به نام خدا

درس یادگیری عمیق

تمرین سری ششم

مدرس درس:
سرکار خانم دکتر داوودآبادی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۱۰/۲۰

سوال ۱:

خروجی یک لایه از شبکه به ازای N داده به صورت زیر است (ستون‌ها D و ردیف‌ها N می‌باشند. Notation مطابق اسلاید ۵ جلسه ۲۵ است). بر روی آن batch normalization و layer normalization را محاسبه کنید. (حل سوال به صورت کتبی یا با استفاده از کد می‌تواند انجام گیرد. استفاده از توابع پایه‌ای numpy در کد مجاز است.) (۱۵ نمره)

20	17	32	42	65
13	65	96	53	21
45	63	74	38	64
23	76	40	34	26
14	66	78	49	23

پاسخ ۱:

فرمول‌های لازم برای این بخش عبارتند از:

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

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

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

برای نرمال‌سازی دسته‌ای، به ازای هر کانال باید میانگین (μ) و انحراف معیار (σ) را محاسبه کنیم. یعنی (BN)، روی تصویر در همه مکان‌ها میانگین می‌گیرد و خروجی هر فیلتر را نرمال می‌کند. چون محاسبات زمان‌گیر است، من کد مربوط به محاسبات این بخش را پیاده‌سازی کردم. این‌جا تصویر تک‌کاناله است. بنابراین کافیست میانگین و واریانس را روی همین یک کانال حساب کنیم و در انتها با استفاده از فرمول‌های بالا، عملیات نرمال‌سازی دسته‌ای یا Batch Normalization (BN) را انجام دهیم.در قطعه کد زیر، تمام مراحل گفته شده در بالا پیاده‌سازی شده است و نتیجه Batch Normalization آورده شده است.

Batch Noramlization

```
[9]  mean = x.mean(axis=0)
     var = x.var(axis=0)
     std = np.sqrt(var)
     batch_noramlization = (x - mean) / std
     np.set_printoptions(precision=2)
     print("Batch Normalization : \n", batch_noramlization)

     Batch Normalization :
      [[-0.26 -1.95 -1.32 -0.17  1.25]
      [-0.86  0.37  1.32  1.41 -0.93]
      [ 1.89  0.27  0.41 -0.75  1.2 ]
      [ 0.    0.9  -0.99 -1.32 -0.68]
      [-0.78  0.42  0.58  0.83 -0.83]]
```

-0.26	-1.95	-1.32	-0.17	1.25
-0.86	0.37	1.32	1.41	-0.93
1.89	0.27	0.41	-0.75	1.20
0.00	0.90	-0.99	-1.32	-0.68
-0.78	0.42	0.58	0.83	-0.83

برای نرمال‌سازی لایه‌ای، هر تصویر جداگانه نرمال می‌شود. یعنی روی همه کانال‌ها و همه مکان‌ها، μ و σ محاسبه می‌شود. کد و نتیجه این بخش نیز در بخش زیر آمده است.

Layer Noramlization

```
     mean = x.mean(axis=1)
     var = x.var(axis=1)
     std = np.sqrt(var)
     layer_noramlization = (x - mean) / std
     np.set_printoptions(precision=2)
     print("Layer Normalization : \n", layer_noramlization)

     Layer Normalization :
      [[-0.88 -1.08 -1.87  0.12  0.78]
      [-1.28  0.51  2.96  0.69 -1.02]
      [ 0.56  0.44  1.3  -0.09  0.74]
      [-0.7   0.87 -1.27 -0.3  -0.82]
      [-1.22  0.54  1.6   0.48 -0.94]]
```

-0.88	-1.08	-1.87	0.12	0.78
-1.28	0.51	2.96	0.69	-1.02
0.56	0.44	1.30	-0.09	0.74
-0.70	0.87	-1.27	-0.30	-0.82
-1.22	0.54	1.60	0.48	-0.94

سوال ۲:

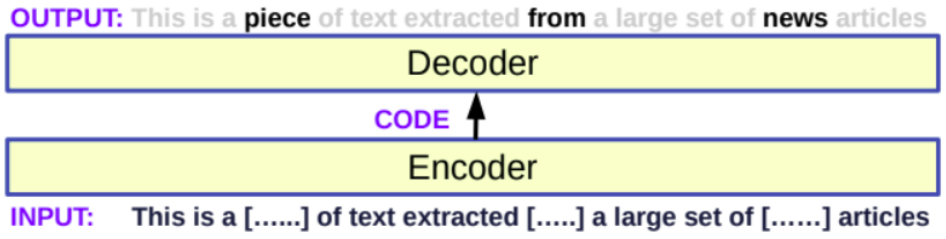
BERT یک مدل یادگیری عمیق است که برای تسک‌های پردازش زبان طبیعی مورد استفاده قرار میگیرد. در وبسایت بخش Deep Unsupervised Representation Learning را مطالعه کنید و در مورد مدل BERT خلاصه‌ای ارائه کنید. (۱۰ نمره)

پاسخ ۲:

به طور کلی ما می‌توانیم دانشی که یک شبکه عصبی از انجام task A (train شده با داده‌های annotate شده زیاد) به دست آورده است را برای یادگیری task B (annotation کم) به کار

ببریم. مشکل در اینجا، این است که ما به داشتن داده‌های annotate شده زیاد تکیه می‌کنیم اما این‌کار می‌تواند خیلی سخت یا حتی غیرممکن باشد. Deep Unsupervised Representation Learning به دنبال یادگیری مجموعه‌ای غنی از ویژگی‌های مفید از داده‌های Unsupervised است. BERT یک مدل بازنمایی زبان عمیق Unsupervised است که بازنمایی‌های متنی را از متن بدون ساختار می‌آموزد. مدل‌های context-free مانند word2vec و GloVe، word embedding را بدون در نظر گرفتن contextای که کلمات در آن ظاهر می‌شوند، یاد می‌گیرند. این یک محدودیت است زیرا بسیاری از کلمات بسته به context استفاده از آنها معانی مختلفی را بیان می‌کنند. به عنوان مثال، کلماتی مانند "bank" ممکن است در یک context مرتبط با مالی مانند "bank account" ظاهر شوند یا ممکن است برای توصیف لبه‌های رودخانه استفاده شوند. به طور متفاوت، BERT بازنمایی ها را بر اساس contextای که کلمات در آن ظاهر می‌شوند، یاد می‌گیرد. در نتیجه، BERT می‌تواند بازنمایی‌های معنایی غنی‌تری را بیاموزد که معانی مختلفی از کلمات را بسته به context آن‌ها دریافت می‌کند.

علاوه بر این، BERT با حل نوع خاصی از supervised task خود که نیازی به داده های annotate شده دستی ندارد، بهینه می‌شود. یعنی، در طول آموزش، درصدی از tokenهای انتخاب شده به‌طور تصادفی از جمله ورودی قبل از عبور از maskTransformer encoder می‌شوند. encoder جمله ورودی را به یک سری از embedding vectorها (یکی برای هر کلمه در جمله) map می‌کند. این بردارها متعاقباً از یک لایه softmax عبور می‌کنند که احتمالات را در کل واژگان محاسبه می‌کند تا محتمل‌ترین کلمات شانس بیشتری برای انتخاب داشته باشند. به عبارت دیگر، این یک پر کردن blank task است که در آن BERT قصد دارد سیگنال ورودی خراب را از context نیمه موجود بازسازی کند.



برای مدل‌های زبان، که در آن فضای راه‌حل گسسته است، وظیفه mask ورودی و بازسازی آن از زمینه یکی از دلایل اصلی موفقیت BERT است.

این task خاص masked auto-encoder نامیده می شود و نشان داده شده است که در موقعیت‌هایی که فضای احتمالی گسسته داریم به خوبی کار می‌کند.
با این وجود، مهم‌ترین ویژگی سیستمی مانند BERT این است که پس از آموزش، می‌توانیم پارامترهای BERT را در downstream tasks مختلف که داده‌های آموزشی زیادی در دسترس ندارند تنظیم کنیم و در مقایسه با آموزش به بهبود دقت قابل توجهی دست یابیم. سیستم از ابتدا و این به ویژه مهم است زیرا data-annotation یک bottleneck مهم در آموزش شبکه‌های عصبی عمیق است.

سوال ۳:

در این سوال می‌خواهیم یک مثال ساده از یادگیری ویژگی‌های بصری با استفاده از روی کرد یادگیری خودنظارتی را پیاده‌سازی کنیم. مراحل زیر را بر روی مجموعه داده CIFAR10 انجام دهید. برای حل این تمرین یک شبکه با قابلیت یادگیری بالا با استفاده از لایه‌های کانولوشنی و دیگر لایه‌های خوانده شده طراحی کنید و تمام مراحل زیر را با استفاده از آن انجام دهید. در این آزمایش، از داده‌های آموزشی هر کلاس تنها ۲۰ داده را دارای برچسب نگه می‌داریم و باقی داده‌ها را بدون برچسب استفاده خواهیم کرد. به عبارت دیگر، در مجموع ۲۰۰ داده آموزشی دارای برچسب و ۴۹۸۰۰ داده آموزشی بدون برچسب برای آموزش مدل خواهیم داشت و ۱۰۰۰۰ داده تست دارای برچسب برای ارزیابی مدل خواهیم داشت (در قسمت دوم نوتبوک تمرین، نحوه آماده سازی داده‌ها مشخص شده است.)
الف) مدل خود را تنها با استفاده از داده‌های آموزشی دارای برچسب آموزش دهید و بر روی داده‌های تست ارزیابی کنید. (۱۰ نمره)
ب) با استفاده از داده‌های آموزشی بدون برچسب، مسئله تشخیص زاویه تصویر را حل کنید. سپس، لایه انتهایی شبکه را حذف کرده و به‌جای آن یک لایه دارای ۱۰ نورون برای دسته‌بندی قرار دهید و مدل خود را با این وزن‌های اولیه و با استفاده از داده‌های آموزشی دارای برچسب آموزش دهید (با نرخ آموزش کوچکتر) و ارزیابی کنید. (۱۵ نمره)
پ) مدل خود را به گونه‌ای تغییر دهید که دارای دو خروجی باشد (یک خروجی برای دسته‌بندی زاویه و یک خروجی برای دسته‌بندی ۱۰کلاسه.) سپس، مدل خود را با تمام ۵۰۰۰۰ داده آموزشی آموزش دهید (۴۹۸۰۰ نمونه از داده‌ها دارای برچسب نیستند و بنابراین برای این داده‌ها خروجی مطلوب دسته‌بند ۱۰کلاسه را برابر با بردار صفر قرار دهید تا اثری روی تابع ضرر آن نداشته باشند.) مدل

آموزش دیده را بر روی داده‌های تست ارزیابی و با نتایج قبل مقایسه کنید. در این حالت، میزان اثر هر تابع ضرر باید به درستی تنظیم شود (با توجه به کم بودن داده‌های دارای برچسب، اثر آن‌ها در مجموع کم خواهد بود). چند ضریب مختلف برای تابع ضرر تخمین زاویه را امتحان کنید و نتایج خود را با دقت تحلیل کنید. (۱۵ نمره)

* برای تعریف یک مدل با چند خروجی می‌توانید از مدل functional در keras استفاده کنید. همچنین، برای تعیین وزن هر کدام از توابع ضرر می‌توانید از weights_loss در هنگام compile مدل استفاده کنید. برای راهنمایی بیشتر می‌توانید از این لینک کمک بگیرید (البته توجه داشته باشید که در مسئله ما، فقط ورودی دو مسئله مشترک نیست بلکه بخش عمده شبکه CNN برای دو مسئله مشترک است).

پاسخ ۳:

در ابتدا برای ساده‌سازی مسئله، x_test و x_train را نرمالیزه می‌کنیم.

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

سپس hyperparameterهای خود را تعیین می‌کنیم.

```
Define hyperparameters

[ ] INPUT_SHAPE = x_train.shape[1:]
    EPOCHS = 50
    BATCH_SIZE = 32
    NUM_CLASSES = 10
```

در انتها، y_train و y_test را به فرمت categorical تبدیل می‌کنیم.

```
Make y_train and y-test categorical

[ ] y_train = to_categorical(y_train, num_classes=NUM_CLASSES)
    y_test = to_categorical(y_test, num_classes=NUM_CLASSES)
```

یک مدل پایه به نام base از نوع Functional API تعریف کرده و از این مدل، در بخش‌های مختلف مسئله، استفاده می‌کنیم و لایه آخر را در هر بخش متناسب با سوال، اضافه می‌کنیم.

Define model

```
def model_maker(input_shape=INPUT_SHAPE):
  inputs = Input(shape=(input_shape))
  x = Conv2D(32, (3, 3), activation='relu')(inputs)
  x = BatchNormalization(axis=-1)(x)
  x = MaxPool2D(pool_size=(3, 3))(x)
  x = Dropout(rate=0.25)(x)

  x = Conv2D(64, (3, 3), activation='relu')(x)
  x = BatchNormalization(axis=-1)(x)
  x = MaxPool2D(pool_size=(2, 2))(x)
  x = Dropout(rate=0.25)(x)

  x = Conv2D(128, (3, 3), activation='relu')(x)
  x = BatchNormalization(axis=-1)(x)
  x = MaxPool2D(pool_size=(2, 2))(x)
  x = Dropout(rate=0.25)(x)

  x = Flatten()(x)
  x = Dense(1024, activation='relu')(x)
  x = BatchNormalization()(x)
  output = Dropout(rate=0.5)(x)

  model = Model(inputs=inputs, outputs=output, name='base')
  return model
```

الف) برای این بخشر، تنها کافیست به مدل خود یک classifier (یک لایه Dense با ۱۰ نورون به تعداد کلاس‌های dataset و softmax activation function به علت چند کلاسه بودن مسئله) اضافه کنیم.

Q3.A

```
[41] base_model = model_maker()
     model_1 = Sequential()
     model_1.add(base_model)
     model_1.add(Dense(units=10, activation='softmax'))
```

با compile کردن مدل ۱ که از ترکیب مدل base و یک classifier ساخته شده است، معماری مدل مطابق شکل زیر می‌شود. قابل ذکر است که optimizer را Adam با learning rate برابر با 0.001 در نظر گرفته و loss را به علت چند کلاسه بودن، CategoricalCrossentropy می‌گذاریم و برای metrics، معیار accuracy را در نظر می‌گیریم.

Q3.A) Compile model

```
model_1.compile(
    optimizer=Adam(0.001),
    loss=CategoricalCrossentropy(),
    metrics=['accuracy'],
)
model_1.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 base (Functional)           (None, 1024)              230336

 dense_12 (Dense)            (None, 10)                10250

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

در انتها مدل خود را fit می‌کنیم.

Q3.A) Fit model

```
history = model_1.fit(
  x_train, y_train,
  batch_size=BATCH_SIZE,
  epochs=EPOCHS,
  validation_data=[x_test, y_test],
  verbose=2
)
```

پس از ۵۰ epoch، دقت روی داده‌های train به 0.82، روی داده‌های validation به 0.14، loss بر داده‌های train به 0.52 و روی داده‌های validation به 4.34 می‌رسد.

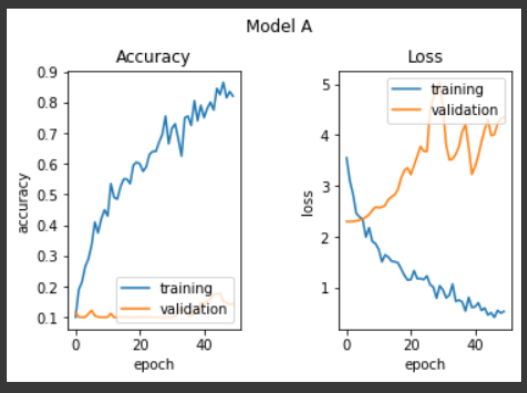
```
Epoch 46/50
7/7 - 5s - loss: 0.5009 - accuracy: 0.8250 - val_loss: 3.9919 - val_accuracy: 0.1770 - 5s/epoch - 660ms/step
Epoch 47/50
7/7 - 5s - loss: 0.4082 - accuracy: 0.8650 - val_loss: 4.0128 - val_accuracy: 0.1534 - 5s/epoch - 673ms/step
Epoch 48/50
7/7 - 5s - loss: 0.5441 - accuracy: 0.8150 - val_loss: 4.2268 - val_accuracy: 0.1475 - 5s/epoch - 689ms/step
Epoch 49/50
7/7 - 5s - loss: 0.4925 - accuracy: 0.8350 - val_loss: 4.3275 - val_accuracy: 0.1414 - 5s/epoch - 695ms/step
Epoch 50/50
7/7 - 5s - loss: 0.5252 - accuracy: 0.8200 - val_loss: 4.3483 - val_accuracy: 0.1438 - 5s/epoch - 663ms/step
```

با plot کردن loss و accuracy این مدل، نتایج زیر حاصل می‌شود. همانطور که مشخص است، مدل دچار overfitting شده است و این به این علت است که تعداد داده‌های برچسب خورده و مورد

استفاده در آموزش مدل، ۲۰۰ تا بود و باعث شد مدل آن‌ها را حفظ کند؛ اما بر داده‌های validation که تعداد زیادی (۴۹۸۰۰) داشت، عملکرد ضعیفی دارد. زیرا مدل به علت overfit شدن، قابلیت generalization ندارد.

Q3.A) Plot results

```
plot(history, "Model A")
```



ب) تصاویر برچسب نخورده را به صورت رندوم بین (۰، ۹۰، ۱۸۰ و ۲۷۰) درجه چرخش داده و label آن را به صورت one-hot در می‌آوریم.

Q3.B

```
[78] x_train_rotated = np.zeros_like(x_unlabeld)
     y_train_rotated = np.zeros((x_unlabeld.shape[0], 4))
```

```
labels = [0, 1, 2, 3]

for i in range(x_train_rotated.shape[0]):
  label = random.choices(labels, weights=[1, 3, 3, 3], k=1)[0]
  if (label == 0):
    rotated_image = x_unlabeld[i]
  elif (label == 1):
    rotated_image = np.rot90(x_unlabeld[i])
  elif (label == 2):
    rotated_image = np.rot90(np.rot90(x_unlabeld[i]))
  else:
    rotated_image = np.rot90(np.rot90(np.rot90(x_unlabeld[i])))
  x_train_rotated[i] = rotated_image
  y_train_rotated[i] = to_categorical(label, num_classes=4)
```

سپس مدل خود را به صورت زیر تعریف می‌کنیم. برای این بخش، چون مسئله ۴ کلاسه است، در لایه آخر ۴ نورون گذاشته و تابع فعال‌سازی softmax را استفاده می‌کنیم.

```
base_model = model_maker()
model_test = Sequential()
model_test.add(base_model)
model_test.add(Dense(units=4, activation='softmax'))
```

مدل test خود را که از وزن‌های آن در حل مسئله می‌خواهیم استفاده کنیم را با همان optimizer، loss و metrics کامپایل می‌کنیم.

Q3.B) Compile test model

```
model_test.compile(
  optimizer=Adam(0.001),
  loss=CategoricalCrossentropy(),
  metrics=['accuracy'],
)
model_1.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 base (Functional)           (None, 1024)              230336

 dense_12 (Dense)            (None, 10)                10250

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

مدل test خود را با در نظر گرفتن ۲۰ درصد داده‌ها به عنوان validation، fit می‌کنیم.

Q3.B) Fit model

```
history = model_test.fit(
  x_train_rotated, y_train_rotated,
  batch_size=BATCH_SIZE,
  epochs=EPOCHS,
  validation_split=0.2,
)
```

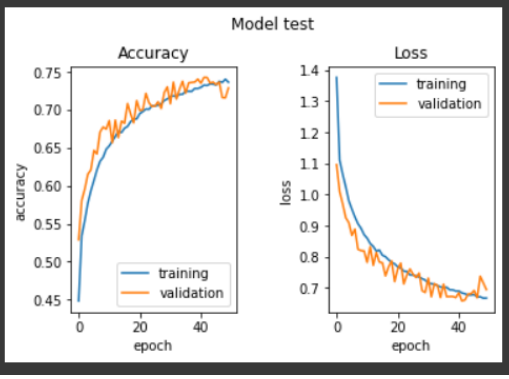
پس از ۵۰ epoch، دقت روی داده‌های train به 0.74، روی داده‌های validation به 0.73، loss بر داده‌های train به 0.66 و روی داده‌های validation به 0.69 می‌رسد.

```
Epoch 46/50
1245/1245 [==============================] - 7s 5ms/step - loss: 0.6785 - accuracy: 0.7324 - val_loss: 0.6917 - val_accuracy: 0.7329
Epoch 47/50
1245/1245 [==============================] - 7s 6ms/step - loss: 0.6707 - accuracy: 0.7371 - val_loss: 0.6682 - val_accuracy: 0.7362
Epoch 48/50
1245/1245 [==============================] - 7s 6ms/step - loss: 0.6713 - accuracy: 0.7358 - val_loss: 0.7374 - val_accuracy: 0.7159
Epoch 49/50
1245/1245 [==============================] - 6s 5ms/step - loss: 0.6668 - accuracy: 0.7400 - val_loss: 0.7162 - val_accuracy: 0.7156
Epoch 50/50
1245/1245 [==============================] - 7s 5ms/step - loss: 0.6672 - accuracy: 0.7360 - val_loss: 0.6947 - val_accuracy: 0.7285
```

با plot کردن مقادیر loss و accuracy مدل تست خود، نتایج زیر به دست می‌آید که همانطور که مشخص است، در اینجا دیگر با چالش overfitting روبرو نیستیم؛ چرا که دیگر تعداد محدود داده آموزشی نداریم.

Q3.B) Plot results

```
plot(history, "Model test")
```



مدل جدید خود را با استفاده از وزن‌های به دست آمده از model قبلی که بر روی تصاویر برچسب نخورده بود، آموزش می‌دهیم. هدف این بخش، یادگیری self-supervised است؛ بنابراین از وزن‌های به‌دست آمده از مدل test، برای Downstream Task دسته‌بندی استفاده می‌کنیم. به این منظور، مدل از پیش‌آموخته قبلی را بدون لایه آخر (classifier) برداشته و یک لایه Dense ۱۰ نورونه به منظور دسته‌بندی ۱۰ کلاسه می‌گذاریم.

```
[23] model_2 = Sequential()
     model_2.add(model_test.layers[0])
     model_2.add(Dense(units=10, activation='softmax'))
```

در این بخش، learning rate را کاهش داده و مدل را compile می‌کنیم. معماری مدل، در بخش زیر آمده است.

Q3.B) Compile new model

```
model_2.compile(
  optimizer=Adam(learning_rate=0.0001),
  loss=CategoricalCrossentropy(),
  metrics=['accuracy'],
)
model_2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 base (Functional)           (None, 1024)              230336

 dense_5 (Dense)             (None, 10)                10250

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

مدل جدید را fit می‌کنیم.

Q3.B) Fit new model

```
history = model_2.fit(
  x_train, y_train,
  batch_size=BATCH_SIZE,
  epochs=EPOCHS,
  validation_data=[x_test, y_test],
)
```

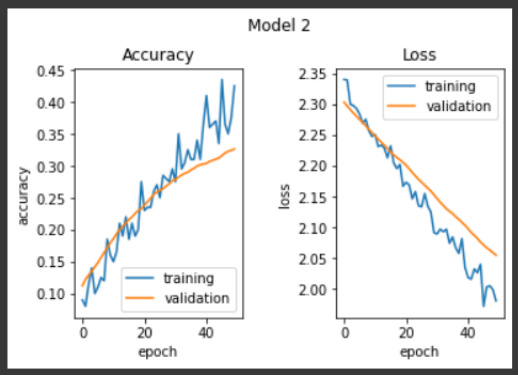
پس از ۵۰ epoch، دقت روی داده‌های train به 0.43، روی داده‌های validation به 0.33، loss بر داده‌های train به 1.9 و روی داده‌های validation به 2 می‌رسد.

```
Epoch 46/50
7/7 [==============================] - 1s 219ms/step - loss: 1.9712 - accuracy: 0.4350 - val_loss: 2.0710 - val_accuracy: 0.3160
Epoch 47/50
7/7 [==============================] - 1s 154ms/step - loss: 2.0026 - accuracy: 0.3650 - val_loss: 2.0663 - val_accuracy: 0.3200
Epoch 48/50
7/7 [==============================] - 1s 219ms/step - loss: 2.0045 - accuracy: 0.3500 - val_loss: 2.0626 - val_accuracy: 0.3225
Epoch 49/50
7/7 [==============================] - 1s 148ms/step - loss: 1.9983 - accuracy: 0.3750 - val_loss: 2.0583 - val_accuracy: 0.3242
Epoch 50/50
7/7 [==============================] - 1s 219ms/step - loss: 1.9801 - accuracy: 0.4250 - val_loss: 2.0544 - val_accuracy: 0.3263
```

loss و accuracy را برای مدل ۲ plot می‌کنیم. همانطور که مشخص است، مشکل overfitting که در بخش الف وجود داشت، حل شد. در این بخش، تاثیر آموزش self-supervised و انتقال دانش آموخته شده از تسک تشخیص زاویه چرخش به Classification Downstream Task مشاهده کرد. دقت روی داده‌های validation در بخش الف، نهایتا به 0.14 رسید، در حالیکه در این بخش به 0.33 رسید و قابلیت generalization آن بیشتر شد. با توجه به روند آموزش، مشخص است اگر تعداد epoch را بیشتر می‌گذاشتیم، دقت بیشتر هم افزایش می‌یافت.

Q3.B) Plot results new model

```
plot(history, "Model 2")
```



پ) ساختار مدل دو خروجی خود را مطابق شکل زیر تعریف می‌کنیم.

```
def final_model():
  base_model = Sequential([model_maker()])
  classifier = Dense(10, activation='softmax', name='classifier')(base_model.outputs[0])
  pretrained = Dense(4, activation='softmax', name='pretrained')(base_model.outputs[0])
  end_model = Model(inputs=base_model.inputs, outputs=[classifier, pretrained])
  return end_model
```

سپس history مربوط به مدل دو خروجی خود را نمایش می‌دهیم.

Q3.C) History of two outputs model

```
final_model().summary()
```

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

 base (Functional)           (None, 1024)              230336      ['base_input[0][0]']

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

 pretrained (Dense)          (None, 4)                 4100        ['base[0][0]']

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

طبق صورت سوال، داده‌ها را طوری تنظیم می‌کنیم که دارای ۱ ورودی و ۲ label باشند.

Q3.C) Prepare data

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

x_train_two = np.concatenate((x_train, x_train_rotated), axis=0)
y_train_two = [y_train_class, y_train_rot]

print(x_train_two.shape)
print(y_train_two[0].shape)
print(y_train_two[1].shape)
```

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

طبق خواسته صورت سوال، ضرایب مختلف برای loss را در ۳ حالت و با آموزش در ۱۵ epoch بررسی می‌کنیم.

- حالت اول) در این حالت، وزن خطای خروجی مسئله دسته‌بندی ۱۰ کلاسه را ۱۰ برابر مسئله ۴ کلاسه (چرخش) قرار می‌دهیم. مقادیر مختلف برای loss و accuracy دو مسئله به شرح زیر است:

$pretrained\_accuracy$: 0.2822 $pretrained\_loss$: 4.6407

$classifier\_accuracy$: 0.0970 $classifier\_loss$: 0.0200

$val\_classifier\_accuracy$: 0.1000 $val\_classifier\_loss$: 5.1230

val_loss : 51.2300 $loss$: 4.8406

- حالت دوم) در این حالت، وزن خطای خروجی مسئله دسته‌بندی ۴ کلاسه را ۱۰ برابر مسئله ۱۰ کلاسه (classifier) قرار می‌دهیم. مقادیر مختلف برای loss و accuracy دو مسئله به شرح زیر است:

$pretrained\_accuracy$: 0.2882 $pretrained\_loss$: 4.7362

$classifier\_accuracy$: 0.0934 $classifier\_loss$: 0.0174

$val\_classifier\_accuracy$: 0.1000 $val\_classifier\_loss$: 3.6343

val_loss : 3.6343 $loss$: 47.3793

- حالت سوم) در این حالت، وزن خطای خروجی مسئله دسته‌بندی ۴ کلاسه را برابر مسئله ۱۰ کلاسه (classifier) قرار می‌دهیم. مقادیر مختلف برای loss و accuracy دو مسئله به شرح زیر است:

$pretrained\_accuracy$: 0.2873 $pretrained\_loss$: 4.9415

$classifier\_accuracy$: 0.1089 $classifier\_loss$: 0.0205

$val\_classifier\_accuracy$: 0.1000 $val\_classifier\_loss$: 3.8603

val_loss : 3.8603 $loss$: 4.9620

از لحاظ عملکرد تسک چرخش، حالت سوم از حالت اول، و حالت دوم از هر دو بهتر است. از لحاظ عملکرد تسک classification، حالت اول از حالت سوم، و حالت سوم از حالت دوم بهتر است. از لحاظ عملکرد کلی نیز حالت سوم از حالت دوم، و حالت دوم از حالت اول بهتر است.

وزن‌هایی که بر روی توابع ضرر دو وظیفه در نظر می‌گیریم بر روی عملکرد مدل تاثیر دارند. با توجه به این که در مسئله دسته‌بندی تعداد داده آموزشی بسیار کم است تاثیر خطا در حالت ۱ 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.

لینک استفاده شده در حل این سوال:

https://github.com/ali-sedaghi/IUST-DL-Assignments/blob/main/A13/Solutions/A13-Solutions-Q4.ipynb

سوال ۴:

در نوتبوک Q4 ،ابتدا یک مدل CNN روی دیتاست cifar10 آموزش داده‌ایم. پیش‌پردازش‌های لازم انجام شده است و می‌بینید که با یک مدل نسبتا ساده به دقت معقولی رسیده‌ایم.

الف) با استفاده از ابزار keras tuner برای هر کدام ازهایپرپارامترهای زیر، حداقل ۳ مقداری که به نظر شما می‌تواند مناسب باشد را در نظر بگیرید و در گزارش مقادیر انتخابی را بیاورید. (۱۵ نمره)

- تعداد بلاک‌های کانولوشنی (منظور از بلاک تمام ۲ لایه با کانولوشنی و MaxPooling است که برای مدل پایه مثال ۲ بلاک داریم)
- مقدار احتمال Dropout
- مقدار learning rate برای Adam
- تعداد نورون‌های لایه Dense ماقبل آخر

ب) با این ابزار معرفی شده، بهترین مدل ممکن را با حالات مختلف بیابید و در گزارش خود هایپرپارامترهای مناسب پیدا شده را بیان کنید. تحلیل خود را از دلیل بهتر بودن این هایپرپارامترها بنویسید. (۱۰نمره)

پ) بهترین مدل را با همان batch size و تعداد epoch مدل اولیه آموزش دهید. سپس مقادیر precision و recall و f1-scoreرا برای آن محاسبه و گزارش کنید (می‌توانید از توابع آماده کتابخانه‌ها استفاده کنید) و بگویید آیا عملکرد مدل مناسب بوده است؟ دلیل اصلی استفاده از این متریک‌ها به جای accuracy چیست؟ آیا در این دیتاست استفاده از آن‌ها لازم است؟ (۱۵نمره)

ت) مفاهیم TP، TN، FP و FN را توضیح دهید و با رسم confusion matrix بگویید که مدل در کدام قسمت ضعیف عمل کرده است. (۱۰ نمره)

پاسخ ۴:

الف) در این بخش، تعداد بلاک‌های کانولوشنی را 3، 4 و 5 در نظر گرفته تا keras tuner، بهترین آن را انتخاب کند. برای احتمال Dropout نیز از بین مقادیر 0، 0.05، 0.1، 0.15، 0.2، 0.25، 0.3، 0.35، 0.4، 0.45 و 0.5 بهترین حالت را برمی‌گزینند. مقدار learning rate بهینه‌ساز را نیز بین مقادیر 10^{-2} و 10^{-4} در نظر می‌گیریم. تعداد نورون‌های لایه Dense ماقبل لایه آخر نیز بین 30، 40، 50، 60، 70، 80، 90 و 100 انتخاب می‌شوند.

```
def build_model(hp):
  inputs = Input(shape = (32, 32, 3))
  x = inputs

  for i in range(hp.Int('conv_blocks',min_value = 3, max_value = 5, default=3)):
    filters = hp.Int('filters_' + str(i),min_value = 32,max_value = 288, step=128)

    for _ in range(2):
      x = Conv2D(filters, kernel_size=(3, 3), padding= 'same')(x)
      x = BatchNormalization()(x)
      x = ReLU()(x)

    if hp.Choice('pooling_' + str(i), ['avg', 'max']) == 'max':
        x = MaxPool2D()(x)
    else:
        x = AvgPool2D()(x)
    x = Dropout(hp.Choice('rate', values=[0.1, 0.3, 0.5]))(x)

  x = GlobalAvgPool2D()(x)
  x = Dense(hp.Int('Dense units',min_value = 40, max_value = 100, step=30, default=50), activation='relu')(x)
  outputs = Dense(10, activation= 'softmax')(x)

  model = Model(inputs, outputs)
  model.compile(optimizer= Adam(hp.Float('learning_rate', min_value = 1e-4, max_value =1e-2, sampling='log')),
                loss= CategoricalCrossentropy(), metrics = ['accuracy'])

  return model
```

ب) در ابتدا الگوریتم serch خود را مشخص می‌کنیم. برای این کار، از بین Baysian Optimization، Hyperband و Random Search یکی را باید انتخاب کنیم. در این سوال، من از الگوریتم Hyperband استفاده کردم؛ زیرا منابع سخت‌افزاری و زمانی محدودی داشتیم و نمی‌خواستیم زمان زیادی را صرف آموزش مدل‌های معیوب کنیم. تابع tuner پارامترهایی مانند hypermodel، یک متریک هدف برای ارزیابی مدل، max_epochs برای آموزش، تعداد hyperband_iterations برای هر مدل، و یک فهرست برای ذخیره گزارش‌های آموزشی و project_name را می‌گیرد.

```
[19] tuner = kt.Hyperband(
       hypermodel=build_model,
       objective='val_accuracy',
       max_epochs=30,
       directory='./tuner_results',
     )
```

سپس می‌توانیم از tuner خود برای جستجوی hyperparameterهای بهینه برای مدل در فضای جستجوی تعریف شده استفاده کنیم. این روش مشابه fit کردن یک مدل با استفاده از Keras است.

```
[21] tuner.search(
       img_train, label_train,
       epochs=30,
       batch_size=32,
       validation_data= (img_test,label_test),
       callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])

     Trial 90 Complete [00h 04m 35s]
     val_accuracy: 0.8371000289916992

     Best val_accuracy So Far: 0.8403000235557556
     Total elapsed time: 01h 29m 53s
```

بهترین هایپرپارامترها برای مدل در فضای جستجوی تعریف شده را می‌توان با استفاده از روش get_best_hyperparameters و بهترین مدل با استفاده از تابع get_best_models به دست آورد.

```
[22] best_hps= tuner.get_best_hyperparameters(1)[0]
     best_model = tuner.get_best_models(1)[0]
```

در بخش زیر، هایپرپارامترهای انتخابی توسط keras tuner مشاهده می‌شوند.

```
nblocks = best_hps.get('conv_blocks')
print(f'Number of conv blocks: {nblocks}')
for hyparam in [f'filters_{i}' for i in range(nblocks)] + [f'pooling_{i}' for i in range(nblocks)] + ['rate'] +  ['Dense units'] + ['learning_rate']:
  print(f'{hyparam}: {best_hps.get(hyparam)}')

Number of conv blocks: 4
filters_0: 64
filters_1: 224
filters_2: 192
filters_3: 192
pooling_0: avg
pooling_1: avg
pooling_2: avg
pooling_3: max
rate: 0.15
Dense units: 100
learning_rate: 0.00025150107550240896
```

دلیل بهتر بودن این هایپرپارامترها این است که بالاترین دقت را روی داده‌های validation داشتند.

پ) ابتدا به توضیح مفهموم هر یک از موارد خواسته شده در سوال می‌پردازیم.

Percision: درصد نمونه‌هایی که توسط مدل به عنوان کلاس مثبت تشخیص داده شده‌اند و درست بودند. در واقع، Precision به ما می‌گوید مدل در پیش‌بینی‌هایی که به عنوان کلاس مثبت داشته چقدر دقیق بوده است. این معیار برای زمانی مناسب است که هزینه False Positive زیاد است. مثال نباید ایمیلی که اسپم نیست را اسپم پیش‌بینی کنیم. مقدار Precision، از رابطه زیر به دست می‌آید:

$$Precision = \frac{TP}{TP + FP}$$

Recall: درصد نمونه‌هایی که مثبت بوده‌اند و به درستی توسط مدل تشخیص داده شده‌اند. این معیار برای زمانی مناسب است که که هزینه False Negative زیاد است. مثال اگر یک فرد بیمار را سالم تشخیص بدهیم بسیار بد است. مقدار Recall نیز توسط رابطه زیر تعریف می‌شود.

$$Recall = \frac{TP}{TP + FN}$$

F1-score: این معیار از ترکیب Precision و Recall به دست می‌آید و PR را با یک عدد خلاصه می‌کند. این معیار زمانی مناسب است که می‌خواهیم یک Balance میان Precision و Recall برقرار کنیم. همچنین این معیار برای زمانی که که توزیع داده‌ها در کلاس‌ها نابرابر است مناسب است. فرمول این معیار، به شرح زیر است.

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

حال با استفاده از تابع classification_report، این سه مقدار را محاسبه می‌کنیم.

```
eval_result = model.evaluate(img_test, label_test)
print(f"test loss: {eval_result[0]}, test accuracy: {eval_result[1]}")
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.8058 - accuracy: 0.8157
test loss: 0.8058450222015381, test accuracy: 0.8156999945640564
```

```
y_pred = np.argmax(model.predict(img_test), axis=1)
y_pred = tf.keras.utils.to_categorical(y_pred, num_classes)
print(classification_report(label_test, y_pred))
```

```
313/313 [==============================] - 2s 5ms/step
              precision    recall  f1-score   support

           0       0.81      0.88      0.85      1000
           1       0.92      0.93      0.93      1000
           2       0.82      0.73      0.77      1000
           3       0.82      0.53      0.64      1000
           4       0.71      0.89      0.79      1000
           5       0.88      0.64      0.74      1000
           6       0.69      0.95      0.80      1000
           7       0.96      0.74      0.84      1000
           8       0.79      0.96      0.87      1000
           9       0.88      0.91      0.89      1000

   micro avg       0.82      0.82      0.82     10000
   macro avg       0.83      0.82      0.81     10000
weighted avg       0.83      0.82      0.81     10000
 samples avg       0.82      0.82      0.82     10000
```

تمامی معیارها در اینجا نشان دهنده این هستند که مدل به خوبی عمل کرده است. در مورد اینکه هر معیار در چه زمانی کاربرد دارد نیز در ابتدای این بخش توضیح داده شد. در این سوال، چون داده‌ها balance هستند و هزینه False Negative یا False Positive زیاد نیست، همان معیار accuracy مناسب است؛ بنابراین نیازی به استفاده از معیارهای دیگر نیست.

ت) ابتدا به توضیح هر یک از موارد TP، TF، FP و FN می‌پردازیم.

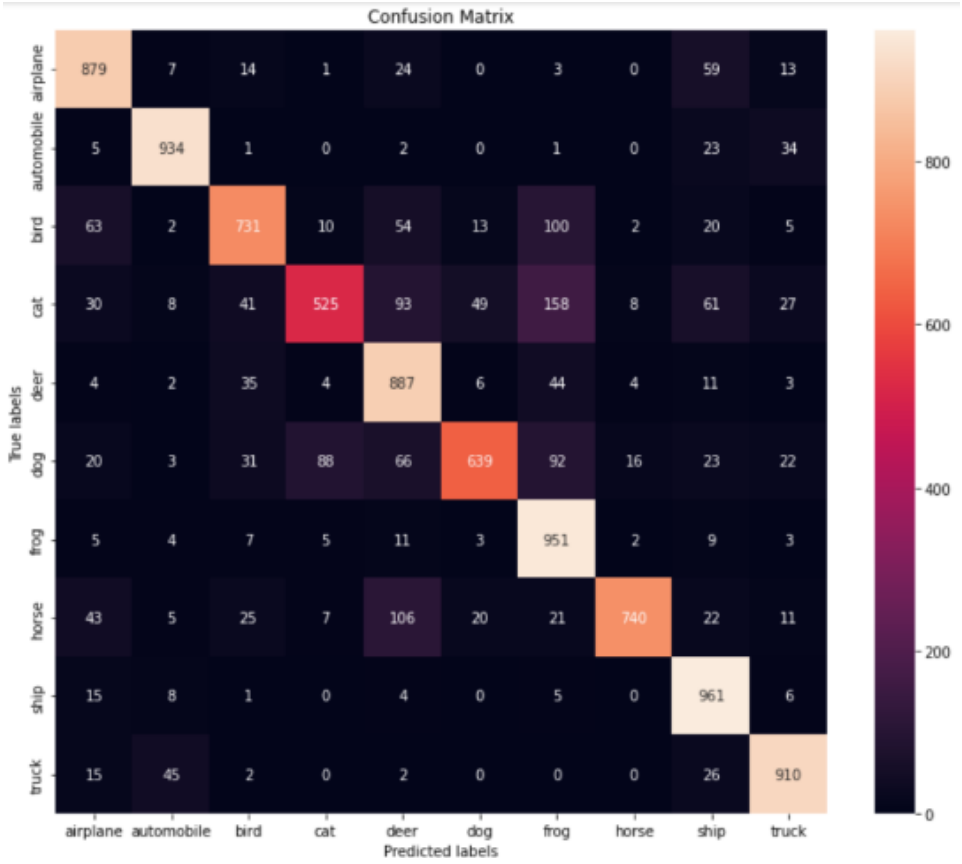
TP مخفف True Positive است و تعداد نمونه‌های مثبت که درست دسته‌بندی شده‌اند را مشخص می‌کند.

TN مخفف True Negative است و تعداد نمونه‌های منفی که درست دسته‌بندی شده‌اند را مشخص می‌کند.

FP مخفف False Positive است و تعداد نمونه‌های منفی که به اشتباه دسته‌بندی شده‌اند را نشان می‌دهد.

FN معادل False Negative است و بیانگر تعداد نمونه‌های منفی که اشتباه دسته‌بندی شده‌اند، می‌باشد.

ماتریس Confusion، یک ماتریس N*N است که N تعداد کلاس‌ها می‌باشد. این ماتریس نشان می‌دهد که کلاس سطر i چقدر به عنوان کلاس j در نظر گرفته شده است و هر چه این میزان تشخیص بیشتر باشد، رنگ آن روشن‌تر است. برای مثال در تصویر ماتریس confusion زیر، قطر آن که کلاس‌های مشابهی را دارا هستند (مثلا سطر ۱ و ستون ۱ هر دو محتوی کلاس airplane هستند)، بیشترین میزان تشخیص در نظر گرفته شده است که منطقی است. به طور کلی، با نگاه کردن به ماتریس زیر متوجه می‌شویم که مدل تقریبا خیلی خوب عمل کرده است و تعداد اشتباهات آن به نسبت کم است. می‌توان گفت بیشترین جایی که مدل دچار اشتباه شده است، در سطر ۴ و ستون ۷ بوده است که تعدادی گربه را به عنوان قورباغه تشخیص داد است. در سطر ۸ و ستون ۵ که مربوط به کلاس‌های اسب و گوزن می‌باشد نیز مدل اشتباها تعدادی اسب را گوزن تشخیص داده است. مورد بعدی نیز سطر ۳ ستون ۷ است که تقریبا تعداد زیادی پرنده به عنوان قورباغه تشخیص داده شده‌اند. با توجه به ماتریس confusion، مشخص است مدل در تشخیص گربه، سگ، پرنده و اسب به نسبت ضعیف عمل کرده است و تعداد نسبتا زیادی از آن‌ها را اشتباها به عنوان کلاس‌های دیگر تشخیص داده است.



لینک‌های مورد استفاده شده در حل این سوال:

https://blog.paperspace.com/hyperparameter-optimization-with\-keras-tuner/