دانشکده مهندسی کامپیوتر

مباحث ویژه ۱ (یادگیری عمیق)

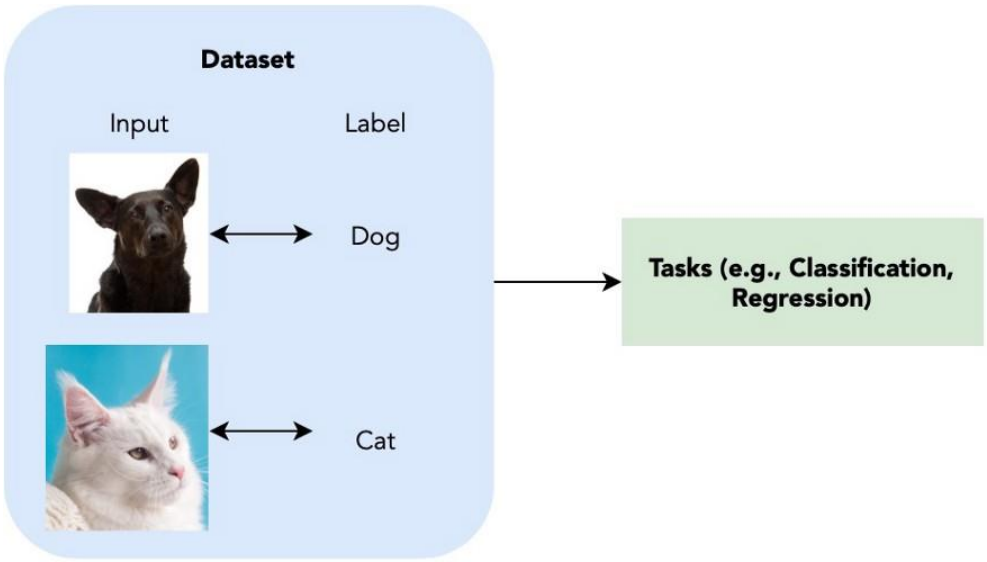
تمرین سری سیزدهم

علی صداقی

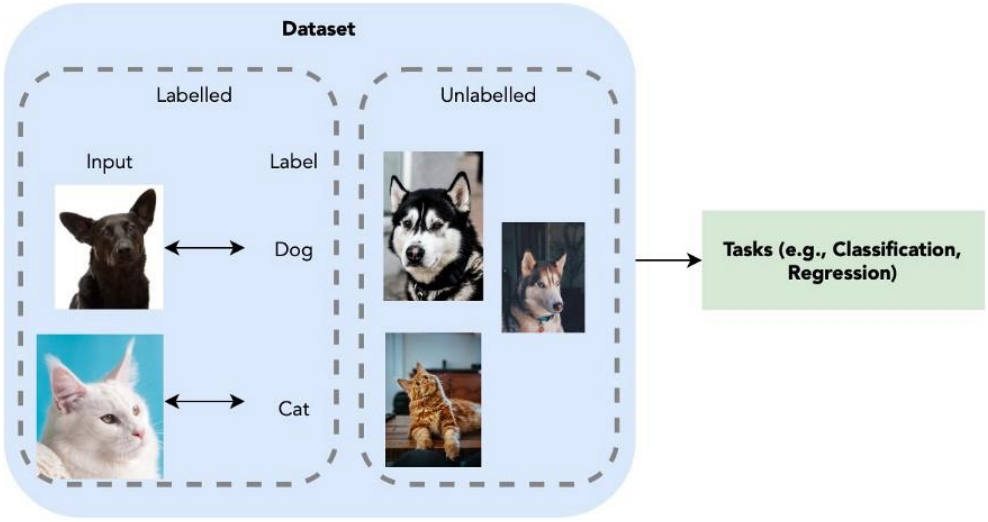
۹۷۵۲۱۳۷۸

۱ سوال اول

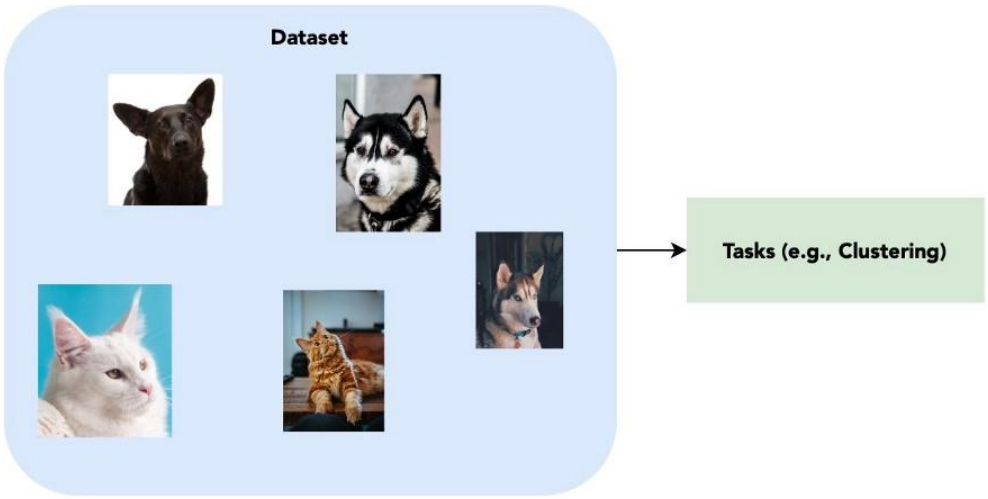
Supervised Learning: همان نوع یادگیری است که تا کنون با آن سروکار داشتیم. در این نوع یادگیری زوج‌های (Input, Output) را داریم و هدف یادگیری سیستمی است که بتواند ورودی را به خروجی درست نگاشت دهد. برای مثال دسته‌بندی (Classification) گربه و سگ از این نوع یادگیری است که ورودی یک تصویر و خروجی کلاس یا Label آن تصویر است. برای یادگیری همچین سیستمی یک تابع ضرر میان خروجی واقعی و خروجی پیشبینی شده در نظر می‌گیریم سپس با استفاده از Backpropagation این خطا را میان وزن‌های قابل آموزش شبکه منتشر می‌کنیم تا وزن‌ها در جهت کاهش خطا بروز شوند.



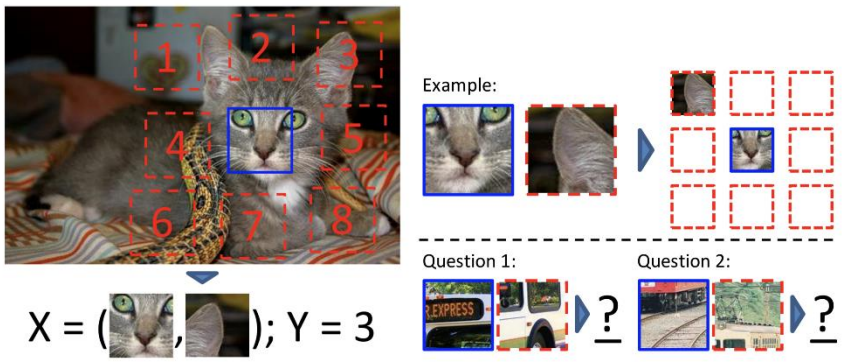
Semi-Supervised Learning: همان یادگیری Supervised می‌باشد با این تفاوت که تمام دیتاست دارای Label نیست و بخشی از داده‌های ورودی بدون Label هستند. دلیل این بی Label بودن این است که فرایند Label زدن کاری زمان‌بر است که توسط انسان صورت می‌گیرد. برای حل مشکل بدون Label بودن از روش pseudo-labelling استفاده می‌کنیم. ابتدا مدل را با استفاده از داده‌های با Label آموزش می‌دهیم سپس مدل آموزش دیده را روی داده‌های بدون Label اجرا می‌کنیم تا شبکه به ازای هر ورودی یک Label پیشبینی کند. سپس پیشبینی‌هایی که دارای Confidence بالا هستند را به عنوان داده Label دار (pseudo label) وارد شبکه می‌کنیم و آموزش را ادامه می‌دهیم. این کار را مکررا انجام می‌دهیم تا از تمامی داده‌ها برای آموزش مدل استفاده شود. در این روش اگر داده‌های دارای Label کم باشد ممکن است دچار Overfit شدید شویم و pseudo label های اشتباه ایجاد کنیم و با استفاده از این pseudo label های اشتباه مدل را به سمتی کاملا اشتباه هدایت کنیم. می‌توانیم با استفاده از Augmentation تعداد داده‌های با Label را افزایش دهیم که مشکل Overfit کم‌تر شود.

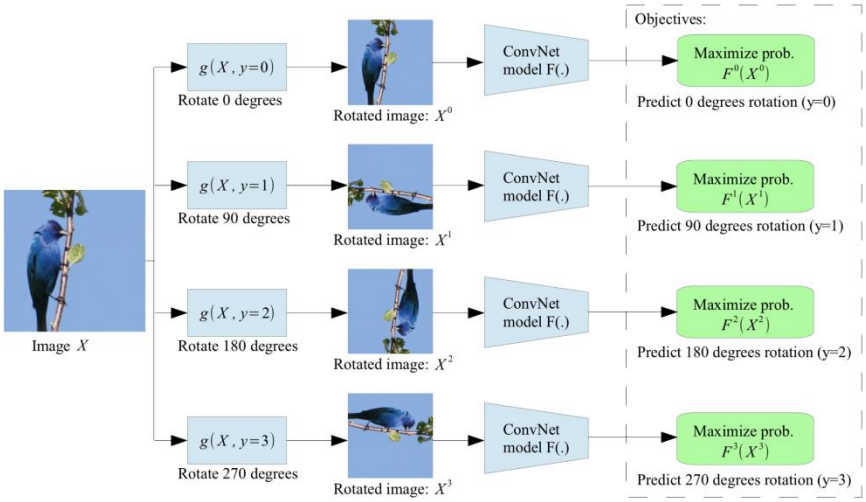


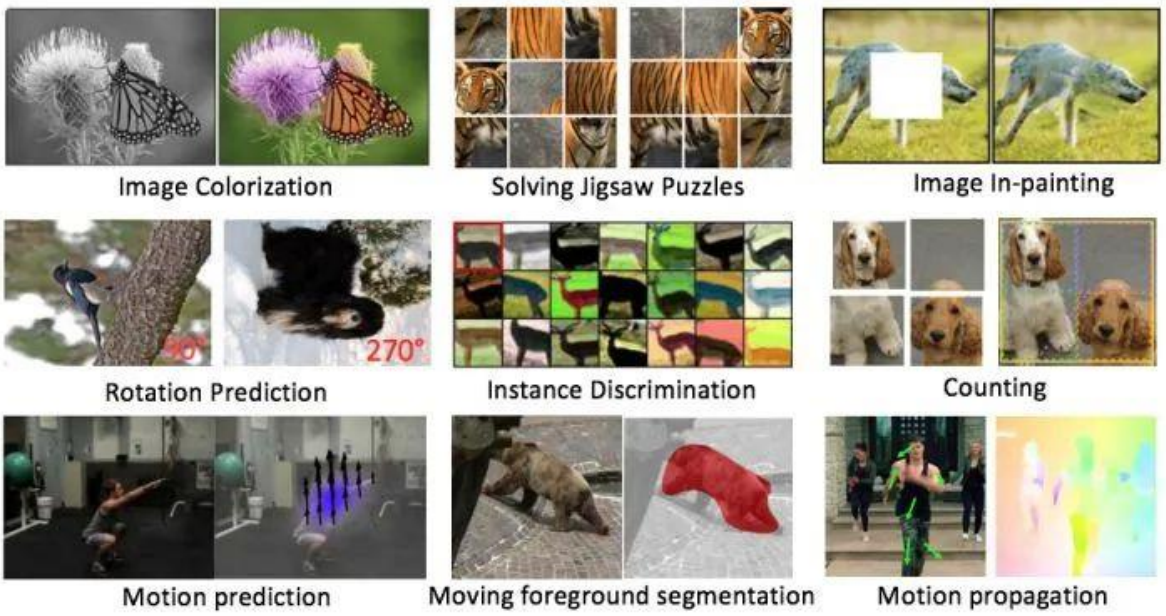
Unsupervised Learning: در این نوع یادگیری دیتاست مطلقا بدون Label می‌باشد و تنها دارای ورودی هستیم. هدف در این نوع یادگیری پیدا کردن الگوهای (Patterns) حاکم بر دیتاست است. کارهایی که می‌توان با استفاده از این نوع یادگیری انجام داد شامل Customer segmentation، Recommendation systems و ... است. با توجه به این که در این نوع یادگیری هیچ Label، Class یا جواب درست نداریم پس بهترین راه برای پیدا کردن این الگوها خوشه‌بندی (Clustering) است. به عبارت دیگر با داشتن ویژگی (Feature) های داده ورودی، هدف پیدا کردن ویژگی‌هایی است که شبیه همدیگر هستند، سپس این ویژگی‌های شبیه به هم را با هم یک گروه می‌کنیم. از روش‌های کلاسترینگ K-Means یا K-Medoids می‌توانیم استفاده کنیم. کلاسترینگ بینش خوبی از دیتاست ایجاد می‌کند. برای مثال در Recommendation system با گروه‌بندی کاربران بر اساس فعالیت‌هایشان می‌توانیم مطالب پیشنهادی به یک کاربر در یک کلاستر را به تمامی کاربران درون آن کلاستر پیشنهاد دهیم بدون اینکه از آن سلیقه و جزییات آن کاربران با خبر باشیم.



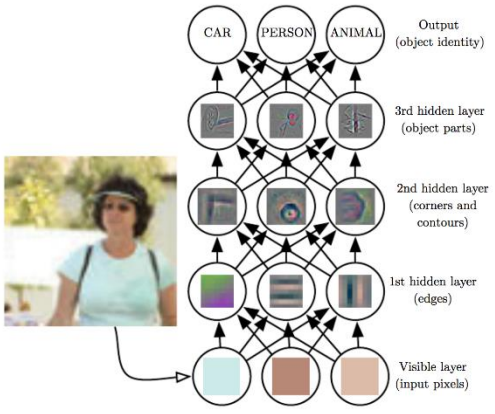
Self-Supervised Learning: از آنجایی که در این نوع یادگیری دیتاست هیچ Labelیی ندارد می‌توان آن را به نوعی Unsupervised Learning تلقی کرد. اما در این نوع یادگیری برخلاف Unsupervised به دنبال پیدا کردن الگوهای (Patterns) سطح بالا برای Clustering نیستیم. در واقع در این نوع یادگیری قصد داریم مسائل Supervised Learning مثل Classification را بدون وجود داده Label دار حل کنیم. در نگاه اول حل کردن چنین مسئله‌ای غیر ممکن به نظر می‌رسد. اما تحقیقات جدید روش‌های خلاقانه‌ای برای حل چنین مسائلی ارائه داده‌اند. یکی از این روش‌ها روش یادگیری تضادی (Contrastive Learning) از زوج‌های مثبت و منفی است. به طور خلاصه این روش با انجام داده‌افزایی (Augmentation) روی تصاویر یکسان و اختصاص Label مثبت به آنها و اختصاص Label منفی به سایر تصاویر، سعی می‌کند که ویژگی‌های یاد گرفته شده از تصاویر منفی را دور بگذارد و ویژگی‌های یاد گرفته شده از تصاویر مثبت را نزدیک‌تر سازد. این کار باعث می‌شود شبکه گروه‌بندی تصاویر با کلاس مشابه را یاد بگیرد. با استفاده از Transfer Learning می‌توانیم ویژگی‌های آموخته شده را به وظایف Downstream منتقل کنیم. مثلا در پیش‌بینی موقعیت اشیا بخشی از تصویر را به صورت تصادفی انتخاب می‌کنیم، سپس برای n ناحیه اطراف آن برچسب تولید می‌کنیم و مدل را طوری آموزش می‌دهیم که موقعیت تصاویر را نسبت به هم تشخیص دهد.

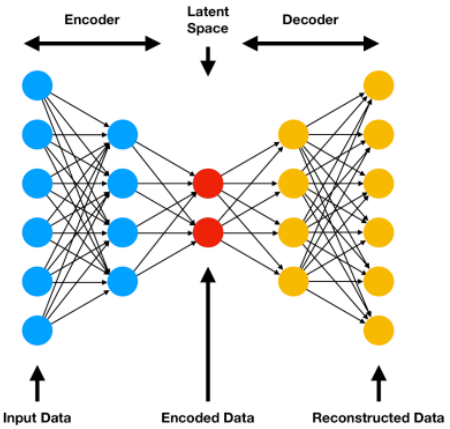






Representation Learning: یادگیری بازنمایی (Feature Learning) روشی است که به مدل امکان یادگیری خودکار ویژگی‌ها و بازنمایی‌های مورد نیاز را از دیتاست می‌دهد. این نوع یادگیری می‌تواند Supervised یا Unsupervised باشد. مثلا در تمرین Transfer Learning ما با استفاده از مدل Pretrain شده روی ImageNet و انتقال دانش آموخته شده توانستیم تسک دیگری را حل کنیم (Supervised). این کار باعث می‌شود نقطه شروع مدل تصادفی نباشد و وزن‌های اولیه بیانگر بازنمایی مطلوبی از داده ورودی باشد. در یادگیری بازنمایی بدون ناظر (Unsupervised) می‌توانیم از Auto-Encoder ها استفاده کنیم. در این روش مدل وزن‌هایی را باید پیدا کند که برگشت پذیر باشند به این معنا که از وزن هر لایه بتوانیم ورودی همان لایه را به طور تقریبی پیشبینی کنیم، اگر این پیشبینی نزدیک داده ورودی باشد می‌توانیم بگوییم وزن‌های آن لایه اطلاعات مفیدی را آموخته اند و بیانگر بازنمایی مناسبی هستند. این روش در NLP کاربرد زیادی دارد، مثلا برای بدست آوردن Embedding.





	Supervised	Semi-Supervised	Unsupervised	Self-Supervised	Representation
Label	All	Some	None	None	All/None
Goal	Classification/Regression	Classification Regression	Clustering Reducing dimensionality	Various tasks	Feature Learning

منابع:

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

https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

https://chowdera.com/2021/01/20210109003603375e.html

https://neptune.ai/blog/understanding-representation-learning-with-autoencoder-everything-you-need-to-know-about-representation-and-feature-learning

۲ سوال دوم

الف) معیار Utility (بهره‌وری) به منظور سنجش سودمندی مدل Self-Supervised تعریف شده است. دلیل تعریف این معیار، صرفه‌جویی در استفاده از داده‌های دارای Label است. به این معنا که برای رسیدن به همان دقت بدون خودنظارتی (Self-Supervision) چند Label دیگر لازم داریم.

$a(n)$: دقت مدل آموزش دیده با استفاده از n داده دارای Label

$a_{ft}(n)$: دقت مدل Fine Tune شده

$\bar{n}$: تعداد داده Label دار مورد نیاز برای اینکه $a(\bar{n})$ برابر $a_{ft}(n)$ شود.

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

عبارت بالا نشان دهنده این است که چند Label اضافه برای رسیدن دقت مدل به حالت Fine tune شده نیاز داریم. بدیهی است که اگر میزان دقت مدل Self-Supervised با همان تعداد Label با دقت مدل برابر شود مقدار Utility برابر 0 می‌شود. اگر هیچ Label یی نیز وجود نداشته باشد که دقت دو مدل با هم برابر شود آن‌گاه Utility به سمت بی‌نهایت میل می‌کند.

ب) در این مقاله ۴ نوع Downstream Task بررسی شده است. این وظایف می‌توانند در دسته‌های معنایی، مکانی، سراسری و متراکم دسته‌بندی شوند.

- Object Classification: مدل را آموزش می‌دهیم تا میان 10 کلاس ShapeNet (که برای ارائه داده مصنوعی است استفاده می‌شود) بتواند تفکیک و دسته‌بندی داشته باشد. تصاویر تولید شده فقط دارای یک Object واحد است و در هر 10 کلاس توزیع یکنواختی داریم. عملکرد این وظیفه با متریک Accuracy سنجیده می‌شود.
- Object Pose Estimation: برای این وظیفه از تصاویری استفاده می‌شود که دارای یک Object در وسط تصویر است (Single Centered Object). برای هر شی Pose را به 5 دسته (زیرا بعضی از اشیا دارای تقارن چرخشی هستند مثل میز و لامپ) تقسیم می‌کنیم و یک Classifier را روی آن آموزش می‌دهیم. در این وظیفه مدل باید پیش‌بینی کند که بالای Object به کدام سمت Upward، Forward، Backward، Right یا Left است. ویژگی‌های باید استخراج شوند که موقعیت ۳ بعدی شی را نمایش می‌دهند. از تابع ضرر Categorical Cross Entropy استفاده شده است. متریک گزارش شده نیز Accuracy است.

- Semantic Segmentation: در این وظیفه درون هر تصویر چندین Object وجود دارد. در آن به دنبال یک Clustering روی بخش‌هایی از تصویر هستیم که مربوط به یک Object است. در واقع روی پیکسل‌های تصویر این خوشه‌بندی را انجام می‌دهیم و هر پیکسل را به یک دسته نگاشت می‌دهیم. از تابع ضرر Cross Entropy استفاده شده است. متریک گزارش شده نیز Accuracy است.
- Depth Estimation: مشابه قسمت بالا درون هر تصویر چندین Object داریم. هدف آن به دست آوردن یک Representation از ساختار فضایی یک Scene و بازیابی شکل و ظاهر سه بعدی اشیا درون تصویر است. برای آموزش آن از تابع ضرر L1 استفاده شده است و متریک Accuracy نیز گزارش شده است که بیانگر درصد پیش‌بینی هایی است که در بازه معینی از عمق Ground Truth قرار گرفته اند.

ج) از ۴ روش برای Pretraining استفاده شده است:

- Variational Auto-Encoder (VAE): برای کاهش ابعاد داده ورودی مناسب است. مثلا یک تصویر را در یک فضا با ابعاد کمتر نگاشت می‌کنیم. در این روش از یک Encoder و یک Decoder استفاده می‌کنیم. در واقع این روش شامل یک شبکه عصبی است که با بهینه‌سازی می‌توانیم وزن‌های مناسب برای Encode کرد را بیابیم. (مشابه کاری که در تمرین برای بدست آوردن Embedding انجام دادیم.
- Rotation: وظیفه این روش تشخیص چرخش در تصویر است. یعنی پیش‌بینی می‌کند که آیا تصویر ورودی 0، 90، 180 یا 270 درجه چرخیده است یا خیر.
- Contrastive Multiview Coding (CMS): در این روش یک تصویر را به کانال‌های متعدد تقسیم می‌کنیم (Channel Splitting) مثلا تصویر درون فضای Lab را به کانال‌های L و ab تفکیک می‌کنیم. سپس کانال‌های جدا شده از دو شبکه نصف شده عبور داده می‌شوند و سپس Embedding های خروجی شده با Embedding های تصاویر گروه دیگر مقایسه می‌شود.
- Augmented Multiscale Deep InfoMax (AMDIM): این روش مشابه روش بالا است یعنی مدل را با Contrastive Code (کدگذاری متضاد) کردن آموزش می‌دهد. تفاوت با روش بالا در این است که به جای استفاده از کانال‌های مختلف درون تصویر، از تصاویری که در Augmentation به تصویر اصلی اضافه شده اند استفاده می‌شود. در این روش از نمایش‌های تولید شده در لایه‌های میانی شبکه برای مقایسه استفاده می‌کنیم.

۳ سوال سوم

در قسمت اول جمله‌ها را با استفاده از دیکشنری به بردار عددی تبدیل می‌کنیم.

```python
def vectorize_stories(data, word_idx, story_maxlen, query_maxlen):

    ##########################################
    #     Put your implementation here     #
    ##########################################
    data_length = len(data)
    inputs_train = np.zeros((data_length, story_maxlen))
    queries_train = np.zeros((data_length, query_maxlen))
    answers_train = np.zeros((data_length, len(word_idx)+1))

    i = 0
    for input, query, answer in data:
        j = 0
        for word in input:
            inputs_train[i, j] = word_idx[word]
            j += 1

        j = 0
        for word in query:
            queries_train[i, j] = word_idx[word]
            j += 1

        answers_train[i, word_idx[answer]] = 1
        i += 1

    return inputs_train, queries_train, answers_train
```

در ۲ قسمت بعدی توابع ترسیم نمودار و گزارش بهترین اپاک از نظر Loss و Accuracy را پیاده‌سازی می‌کنیم.

```python
def plot_acc(history, title):

    # This function should show not only the plot of accuracy on training and validation set
    # but also it should show the maximum value of accuracy with its related epoch.
    ##########################################
    #     Put your implementation here     #
    ##########################################
    train_metric = history.history['accuracy']
    val_metric = history.history['val_accuracy']

    plt.title(title)
    plt.plot(train_metric)
    plt.plot(val_metric)
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['training', 'validation'], loc='lower right')
    plt.grid()
    plt.show()

    max_acc_epoch = train_metric.index(max(train_metric))
    max_val_acc_epoch = val_metric.index(max(val_metric))
    print(f"Max train accuracy is {train_metric[max_acc_epoch]} at epoch {max_acc_epoch+1}")
    print(f"Max val accuracy is {val_metric[max_val_acc_epoch]} at epoch {max_val_acc_epoch+1}")
```

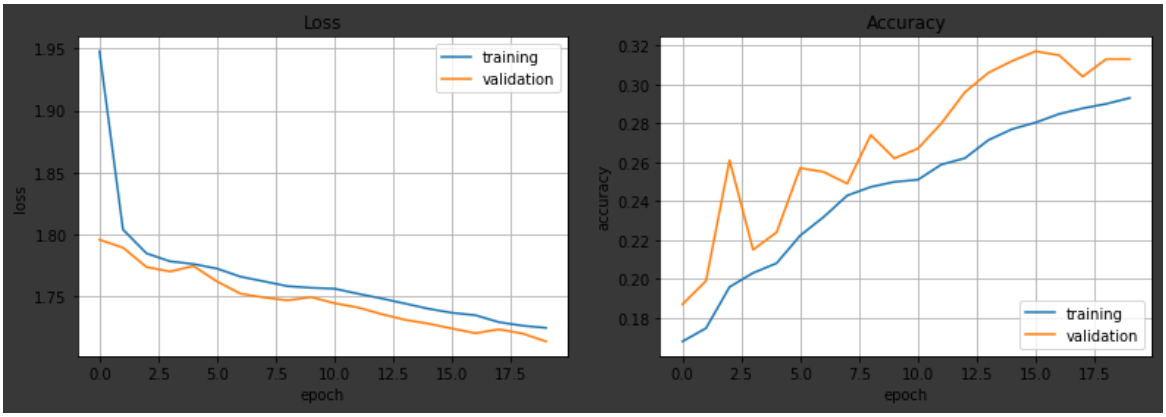
```python
def plot_loss(history, title):

    # This function should show not only the plot of loss on training and validation set
    # but also it should show the minimum value of loss with its related epoch.
    ##########################################
    #     Put your implementation here     #
    ##########################################
    train_metric = history.history['loss']
    val_metric = history.history['val_loss']

    plt.title(title)
    plt.plot(train_metric)
    plt.plot(val_metric)
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['training', 'validation'], loc='upper right')
    plt.grid()
    plt.show()

    min_loss_epoch = train_metric.index(min(train_metric))
    min_val_loss_epoch = val_metric.index(min(val_metric))
    print(f"Min train loss is {train_metric[min_loss_epoch]} at epoch {min_loss_epoch+1}")
    print(f"Min val loss is {val_metric[min_val_loss_epoch]} at epoch {min_val_loss_epoch+1}")
```

با توجه به شماتیک و Summary مدل آن را پیاده‌سازی می‌کنیم.

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, Permute, LSTM
from tensorflow.keras.layers import Add, Dot, Concatenate
from tensorflow.keras.layers import Dropout, Dense, Activation

sequential1 = Sequential([Embedding(vocab_size, 64)])(input_sequence)
sequential2 = Sequential([Embedding(vocab_size, 4)])(input_sequence)
sequential3 = Sequential([Embedding(vocab_size, 64)])(question)
x = Dot(axes=(2, 2))([sequential1, sequential3])
x = Activation('relu')(x)
x = Add()([x, sequential2])
x = Permute((2, 1))(x)
x = Concatenate()([x, sequential3])
x = LSTM(lstm_size)(x)
x = Dropout(0.1)(x)
x = Dense(vocab_size)(x)
answer = Activation('softmax')(x)
```

مدل را در 20 ایپاک آموزش می‌دهیم. نتایج به صورت زیر است:

```
Epoch 15/20
313/313 [==============================] - 19s 60ms/step - loss: 1.7400
- accuracy: 0.2770 - val_loss: 1.7280 - val_accuracy: 0.3120
Epoch 16/20
313/313 [==============================] - 19s 60ms/step - loss: 1.7368
- accuracy: 0.2804 - val_loss: 1.7241 - val_accuracy: 0.3170
Epoch 17/20
313/313 [==============================] - 19s 60ms/step - loss: 1.7348
- accuracy: 0.2848 - val_loss: 1.7203 - val_accuracy: 0.3150
Epoch 18/20
313/313 [==============================] - 19s 60ms/step - loss: 1.7292
- accuracy: 0.2877 - val_loss: 1.7234 - val_accuracy: 0.3040
Epoch 19/20
313/313 [==============================] - 19s 60ms/step - loss: 1.7264
- accuracy: 0.2900 - val_loss: 1.7200 - val_accuracy: 0.3130
Epoch 20/20
313/313 [==============================] - 18s 59ms/step - loss: 1.7246
- accuracy: 0.2930 - val_loss: 1.7136 - val_accuracy: 0.3130
```




```
Min train loss is 1.724563479423523 at epoch 20
Min val loss is 1.713639259338379 at epoch 2

Max train accuracy is 0.2930000126361847 at epoch 20
Max val accuracy is 0.31700000166893005 at epoch 16
```

با اینکه تعداد Epoch را از عدد 10 به 20 تغییر دادیم همچنان مدل مشکل Underfit شدید را دارد و نتوانسته عملکرد خوبی چه روی داده آموزشی و ارزیابی داشته باشد. قطعا اگر مدل را مدت طولانی‌تری آموزش دهیم نتیجه بهتری نیز خواهیم داشت. همانطور که از تصویر زیر نیز پیداست مدل اشتباهات زیادی دارد.

```
Sandra travelled to the kitchen . Sandra travelled to the hallway . Where is Sandra ? | Prediction: hallway | Ground Truth: garden
--------------------------------------------------------------------------------
Sandra travelled to the kitchen . Sandra travelled to the hallway . Where is Sandra ? | Prediction: hallway | Ground Truth: office
--------------------------------------------------------------------------------
John travelled to the office . Mary journeyed to the kitchen . Where is Mary ? | Prediction: kitchen | Ground Truth: kitchen
--------------------------------------------------------------------------------
John travelled to the office . Mary journeyed to the kitchen . Where is Daniel ? | Prediction: kitchen | Ground Truth: office
--------------------------------------------------------------------------------
John travelled to the office . Mary journeyed to the kitchen . Where is Mary ? | Prediction: kitchen | Ground Truth: bedroom
--------------------------------------------------------------------------------
John moved to the hallway . John journeyed to the kitchen . Where is John ? | Prediction: kitchen | Ground Truth: garden
--------------------------------------------------------------------------------
John moved to the hallway . John journeyed to the kitchen . Where is Daniel ? | Prediction: garden | Ground Truth: office
--------------------------------------------------------------------------------
```

این مدل در تلاش است با دریافت چند جمله آن را بیاموزد و سپس پرسش مربوطه را پاسخ دهد. برای تحقق این امر Embedding مربوط به Story و Query را در هم ضرب (Dot) می‌کنیم سپس Embedding مربوط به Story را با آن جمع (Add) می‌کنیم. از لایه بازگشتی LSTM برای افزایش قدرت حافظه استفاده کردیم.

مزایا: این یک مدل ساده و بدون پیچیدگی است که از ترکیب چندین مدل Sequential که دارای ترکیبات Story و Query هستند شکل گرفته است. تعداد کل پارامترهای آن 17838 است که بسیار کم و مناسب است و قادر است در صورت آموزش طولانی‌تر به نتیجه مطلوب برسد (رفتار منحنی).

معایب: این مدل قدرت تعمیم کمی دارد یعنی نمی‌توانیم جملاتی با کلمات متفاوت را روی آن بررسی کنیم. دلیل این اتفاق کوچ بودن دیکشنری استخراج شده از دیتاست است. بدیهی است که یک دیکشنری با 21 کلمه مناسب Taskهای جهان واقعی (Real World) نیست.

برای استفاده مدل در جهان واقعی باید موارد زیر را اصلاح کنیم:

۱. دیکشنری گسترده‌تری استفاده کنیم.
۲. زمان آموزش را بیشتر کنیم.
۳. از Embeddingهای از پیش آموخته مانند Word2Vec یا Glove استفاده کنیم.
۴. دیتاست بزرگ‌تر و عمومی‌تر که شامل جملات عام و روزانه هست استفاده کنیم.

۴ سوال چهارم

ابتدا داده‌ها را با تقسیم بر ۲۵۵ نرمالیزه می‌کنیم سپس به فرم Categorical می‌بریم.

```
x_train = x_train / 255.0
x_test = x_train / 255.0
y_train = to_categorical(y_train, num_classes=NUM_CLASSES)
y_test = to_categorical(y_test, num_classes=NUM_CLASSES)
```

مدل را در تمامی حالات با Batch Size = 32 آموزش می‌دهیم.

یک مدل Core طراحی می‌کنیم که لایه آخر (Classifier) ندارد و لایه آخر را متناسب با مسئله اضافه می‌کنیم.

```
Model: "core"
_________________________________________________________________
 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
_________________________________________________________________
```

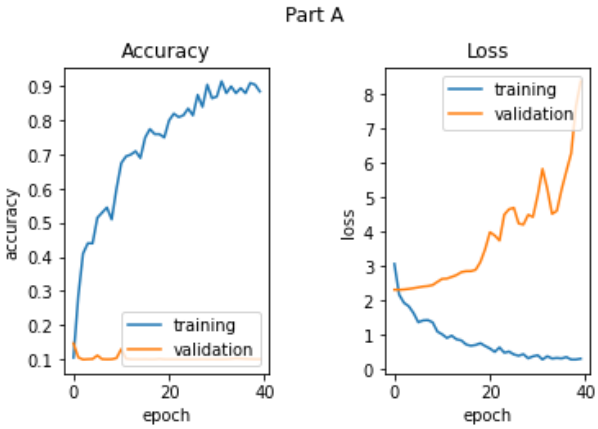
از بهینه‌ساز Adam و تابع ضرر Categorical Cross Entropy استفاده خواهیم کرد.

الف) به مدل Core یک لایه Dense با اضافه می‌کنیم که نقش Classifier ما را دارد. مدل جدید به این صورت است:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 core (Functional)           (None, 1024)              230336

 dense_2 (Dense)             (None, 10)                10250

=================================================================
Total params: 240,586
Trainable params: 238,090
Non-trainable params: 2,496
_________________________________________________________________
```

مدل را در 40 ایپاک آموزش می‌دهیم. نتایج به صورت زیر است:

```
Epoch 35/40
7/7 [==============================] - 1s 222ms/step - loss: 0.3129 - accuracy: 0.8800 - val_loss: 4.5940 - val_accuracy: 0.1027
Epoch 36/40
7/7 [==============================] - 1s 222ms/step - loss: 0.2989 - accuracy: 0.8950 - val_loss: 5.2170 - val_accuracy: 0.1020
Epoch 37/40
7/7 [==============================] - 1s 206ms/step - loss: 0.3382 - accuracy: 0.8800 - val_loss: 5.7504 - val_accuracy: 0.1013
Epoch 38/40
7/7 [==============================] - 1s 207ms/step - loss: 0.2661 - accuracy: 0.9100 - val_loss: 6.2657 - val_accuracy: 0.1008
Epoch 39/40
7/7 [==============================] - 1s 208ms/step - loss: 0.2670 - accuracy: 0.9050 - val_loss: 7.6357 - val_accuracy: 0.1003
Epoch 40/40
7/7 [==============================] - 1s 223ms/step - loss: 0.2911 - accuracy: 0.8850 - val_loss: 8.3649 - val_accuracy: 0.1003
```



همانطور که می‌توانستیم حدس بزنیم شبکه دچار Overfit بسیار شدید است. زیرا تنها 200 داده دارای Label هستند و آموزش روی آن‌ها صورت گرفته و شبکه توانسته این 200 داده را حفظ کند. اما دقت روی داده ارزیابی که شامل 49800 تصویر است اصلا جالب نیست به طوری که گویا شبکه به صورت Random (دقت ۱۰ درصد) پیش‌بینی کرده است.

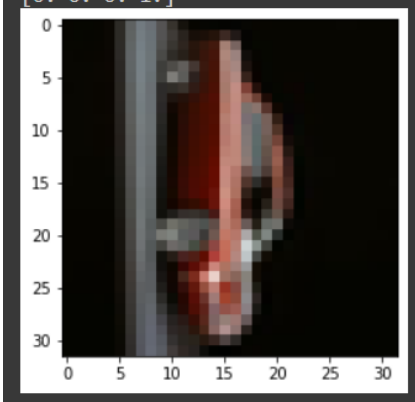
ب) ابتدا تصاویر اصلی را در 4 زاویه به صورت شانسی Rotate می‌کنیم. سپس به خروجی را به فرم One-Hot با 4 کلاس می‌بریم.

```
def rotate(x, deg):
    if deg == 0:
        return x

    elif deg == 90:
        return np.rot90(x)

    elif deg == 180:
        return np.rot90(np.rot90(x))

    else:
        return np.rot90(np.rot90(np.rot90(x)))
```

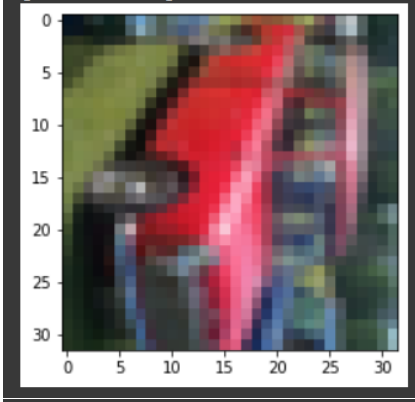
```
x_train_rotate = np.zeros_like(x_unlabeld)
y_train_rotate = np.zeros((x_unlabeld.shape[0], 4))

labels = [0, 1, 2, 3]
degrees = [0, 90, 180, 270]

for i in range(x_train_rotate.shape[0]):
    label = random.choices(labels, weights=[1, 3, 3, 3], k=1)[0]
    deg = degrees[label]
    rotated_image = rotate(x_unlabeld[i], deg)

    x_train_rotate[i] = rotated_image
    y_train_rotate[i] = to_categorical(label, num_classes=4)
```

برای مثال دو تصویر زیر هر دو 270 درجه دوران پیدا کرده اند بنابراین مربوط به کلاس آخر می‌باشند.

```
[0. 0. 0. 1.]
```




```
[0. 0. 0. 1.]
```

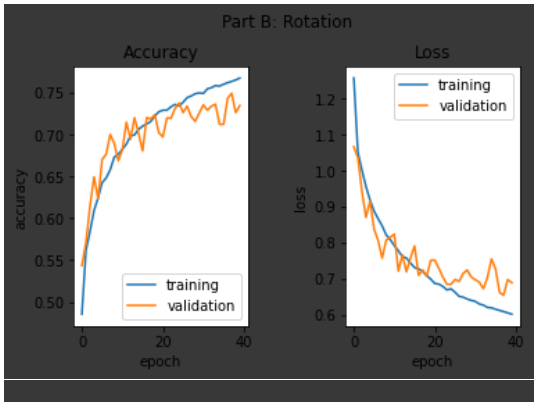


مدل پایه Core را با یک لایه Dense با 4 نورون (Classifier) ترکیب می‌کنیم.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 core (Functional)           (None, 1024)              230336

 dense_4 (Dense)             (None, 4)                 4100

=================================================================
Total params: 234,436
Trainable params: 231,940
Non-trainable params: 2,496
_________________________________________________________________
```

20 درصد داده‌ها را به عنوان Validation در نظر می‌گیریم. مدل را در 40 ایپاک آموزش می‌دهیم. نتیجه به صورت زیر است.

```
Epoch 35/40
1245/1245 [==============================] - 14s 11ms/step - loss: 0.6194 - accuracy: 0.7577 - val_loss: 0.7550 - val_accuracy: 0.7120
Epoch 36/40
1245/1245 [==============================] - 14s 11ms/step - loss: 0.6155 - accuracy: 0.7599 - val_loss: 0.7278 - val_accuracy: 0.7120
Epoch 37/40
1245/1245 [==============================] - 14s 11ms/step - loss: 0.6121 - accuracy: 0.7617 - val_loss: 0.6619 - val_accuracy: 0.7432
Epoch 38/40
1245/1245 [==============================] - 14s 11ms/step - loss: 0.6088 - accuracy: 0.7633 - val_loss: 0.6542 - val_accuracy: 0.7491
Epoch 39/40
1245/1245 [==============================] - 14s 11ms/step - loss: 0.6054 - accuracy: 0.7650 - val_loss: 0.6974 - val_accuracy: 0.7260
Epoch 40/40
1245/1245 [==============================] - 14s 11ms/step - loss: 0.6019 - accuracy: 0.7673 - val_loss: 0.6891 - val_accuracy: 0.7344
```

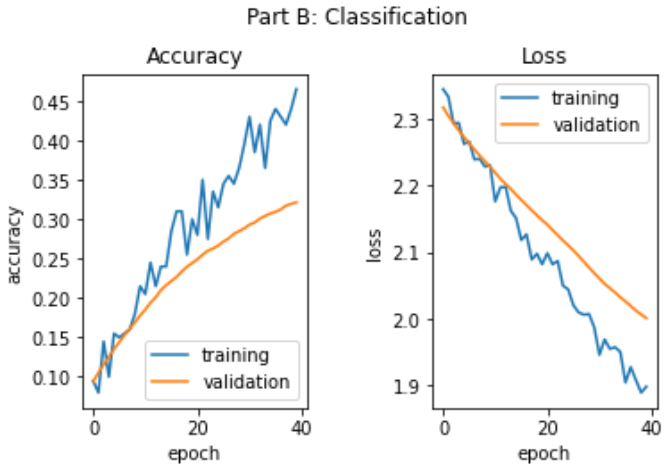


هدف از این بخش یادگیری Self-Supervised می‌باشد. از وزن‌های بدست آمده روی تسک ساختگی Rotation برای Downstream Task دسته‌بندی استفاده می‌کنیم. برای انجام اینکار مدل از پیش‌آموخته را بدون لایه آخر برمی‌داریم و با یک دسته‌بند جدید 10 کلاسه ترکیب می‌کنیم. نرخ آموزش را نیز از 0.001 به 0.0001 می‌رسانیم.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 core (Functional)           (None, 1024)              230336

 dense_5 (Dense)             (None, 10)                10250

=================================================================
Total params: 240,586
Trainable params: 238,090
Non-trainable params: 2,496
_________________________________________________________________
```

نتایج به دست آمده به صورت زیر است:

```
Epoch 35/40
7/7 [==============================] - 1s 222ms/step - loss: 1.9503 - accuracy: 0.4250 - val_loss: 2.0324 - val_accuracy: 0.3072
Epoch 36/40
7/7 [==============================] - 1s 211ms/step - loss: 1.9039 - accuracy: 0.4400 - val_loss: 2.0261 - val_accuracy: 0.3094
Epoch 37/40
7/7 [==============================] - 1s 222ms/step - loss: 1.9272 - accuracy: 0.4300 - val_loss: 2.0193 - val_accuracy: 0.3124
Epoch 38/40
7/7 [==============================] - 1s 222ms/step - loss: 1.9078 - accuracy: 0.4200 - val_loss: 2.0123 - val_accuracy: 0.3172
Epoch 39/40
7/7 [==============================] - 1s 222ms/step - loss: 1.8888 - accuracy: 0.4400 - val_loss: 2.0064 - val_accuracy: 0.3195
Epoch 40/40
7/7 [==============================] - 1s 223ms/step - loss: 1.8977 - accuracy: 0.4650 - val_loss: 2.0008 - val_accuracy: 0.3214
```



به وضوح می‌توان تاثیر Self-Supervised Learning و انتقال دانش آموخته شده از Rotation Task به Classification Downstream Task را مشاهده کرد. در حالت عادی (بخش الف) دقت روی داده ارزیابی بیشتر از 10 درصد نشد اما در این روش به دقت 32 درصد رسیدیم. از نمودارها پیداست که مدل همچنان ظرفیت آموزش را دارد و با افزایش مدت آموزش می‌توانستیم به دقت بالاتر نیز برسیم. یکی از عواملی که سرعت الگوریتم را کم کرد کاهش نرخ آموزش بود که شاید باید این کار با شدت کمتری صورت می‌گرفت.

پ) ابتدا با استفاده از مدل پایه Core و دو لایه Classifier یک مدل با دو خروجی می‌سازیم.

```python
def create_model():
    core_model = Sequential([model_structure()])

    classifier = Dense(
        units=10,
        activation='softmax',
        name='classifier'
    )(core_model.outputs[0])

    rotator = Dense(
        units=4,
        activation='softmax',
        name='rotator'
    )(core_model.outputs[0])

    two_out_model = Model(
        inputs=core_model.inputs,
        outputs=[classifier, rotator]
    )

    return two_out_model
```

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

 core (Functional)              (None, 1024)         230336      ['core_input[0][0]']

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

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

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

سپس دیتاست را به گونه‌ای تغییر می‌دهیم که دارای یک ورودی و دو خروجی (Target) باشد.

```
y_train_class = np.concatenate((y_train, np.zeros((x_train_rotate.shape[0], 10))), axis=0)
y_train_rot = np.concatenate((np.zeros((y_train.shape[0], 4)), y_train_rotate), axis=0)

x_train_all = np.concatenate((x_train, x_train_rotate), axis=0)
y_train_all = [y_train_class, y_train_rot]

print(x_train_all.shape)
print(y_train_all[0].shape)
print(y_train_all[1].shape)
```

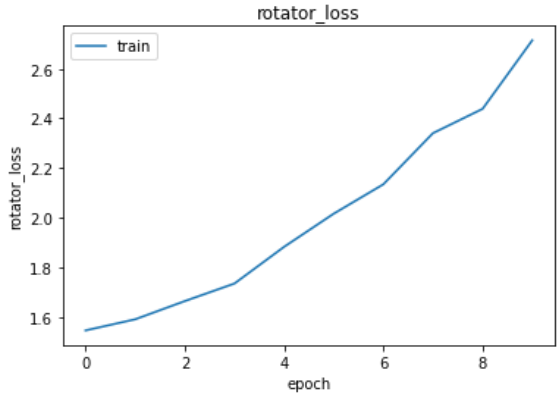
```
(50000, 32, 32, 3)
(50000, 10)
(50000, 4)
```

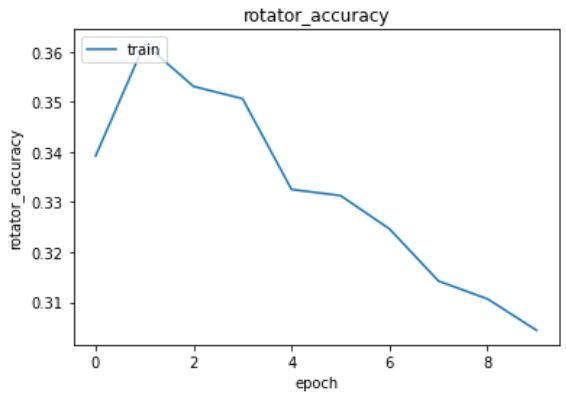
مدل را در سه حالت و در 10 Epoch آموزش می‌دهیم.

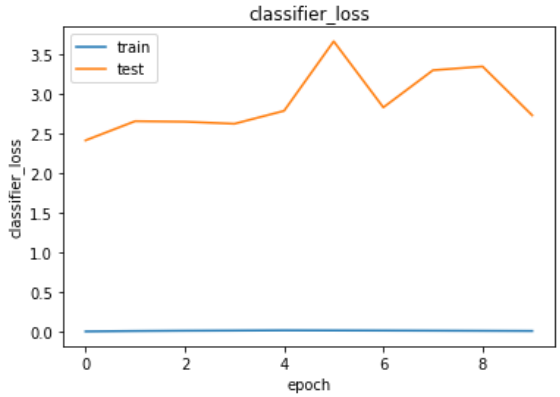
حالت اول (A): در این حالت وزن خطای خروجی مسئله دسته‌بندی 10 برابر مسئله چرخش است.

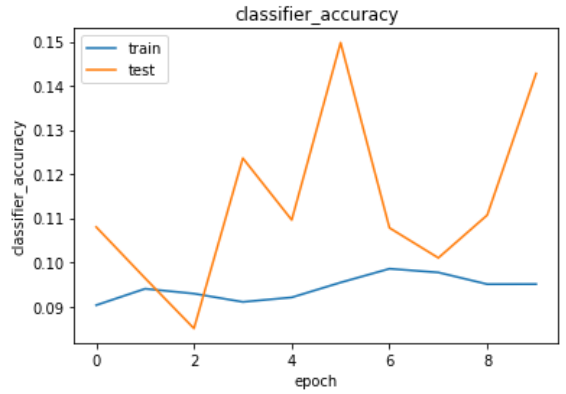
```
    loss_weights={
        'classifier': 10,
        'rotator': 1
    },
```

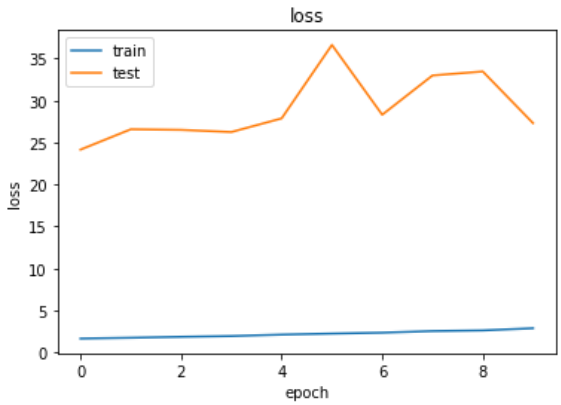
- rotator_loss: 2.7150 rotator_accuracy: 0.3044
- classifier_loss: 0.0163 val_classifier_loss: 2.7300
- classifier_accuracy: 0.0951 val_classifier_accuracy: 0.1428
- loss: 2.8776 val_loss: 27.3000







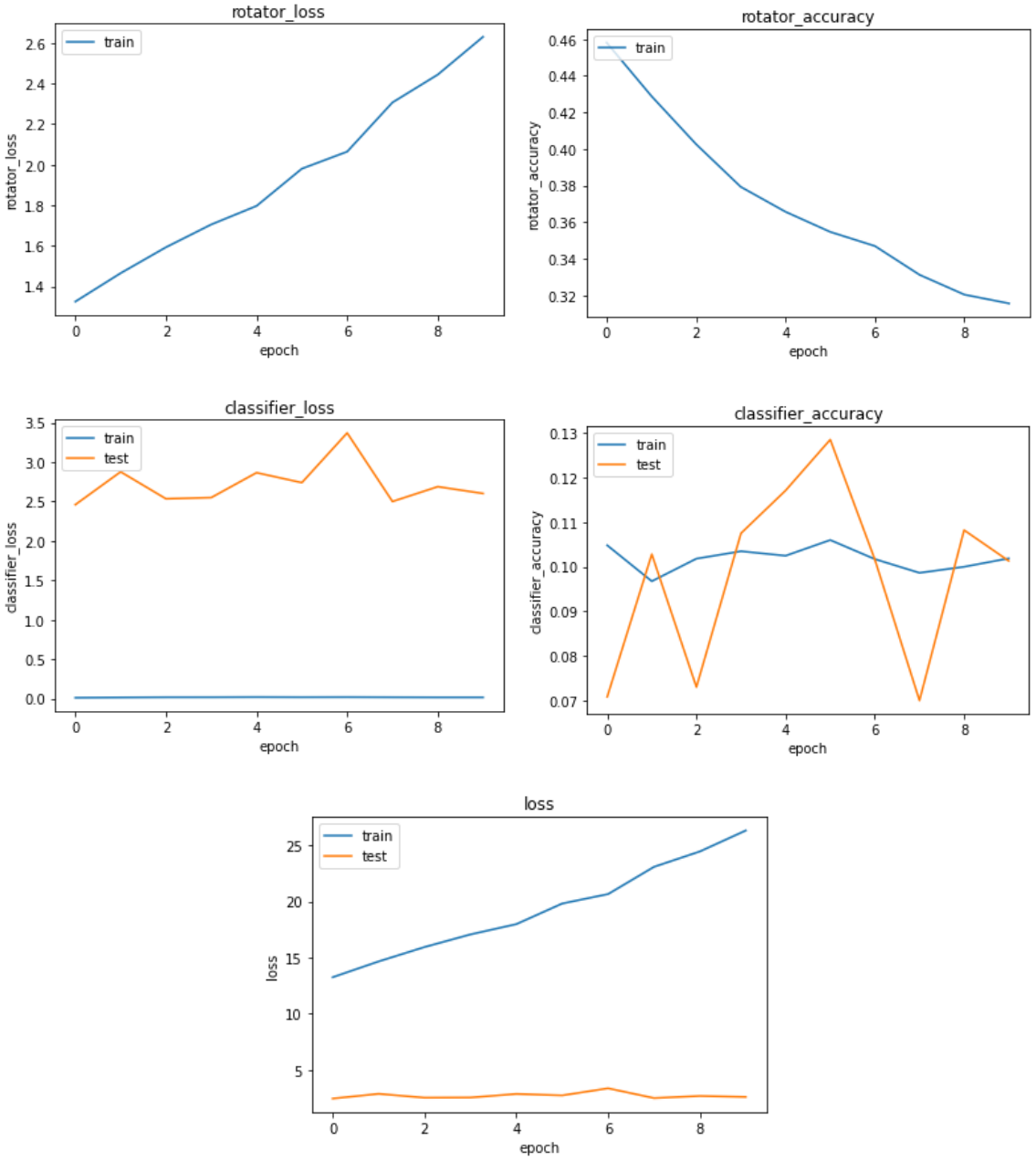




حالت دوم (B): در این حالت وزن خطای خروجی مسئله چرخش 10 برابر مسئله دسته‌بندی است.

```
    loss_weights={
        'classifier': 1,
        'rotator': 10
    },
```

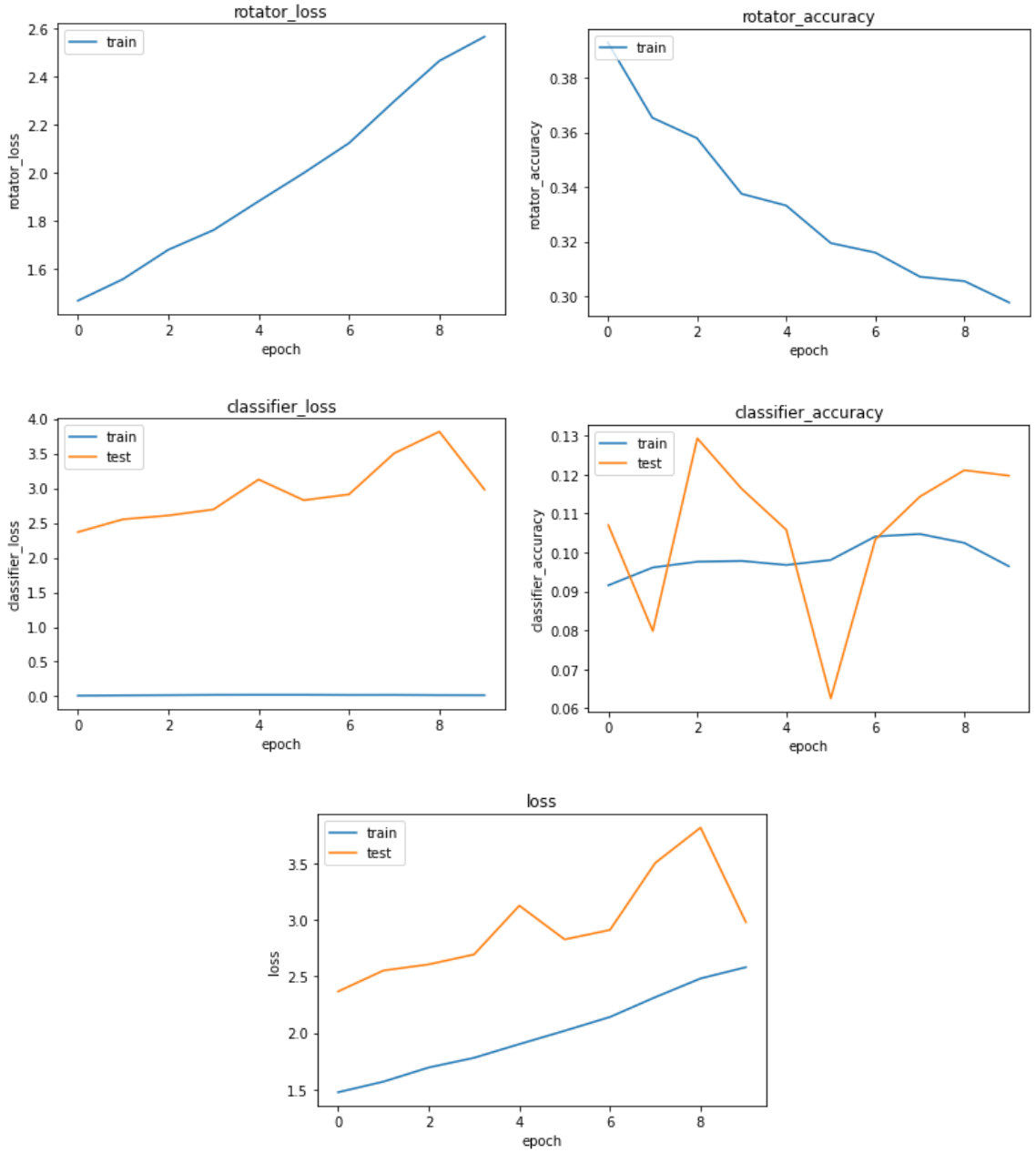
- rotator_loss: 2.6304 rotator_accuracy: 0.3155
- classifier_loss: 0.0149 val_classifier_loss: 2.6012
- classifier_accuracy: 0.1019 val_classifier_accuracy: 0.1013
- loss: 26.3193 val_loss: 2.6012



حالت سوم (C): در این حالت وزن خطای هر دو مسئله با هم برابر است.

```
    loss_weights={
        'classifier': 1,
        'rotator': 1
    },
```

- rotator_loss: 2.5636 rotator_accuracy: 0.2976
- classifier_loss: 0.0172 val_classifier_loss: 2.9785
- classifier_accuracy: 0.0964 val_classifier_accuracy: 0.1197
- loss: 2.5808 val_loss: 2.9785



تفسیر نتایج:

Rotation Task Performance: B > C > A

Classification Task Performance: A > C > B

Overall Performance: C > B > A

وزن‌هایی که بر روی توابع ضرر دو وظیفه در نظر می‌گیریم بر روی عملکرد مدل تاثیر دارند. با توجه به این که در مسئله دسته بندی تعداد داده آموزشی بسیار کم است تاثیر خطا در حالت A بسیار کم‌تر دیده می‌شود. اگر معیار Loss کلی را در نظر بگیریم در حالتی بهترین نتیجه کلی را داشتیم 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 وظیفه بهتر کند.