به نام خدا

درس یادگیری عمیق

تمرین سری اول

مدرس درس:
سرکار خانم دکتر داوودآبادی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۸/۰۳

سوال ۱:

- الف) مجموعه دادگان آموزشی زیر را در نظر بگیرید، توجه کنید که این سوال یک مساله ۲ کلاسه است.

Data	Class
aa	0
ab	0
ba	0
bb	1

مقادیر احتمال‌های زیر را به صورت حاصل ضرب کسرها به دست آورید. در محاسبه احتمال شرطی از هموارسازی لاپلاس (Smoothing Laplace)با ضریب آلفای ۱ استفاده کنید.

$$P(1),\ P(0),\ P(a|0),\ P(b|0),\ P(a|1),\ P(b|1)$$

- ب) حال سعی کنید داده‌های زیر را دسته‌بندی کنید و کلاس مربوطه آن ها را به دست آورید.

Data	Class
aabaa	0
b	0
bba	0
bbbb	1

پاسخ ۱:

- الف)

$$P(1) = \frac{1}{4} = 0.25 \quad , \quad P(0) = \frac{3}{4} = 0.75$$

فرمول هموارسازی لاپلاس:

$$P(a|X) = \frac{X+\alpha}{N+\alpha*k}$$

در فرمول بالا، آلفا نشان‌دهنده ضریب، k تعداد feature های دیتا و N نشان‌دهنده تعداد نتایج با کلاس X هستند. حال با توجه به این فرمول، به محاسبه احتمال‌های شرطی داده شده می‌پردازیم:

$$P(a|0) = \frac{4+1}{6+(2*1)} = \frac{5}{8} = 0.625 \quad , \quad P(b|0) = \frac{2+1}{6+(2*1)} = \frac{3}{8} = 0.375$$
$$P(a|1) = \frac{0+1}{2+(2*1)} = \frac{1}{4} = 0.25 \quad , \quad P(b|1) = \frac{2+1}{2+(2*1)} = \frac{3}{4} = 0.75$$

- ب) حال با توجه به نتایج به دست آمده از بخش الف، احتمال اینکه هر یک از داده‌های داده شده مربوط به کدام کلاس باشند را به دست می‌آوریم و کلاسی که به ازای آن داده احتمالش بیشتر بود، انتخاب می‌شود.

۱. aabaa

$0 : P(0) * P(a|0) * P(a|0) * P(b|0) * P(a|0) * P(a|0) = 0.75 * 0.625 * 0.625 * 0.375 * 0.625 * 0.625 = 0.043$

$1 : P(1) * P(a|1) * P(a|1) * P(b|1) * P(a|1) * P(a|1) = 0.25 * 0.25 * 0.25 * 0.75 * 0.25 * 0.25 = 0.0007$

با توجه به نتایج به دست‌آمده، نتیجه می‌گیریم این داده به کلاس ۰ تعلق دارد.

۲. b

$0 : P(0) * P(b|0) = 0.75 * 0.375 = 0.28$

$1 : P(1) * P(b|1) = 0.25 * 0.75 = 0.19$

با توجه به نتایج به دست‌آمده، نتیجه می‌گیریم این داده به کلاس ۰ تعلق دارد.

۳. bba

$0 : P(0) * P(b|0) * P(b|0) * P(a|0) = 0.75 * 0.375 * 0.375 * 0.625 = 0.066$

$1 : P(1) * P(b|1) * P(b|1) * P(a|1) = 0.25 * 0.75 * 0.75 * 0.25 = 0.035$

با توجه به نتایج به دست‌آمده، نتیجه می‌گیریم این داده به کلاس ۰ تعلق دارد.

۴. bbbb

$0 : P(0) * P(b|0) * P(b|0) * P(b|0) * P(b|0) = 0.75 * 0.375 * 0.375 * 0.375 * 0.375 = 0.015$

$1 : P(1) * P(b|1) * P(b|1) * P(b|1) * P(b|1) = 0.25 * 0.75 * 0.75 * 0.75 * 0.75 = 0.079$

با توجه به نتایج به دست‌آمده، نتیجه می‌گیریم این داده به کلاس ۱ تعلق دارد.

سوال ۳:

حال به سراغ نوتبوک Keras.ipynb رفته و بخشهای خواسته شده را اجرا و پیاده کنید.

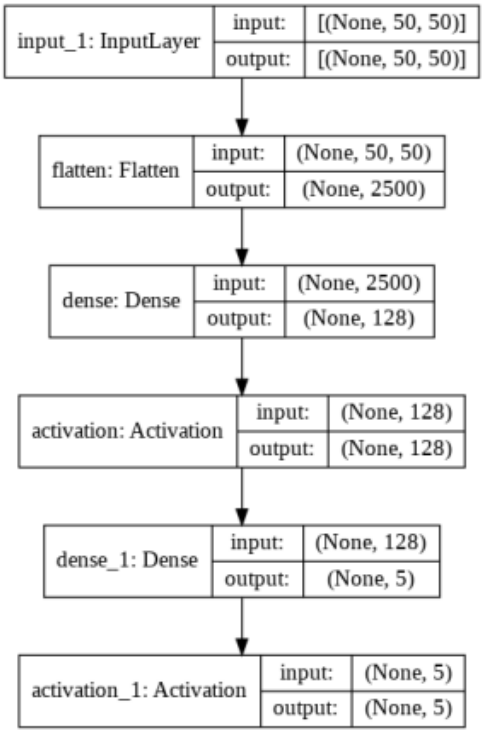
- الف) آشنایی با keras:
در این بخش شما با ۳ دیتاست MNIST، CIFAR-10 و FER-2013 آشنا میشوید، ابتدا سلول‌های مربوط به هر بخش را اجرا کنید.

۱. در گزارش در مورد هر دیتاست توضیح مختصری ارائه دهید.
۲. دلیل استفاده از categorical_to برای تبدیل برچسب‌های دیتاست‌های MNISTو CIFAR-10را بیان کنید.
۳. ابعاد x_train و y_train در هر دیتاست نمایانگر چیست؟
۴. برای دیتاست FER-2013 از ImageDataGenerator استفاده شده است. در مورد دلیل استفاده از آن توضیح دهید.

- ب) پیاده سازی مدل: پیاده سازی مدل در Keras میتواند به دو صورت Sequential و API Functional انجام شود:

۱. مدل اول: در این قسمت مدل را به صورت Sequential پیاده سازی کنید. معماری مدل به شرح زیر است:
   - برای لایه ورودی ابعاد داده ۵۰*۵۰ است. سپس از یک لایه Flatten استفاده کنید.
   - برای لایه مخفی از یک لایه Dense با ۱۲۸ نورون و یک تابع فعال سازی relu استفاده کنید.
   - برای لایه خروجی از یک لایه Dense با ۵ نورون و تابع فعال سازی softmax استفاده کنید.
۲. در این قسمت مدل را با استفاده از API Functional پیاده سازی کنید. معماری مدل به شرح زیر است:
   - برای لایه ورودی ابعاد داده همان $Flattenshape_input$. استفاده کنید.
   - برای لایه مخفی از یک لایه Dense با ۱۲۸ نورون و یک تابع فعال سازی relu استفاده کنید.
   - برای لایه خروجی از یک لایه Dense که تعداد نورون های آن همان classes_num است و تابع فعال سازی softmax استفاده کنید.

سپس سلول‌های مربوط به رسم هر دو مدل را اجرا کنید. خروجی ها باید به صورت زیر باشد:



summary هر دو مدل رو نشان دهید. چه اطلاعاتی از مدل در summary وجود دارد؟

- ج) تابع بهینه ساز: از بهینه ساز SGD با نرخ یادگیری ۰.۰۱ استفاده کنید.
- د) آموزش و تست مدل: در این بخش قصد داریم برای هر یک از دیتاست‌های MNIST و FER-2013 مدلی را آموزش دهیم و عملکرد آن را بررسی کنیم. برای کامپایل کردن مدل ها از تابع compile استفاده کنید. ورودی‌های آن عبارتند از: تابع ضرر crossentropy_categorical، بهینه ساز SGD که از قبل آن را تعریف کرده‌اید و معیار ارزیابی accuracy

۱. آموزش و ارزیابی مدل اول: با استفاده از متد fit مدل را آموزش دهید. ورودی‌های آن عبارتند از: اندازه batch ۶۴، تعداد epoch ۵، ۲۰ درصد داده validation. سپس سلول مربوط به رسم پالت‌ها را اجرا کنید. میزان دقت روی داده train و داده val در طی فرایند آموزش به چه صورت تغییر کرده است؟ با استفاده از تابع evaluate میزان دقت مدل روی داده آزمون را به دست بیاورید. از تابع predict استفاده کنید و پیش‌بینی مدل روی ۳ نمونه از داده آزمون را به همراه برچسب آن نمونه چاپ کنید.

۲. آموزش و ارزیابی مدل دوم: با استفاده از متد fit مدل را آموزش دهید. ورودی‌های آن عبارتند از: تعداد epoch ۵ ،داده validation را همان داده آموزش قرار دهید. سپس سلول مربوط به رسم پالت‌ها را اجرا کنید. میزان دقت روی داده train و داده val در طی فرایند آموزش به چه صورت تغییر کرده است؟ با استفاده از تابع evaluate میزان دقت مدل روی داده آزمون را به دست بیاورید.

۳. پیش‌بینی چند نمونه: سلول مربوط به پیش‌بینی چند نمونه از داده‌های آزمون را اجرا کنید. آیا برچسب‌های پیش‌بینی شده با برچسب‌های حقیقی داده‌ها مطابقت دارند؟ در این مورد توضیح دهید.

پاسخ ۳:

- ۳ـ الف ـ ۱)

— MNIST: MNIST یک dataset از اعداد دست‌نویس بین ۰ تا ۹ می‌باشد که دارای ۶۰۰۰۰ تصویر از این ارقام با سایز ۲۸*۲۸ است. همچنین این dataset دارای ۱۰۰۰۰ نمونه برای تست نیز می‌باشد. این ارقام size-normalized شده و در وسط یک تصویر با سایز fix قرار دارند. البته این dataset در واقع زیرمجموعه‌ای از NIST dataset می‌باشد.

— CIFAR-10: این dataset دارای ۶۰۰۰۰ تصویر رنگی ۳۲*۳۲ در ۱۰ کلاس با ۶۰۰۰ تصویر در هر کلاس است. از این ۶۰۰۰۰ داده، ۵۰۰۰۰ داده مربوط به داده‌های آزمایشی و ۱۰۰۰۰ داده برای تست می‌باشد. دسته‌بندی‌های مربوط به این داده‌ها عبارتند از: airplane، automobile، bird، cat، deer، dog، frog، horse، sheep و truck. البته قابل ذکر است که کلاس‌ها کاملا exclusive هستند.

— FER-2013: این dataset شامل ۳۲۲۹۸ تصویر ۴۸*۴۸ از چهره‌ها است که در ۷ دسته قرار گرفته‌اند. از این ۳۵۸۸۷ داده، ۲۸۷۰۹ تا مربوط به نمونه‌های آموزشی و ۷۱۷۸ نمونه، برای تست می‌باشد. چهره‌ها به صورت خودکار ثبت شده‌اند، به‌طوری که کم و بیش در مرکز قرار گرفته‌اند و تقریبا همان مقدار از فضا را اشغال می‌کند. وظیفه این است که هر چهره را بر اساس احساسات نشان داده شده در حالت چهره به یکی از هفت دسته(۰: عصبانی، ۱: انزجار، ۲: ترس، ۳: خوشحال، ۴: غمگین، ۵: تعجب، ۶: خنثی) دسته‌بندی کنیم.

- ۳ـ الف ـ ۲)

با استفاده از to_categorical، از یک encoding برای عنصر کلاس هر نمونه استفاده می‌کنیم و عدد صحیح را به یک بردار با تعداد عناصر به تعداد کلاس‌ها با شاخص ۱ برای کلاس مربوطه استفاده می‌کنیم و بقیه عناصر ۰ باقی می‌مانند. پس در واقع، یک کلاس بردار integer را به یک بردار باینری تبدیل می‌کند. خروجی آن نیز یک بردار باینری می‌باشد.

- ۳ـ الف ـ ۳

در دیتاست MNISt، ابعاد x_train، به ترتیب تعداد نمونه‌ها و سایز نمونه‌ها می‌باشد. همچنین ابعاد y_train نیز نشان‌دهنده تعداد نمونه‌ها و تعداد کلاس‌ها می‌باشد که در این

دیتاست، کلاس‌ها ارقام ۰ تا ۹ هستتند.
در دیتاست CIFAR-10 نیز خروجی‌های x-train به ترتیب تعداد نمونه‌ها، ابعاد و تعداد کانال‌ها می‌باشد. y-train نیز نمایانگر تعداد نمونه‌ها و تعداد کلاس‌ها است.
در دیتاست FER-2013 نیز به طور کلی، ۲۸۷۰۹ داده train و ۷۱۷۸ داده test داریم که متعلق به ۷ کلاس هستند و ابعاد آن نیز ۴۸*۴۸ می‌باشد.

- ۳ـ الف ـ ۴)

ImageDataGenerator این امکان را به ما می‌دهد که تا زمانی که مدل ما در حال آموزش است، تصاویر خود را به صورت real-time زیاد کنیم. یعنی با استفاده از آن، هرگونه تبدیل تصادفی روی تصویر آموزشی که به مدل منتقل می‌شود را می‌توانیم اعمال کنیم. با این‌کار، علاوه بر تقویت مدل، در حافظه نیز صرفه‌جویی می‌شود. از جمله این تبدیل‌های تصادفی، می‌توان به استانداردسازی، چرخش، جابه‌جایی، تغییر روشنایی و ... اشاره کرد. در واقع مزیت اصلی این کلاس، افزایش داده‌ها در زمان real-time می‌باشد. از دیگر مزایای آن، می‌توان به حافظه کمتر آن اشاره کرد، چرا که بدون استفاده از این کلاس، همه تصاویر را به یکباره بارگذاری می‌کردیم ولی با استفاده از این روش، تصاویر را به صورت دسته‌ای بارگذاری می‌کنیم که مقدار زیادی از حافظه را ذخیره می‌کند.

- ۳ـ ب ـ ۱)

مدل sequential را با توجه به توضیحات داده شده، پیاده‌سازی می‌کنیم. کدها و سامری این بخش، به شرح زیر می‌باشد:

```
model_temp_1 = Sequential()

# Input Layer
# Write your code here
model_temp_1.add(layers.Input(shape=(50, 50)))
model_temp_1.add(layers.Flatten())

# Hidden Layer
# Write your code here
model_temp_1.add(layers.Dense(128))
model_temp_1.add(layers.Activation('relu'))

# Output Layer
# Write your code here
model_temp_1.add(layers.Dense(5))
model_temp_1.add(layers.Activation('softmax'))
```

```
model_temp_1.summary()

Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_8 (Flatten)         (None, 2500)              0

 dense_3 (Dense)             (None, 128)               320128

 activation (Activation)     (None, 128)               0

 dense_4 (Dense)             (None, 5)                 645

 activation_1 (Activation)   (None, 5)                 0

=================================================================
Total params: 320,773
Trainable params: 320,773
Non-trainable params: 0
_________________________________________________________________
```

- ۳ـ ب ـ ۲)

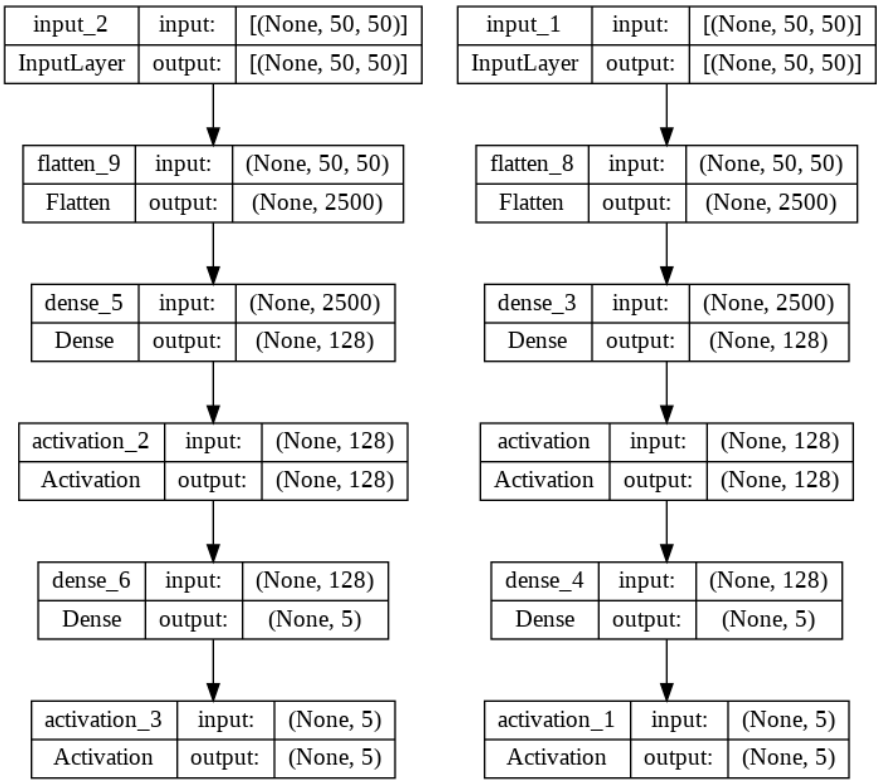
حال به پیاده‌سازی مدل با استفاده از روش خواسته شده (Functional API) می‌پردازیم. کدها و سامری این بخش نیز در ادامه موجود است:

در

```python
def model_factory(input_shape, num_classes):
    # Input Layer
    # Write your code here
    input = layers.Input(shape=input_shape)
    model_temp_2 = layers.Flatten()(input)

    # Hidden Layer
    # Write your code here
    model_temp_2 = layers.Dense(128)(model_temp_2)
    model_temp_2 = layers.Activation('relu')(model_temp_2)

    # Output Layer
    # Write your code here
    model_temp_2 = layers.Dense(num_classes)(model_temp_2)
    output = layers.Activation('softmax')(model_temp_2)

    return Model (inputs = input, outputs = output)
```

```
model_template_2.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 50, 50)]          0

 flatten_9 (Flatten)         (None, 2500)              0

 dense_5 (Dense)             (None, 128)               320128

 activation_2 (Activation)   (None, 128)               0

 dense_6 (Dense)             (None, 5)                 645

 activation_3 (Activation)   (None, 5)                 0

=================================================================
Total params: 320,773
Trainable params: 320,773
Non-trainable params: 0
```

مورد summary نیز، تصاویر سمت راست بالا، مربوط به summary هر مدل هستند.
summary به طور کلی تعداد لایه‌ها و ترتیب آن‌ها در مدل، shape خروجی هر لایه، تعداد پارامترها (وزن‌ها) در هر لایه، تعداد کل پارامترها (وزن‌ها) در مدل را نشان می‌دهد.
همچنین خروجی‌های هر دو مدل نیز، به شرح دو تصویر زیر است:



- ۳ـ ج)

با استفاده از تابع SGD، از بهینه‌ساز با learning_rate ۰.۰۱ استفاده می‌کنیم. کد این بخش نیز در تصویر زیر نمایش داده شده است:

```
[33] from tensorflow.keras.optimizers import SGD

# Write your code here
sgd_optimizer = SGD(learning_rate=0.01)
```

- ۳ـدـ۱)

با استفاده از متغیرهای اولیه داده شده، کد مربوط به این بخش را کامل می‌کنیم. عکس کد و خروجی مربوط به آن در زیر آورده شده است:

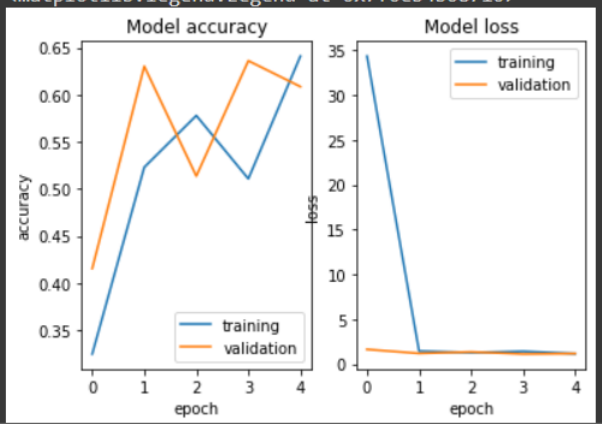
```
# Write your code here
model_mnist = model_factory(
    input_shape = (28,28), num_classes = 10
)

# Write your code here
model_mnist.compile(
    optimizer = sgd_optimizer,
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

# Write your code here
history = model_mnist.fit(
    x_train_1,
    y_train_1,
    batch_size=64,
    epochs=5,
    validation_split=0.2,
)
```

```
Epoch 1/5
750/750 [==============================] - 4s 4ms/step - loss: 34.3513 - accuracy: 0.3246 - val_loss: 1.6324 - val_accuracy: 0.4154
Epoch 2/5
750/750 [==============================] - 4s 5ms/step - loss: 1.4469 - accuracy: 0.5230 - val_loss: 1.1870 - val_accuracy: 0.6302
Epoch 3/5
750/750 [==============================] - 3s 4ms/step - loss: 1.2838 - accuracy: 0.5779 - val_loss: 1.3500 - val_accuracy: 0.5136
Epoch 4/5
750/750 [==============================] - 3s 4ms/step - loss: 1.4141 - accuracy: 0.5107 - val_loss: 1.1219 - val_accuracy: 0.6361
Epoch 5/5
750/750 [==============================] - 3s 4ms/step - loss: 1.1650 - accuracy: 0.6410 - val_loss: 1.1713 - val_accuracy: 0.6086
```

همانطور که در خروجی عکس بالا نیز مشخص است، دقت روی داده train تا epoch ۳ در حال افزایش، سپس تا epoch ۴ کاهش و بعد از آن دوباره افزایش دارد. دقت روی داده‌های validation نیز تا epoch دوم افزایش، سپس تا epoch ۳ کاهش و بعد تا epoch ۴ افزایش و بعد از آن دوباره تا epoch ۵ کاهش می‌یابد.
همچنین با اجرا کردن سلول مربوط به رسم پلات آن، تصویر زیر حاصل می‌شود:

```python
fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.plot(history.history['accuracy'])
ax1.plot(history.history['val_accuracy'])
ax1.set_title('Model accuracy')
ax1.set_ylabel('accuracy')
ax1.set_xlabel('epoch')
ax1.legend(['training', 'validation'], loc='lower right')

ax2.plot(history.history['loss'])
ax2.plot(history.history['val_loss'])
ax2.set_title('Model loss')
ax2.set_ylabel('loss')
ax2.set_xlabel('epoch')
ax2.legend(['training', 'validation'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x7f6eb4b68710>
```



تابع evaluate را نیز با توجه به داده‌هایمان تکمیل کرده و کد آن به شکل زیر در می‌آید. همانطور که در خروجی مشخص است، میزان دقت روی داده‌های test، ۰.۶ می‌باشد.

```python
# Write your code here
model_mnist.evaluate(
    x_test_1,
    y_test_1,
    batch_size = 64
)
```

```
157/157 [==============================] - 0s 2ms/step - loss: 1.1601 - accuracy: 0.6084
[1.1601399183273315, 0.6083999872207642]
```

در کد زیر نیز پیش‌بینی ۳ داده تصادفی از داده‌های تست و مقایسه آن با label مربوط به آن مشاهده می‌کنید. در اینجا حدود $\frac{2}{3}$ داده‌ها درست پیش‌بینی شده‌اند.

```
# Write your code here
y_predict_1 = np.argmax(model_mnist.predict(np.array([x_test_1[10]])))
print("y_predict_1[10] is: ", y_predict_1, "    ", "y_test_1[10] is: ", np.argmax(y_test_1[10]))
y_predict_2 = np.argmax(model_mnist.predict(np.array([x_test_1[27]])))
print("y_predict_1[27] is: ", y_predict_2, "    ", "y_test_1[27] is: ", np.argmax(y_test_1[27]))
y_predict_3 = np.argmax(model_mnist.predict(np.array([x_test_1[52]])))
print("y_predict_1[52] is: ", y_predict_3, "    ", "y_test_1[52] is: ", np.argmax(y_test_1[52]))
```

```
1/1 [==============================] - 0s 23ms/step
y_predict_1[10] is:  0      y_test_1[10] is:  0
1/1 [==============================] - 0s 18ms/step
y_predict_1[27] is:  9      y_test_1[27] is:  4
1/1 [==============================] - 0s 20ms/step
y_predict_1[52] is:  5      y_test_1[52] is:  5
```

- ۳ـ د ـ ۲)

در این بخش نیز با استفاده از اطلاعات داده شده توسط سوال، کد را کامل کرده و به همراه نتایج آن، در بخش زیر آمده است:

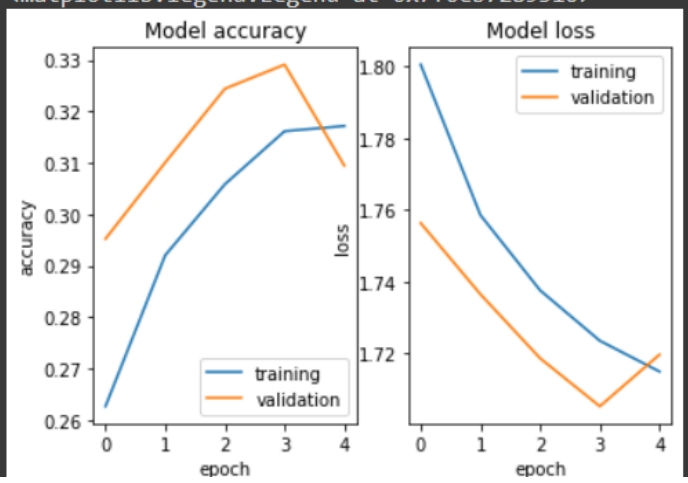
```
# Write your code here
model_fer = model_factory(
    input_shape = (48,48,1), num_classes = 7
)

# Write your code here
model_fer.compile(
    optimizer = sgd_optimizer,
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

# Write your code here
history = model_fer.fit(
    train_set,
    validation_data=train_set,
    epochs=5,
)
```

```
Epoch 1/5
449/449 [==============================] - 44s 97ms/step - loss: 1.8006 - accuracy: 0.2627 - val_loss: 1.7563 - val_accuracy: 0.2952
Epoch 2/5
449/449 [==============================] - 44s 97ms/step - loss: 1.7586 - accuracy: 0.2920 - val_loss: 1.7365 - val_accuracy: 0.3101
Epoch 3/5
449/449 [==============================] - 45s 100ms/step - loss: 1.7376 - accuracy: 0.3059 - val_loss: 1.7186 - val_accuracy: 0.3244
Epoch 4/5
449/449 [==============================] - 44s 97ms/step - loss: 1.7235 - accuracy: 0.3162 - val_loss: 1.7052 - val_accuracy: 0.3291
Epoch 5/5
449/449 [==============================] - 43s 97ms/step - loss: 1.7149 - accuracy: 0.3172 - val_loss: 1.7197 - val_accuracy: 0.3095
```

همانطور که در عکس بالا مشخص است، دقت داده‌های train در این مدل افزایش می‌یابد.
اما دقت روی داده‌های validation، تا epoch ۴ افزایش و سپس کاهش می‌یابد.
با اجرا کردن سلول مربوط به رسم پلات نیز تصویر زیر حاصل می‌شود:

```
fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.plot(history.history['accuracy'])
ax1.plot(history.history['val_accuracy'])
ax1.set_title('Model accuracy')
ax1.set_ylabel('accuracy')
ax1.set_xlabel('epoch')
ax1.legend(['training', 'validation'], loc='lower right')

ax2.plot(history.history['loss'])
ax2.plot(history.history['val_loss'])
ax2.set_title('Model loss')
ax2.set_ylabel('loss')
ax2.set_xlabel('epoch')
ax2.legend(['training', 'validation'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x7f6eb7289310>
```



با اجرا کردن تابع evaluate، مقدار دقت برای داده‌های test، تقریبا 0.32 به دست می‌آید:

```
model_fer.evaluate(
    x=test_set,
)
```

```
113/113 [==============================] - 3s 26ms/step - loss: 1.6959 - accuracy: 0.3252
[1.6958832740783691, 0.3251602053642273]
```

- ۳ـ د ـ۳)

با اجرا کردن تابع evaluate، نتایج زیر به دست می‌آیند که همانطور که در شکل نیز مشخص

است، labelهای واقعی با مقادیر پیش‌بینی شده کاملا متفاوت است. دلیل این اتفاق، پدیده overfitting است. چرا که در اینجا از داده‌های train برای validation استفاده کردیم و مدل روی داده‌های train، overfit شده است و به همین دلیل دقتش برای داده‌های تست جدید، کم می‌شود.

```
prediction = model_fer.predict(test_set)
# convert prediction to labels
labels = prediction.argmax(axis=-1)

# labels list
label_list = list(test_set.class_indices.keys())

plt.figure()
f, axarr = plt.subplots(1,3)

# use the created array to output your multiple images.
axarr[0].imshow(np.squeeze(test_set[0][0][5]))
label_1 = label_list[test_set[0][1][5].argmax(axis=-1)]
axarr[1].imshow(np.squeeze(test_set[0][0][29]))
label_2 = label_list[test_set[0][1][29].argmax(axis=-1)]
axarr[2].imshow(np.squeeze(test_set[0][0][10]))
label_3 = label_list[test_set[0][1][10].argmax(axis=-1)]

# Show predicted label for each image
axarr[0].set_title(f'predicted: {label_list[labels[5]]} \n actual: {label_1}')
axarr[1].set_title(f'label: {label_list[labels[29]]} \n actual: {label_2}')
axarr[2].set_title(f'label: {label_list[labels[10]]} \n actual: {label_3}')
```

```
113/113 [==============================] - 3s 23ms/step
Text(0.5, 1.0, 'label: happy \n actual: fear')
<Figure size 432x288 with 0 Axes>
```