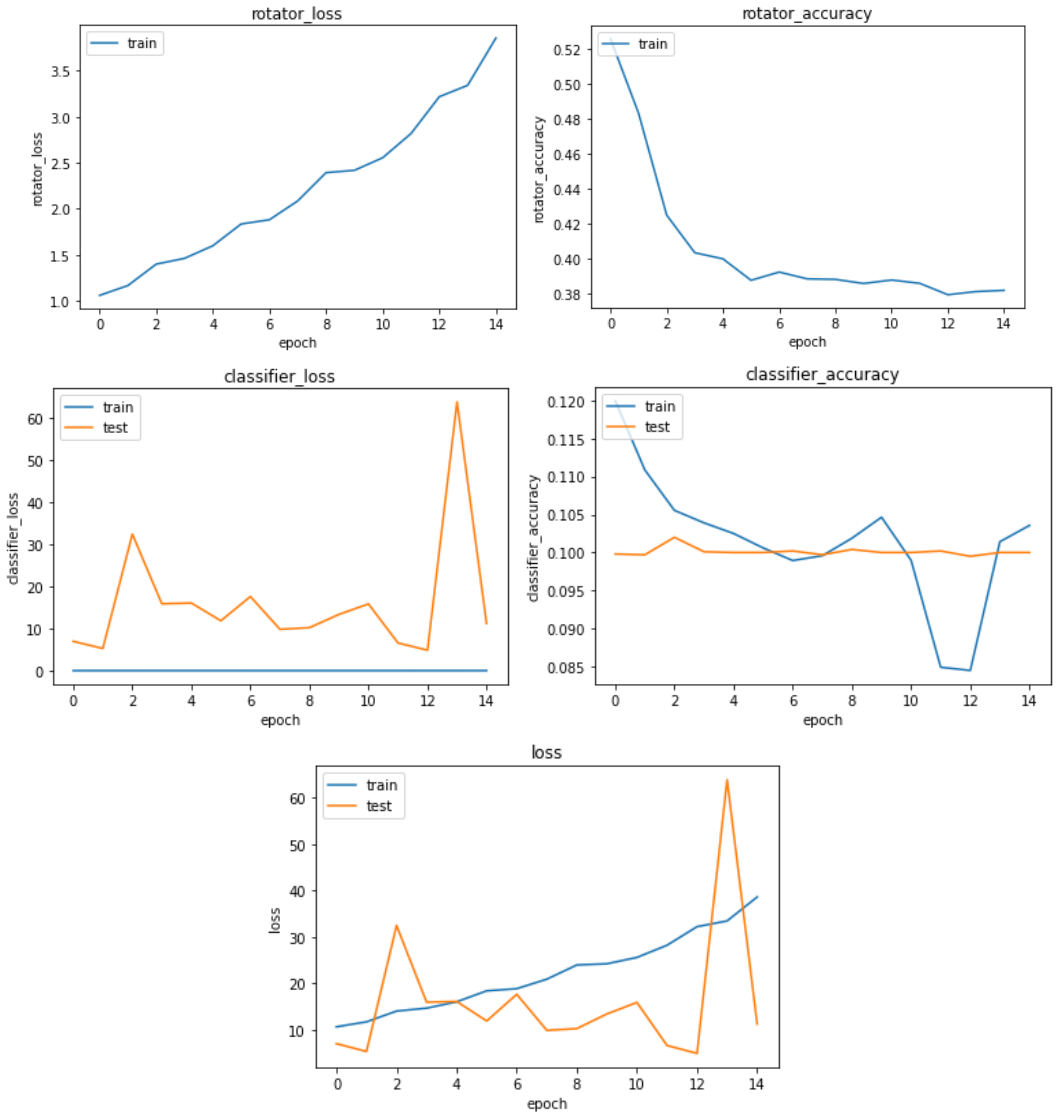
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 back_model_input (InputLayer)  [(None, 32, 32, 3)]  0           []

 back_model (Functional)        (None, 1024)         230336      ['back_model_input[0][0]']

 classifier (Dense)             (None, 10)           10250       ['back_model[0][0]']

 rotator (Dense)                (None, 4)            4100        ['back_model[0][0]']

==================================================================================================
Total params: 244,686
Trainable params: 242,190
Non-trainable params: 2,496
```

دیتاست خود را طوری تغییر می‌دهیم که دارای یک ورودی و دو خروجی باشد.

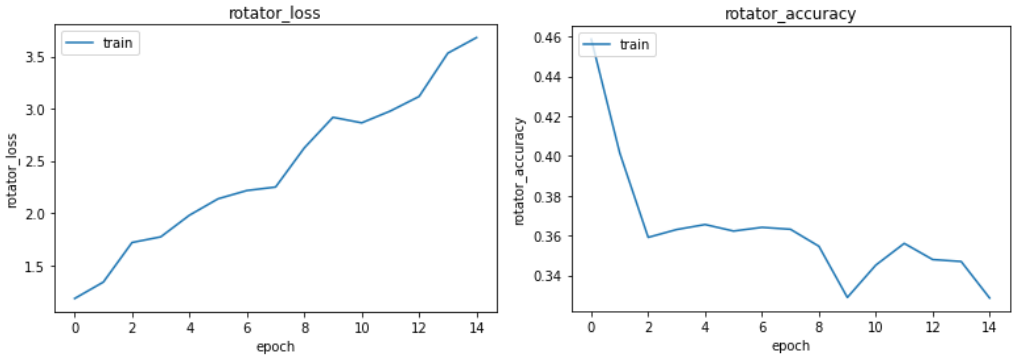
1- وزن loss خروجی classifier را 10 برابر rotator در نظر می گیریم. نتیجه:

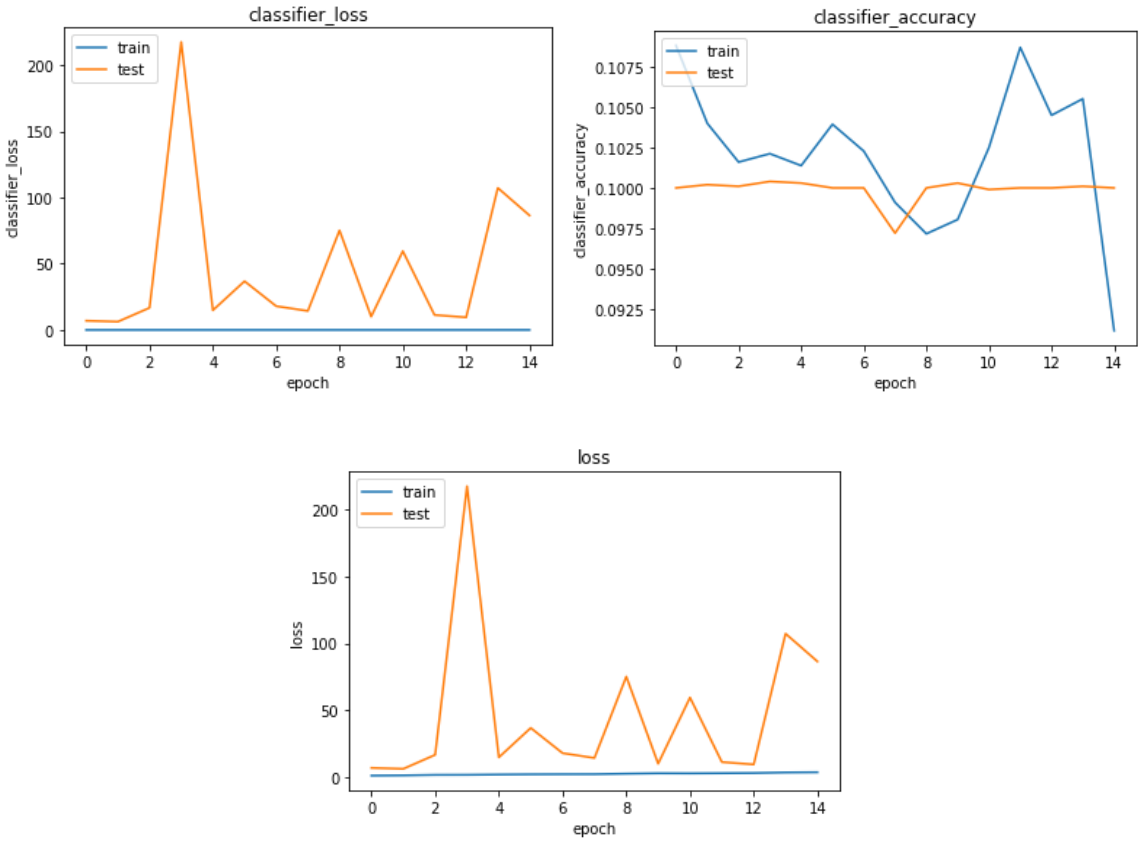


2- وزن loss خروجی rotator را 10 برابر classifier در نظر می گیریم. نتیجه:



3- وزن loss خروجی classifier را برابر rotator در نظر می گیریم. نتیجه:





مقایسه performance برای تسک rotation: حالت دوم > حالت سوم > حالت اول

مقایسه performance برای تسک classification: حالت اول > حالت سوم > حالت دوم

مقایسه performance در حالت کلی: حالت سوم > حالت دوم > حالت اول

همانطور که دیدیم، وزن‌هایی که بر روی loss هر کدام از تسک ها در نظر می‌گیریم بر روی عملکرد مدل تاثیر دارند. با توجه به این که در مسئله classification تعداد داده train بسیار کم است تاثیر loss در حالت اول بسیار کم‌تر دیده می‌شود. اگر معیار loss کلی را در نظر بگیریم در حالتی بهترین نتیجه کلی را داریم که وزن هر دو تسک با هم برابر باشند. اما اگر بخواهیم روی هر تسک نتیجه بهتر داشته باشیم باید حالتی را انتخاب کنیم که وزن آن تسک زیادتر است. زیرا شبکه در جهتی حرکت می‌کند که عملکردش را روی آن تسک بهتر کند.

منابع: لینک

سوال چهارم:

تابع build_model_tune را مینویسیم که مدل مورد نظر ما را میسازد. تابع add_block هم یک block کانولوشنی به مدل اضافه می کند. هر بلاک معادل دو لایه Conv2D به همراه Batch Normalization و در ادامه یک لایه Max pooling و یک لایه Dropout می باشد.

الف) با استفاده از ابزار tuner keras برای هر کدام ازهایپرپارامترها، مقادیر مختفی را در نظر می گیریم.

```
hidden_units = hp.Int(
    name='hidden_units',
    min_value=32,
    max_value=512,
    step=32,
    default=128
)

dropout_rate = hp.Choice(
    'dropout',
    values=[0.2, 0.3, 0.5])

learning_rate = hp.Choice(
    'learning_rate',
    values=[0.0001, 0.001, 0.005, 0.01]
)

blocks = hp.Int(
    'num_cnn_block',
    min_value=2,
    max_value=4,
    default=2,
)
```

مقادیر تعداد بلاک های کانولوشنی: 2، 3، 4

مقدار احتمال Dropout: 0.2، 0.3، 0.5

مقدار learning rate برای Adam: 0.0001، 0.001، 0.005، 0.01 (می توانستیم از hp.float هم استفاده کنیم.)

تعداد نورون های لایه Dense ماقبل آخر: از 32 تا 512، با step=32

```
Search space summary
Default search space size: 4
hidden_units (Int)
{'default': 128, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
dropout (Choice)
{'default': 0.2, 'conditions': [], 'values': [0.2, 0.3, 0.5], 'ordered': True}
learning_rate (Choice)
{'default': 0.0001, 'conditions': [], 'values': [0.0001, 0.001, 0.005, 0.01], 'ordered': True}
num_cnn_block (Int)
{'default': 2, 'conditions': [], 'min_value': 2, 'max_value': 4, 'step': 1, 'sampling': None}
```

ب) با استفاده از ابزار معرفی شده، از Random Search استفاده کرده، 10 مدل مختلف را در نظر گرفته و بهترین مدل ممکن را با حالات مختلف می یابیم. نتیجه برای بهترین مدل به صورت زیر است:

```
Trial 10 Complete [00h 06m 48s]
val_accuracy: 0.8201499879360199

Best val_accuracy So Far: 0.8278999924659729
Total elapsed time: 00h 58m 23s
```

```
blocks: 4
dropout rate: 0.3
learning rate: 0.01
dense hidden units: 64
```

```
loss: 0.5103631615638733
accuracy: 0.830299973487854
```

مدلی که با این هایپرپارامتر ها ساخته و کامپایل می شود بین بقیه حالات بهترین دقت را روی validation بدست آورده است. به همین دلیل نیز به عنوان بهترین مدل انتخاب می شود. تعداد بلاک ها 4 انتخاب شده است که می تواند فیچر های بیشتر و پیچیده تری را استخراج کرده و یادگیری مدل را بهبود بخشد. با توجه به تعداد بالای بلاک ها، تعداد نورون های لایه Dense هم مقدار متوسط رو به پایین انتخاب شده که با توجه به تعداد فیلتر هایی که برای لایه های میانی انتخاب کردیم تعداد پارامتر های مدل خیلی زیاد نشود و مدل overfit نشود. learning rate هم طوری تنظیم شده که سرعت یادگیری مدل با توجه به پیچیدگی آن، مناسب باشد، و نرخ dropout نیز طوری تنظیم شده که مدل دچار overfit نشود. در کل ترکیب هایپرپارامتر ها با این مقادیر باعث شده که مدل به دقت خوبی برسد.

پ) بهترین مدل را دوباره train می کنیم. و آن را روی داده تست ارزیابی می کنیم. نتیجه:

```
313/313 [==============================] - 2s 6ms/step - loss: 0.5188 - accuracy: 0.8651
test loss: 0.5187616944313049, test accuracy: 0.8651000261306763
```

```
f1 score is : 0.8647129427668758
precision score is : 0.8660314030494025
recall score is :  0.8651
```

با توجه به مقدار این معیار ها، عملکر مدل ما از همه نظر خوب بوده است.

Precision: درصد نمونه‌هایی که توسط مدل به عنوان کلاس مثبت تشخیص داده شده‌اند و درست بوده‌اند.

Recall: درصد نمونه‌هایی که مثبت بوده‌اند و به درستی توسط مدل تشخیص داده شده‌اند.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

F_Score: خلاصه‌سازی PR با یک عدد:

$$F_1 = \frac{2PR}{P + R}$$

زمانی از این معیار ها استفاده می کنیم که در تسک ما، accuracy معیار مناسبی برای ارزیابی نباشد. برای مثال دیتای ما balance نباشد. و تعداد داده های مربوط به یک کلاس خیلی بیشتر از کلاس دیگر باشد. گاهی اوقات هم مرتکب شدن یک نوع خطا از نوع دیگر پرهزینه‌تر است.

برای مثال در یک سیستم تشخیص ایمیل هرزنامه، دسته‌بندی نادرست یک پیام قانونی به عنوان هرزنامه می‌تواند هزینه بسیار بیشتری داشته باشد.

معیار Precision برای زمانی مناسب است که هزینه FP زیاد باشد. (مثل مثال بالا). و معیار recall برای زمانی مناسب است که هزینه FN زیاد باشد. مانند تشخیص سرطان. معیار F_Score هم هدف این دو را ترکیب کرده و یک دید خوب از هر دو این معیار ها به ما می دهد.

در این سوال، دیتاست ما دیتاست balance و خوبی بوده و معیار accuracy هم برای آن معیار مناسبی است. و استفاده از این معیار ها حتما لازم نیست.

ت)

TP: True Positive زمانی است که هر دو کلاس واقعی و پیش بینی شده برای نمونه، مثبت هستند.

TN: True Negative زمانی است که هر دو کلاس واقعی و پیش بینی شده برای نمونه، منفی هستند.

FP: False Positive زمانی است که کلاس واقعی نمونه، منفی بوده اما کلاس پیش بینی شده مثبت است.

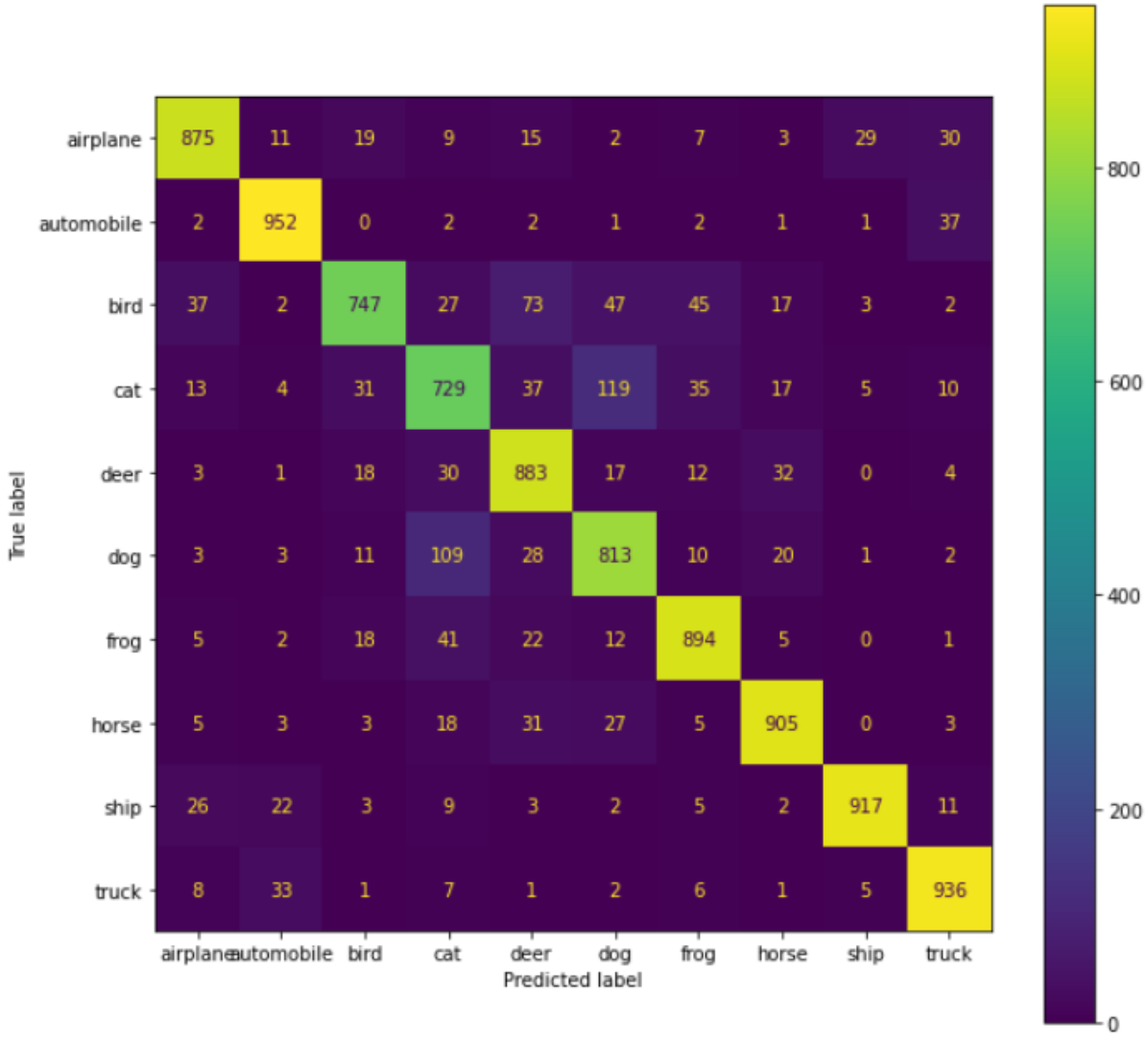
FN: False Negative زمانی است که کلاس واقعی نمونه، مثبت بوده اما کلاس پیش بینی شده منفی است.

برای این تسک این مقادیر به صورت زیر بدست آمده اند:

```
TP:  875
FP:  11
FN:  2
TN:  952
```

با توجه به confusion matrix، مدل ما عملکر خیلی خوبی در پیش بینی اکثر کلاس ها داشته است و تعداد دیتای درست تشخیص داده شده در اکثر کلاس ها بالای 800 است. مدل بیشترین خطا را در کلاس گربه داشته است که 119 گربه را سگ پیش بینی کرده است و در مجموع 729 گربه را درست پیش بینی کرده است. همچنین برای کلاس پرنده هم 747 نمونه را درست پیش بینی کرده و مثلا 73 تا پرنده را آهو پیش بینی کرده است. همچنین 109 سگ، گربه تشخیص داده شده اند.

با توجه به این ماتریس و مقادیر TP و FP و TN و FN می توان گفت مدل در قسمت Positive ضعیفتر عمل کرده، چون FP بالاتری از FN دارد و TP پایین تری از TN دارد. 11 تا FP داشته است. و تنها 2 تا FN بودند.



منابع: لینک