به نام خدا

تمرین هشتم یادگیری عمیق

غزل زمانی‌نژاد

۹۷۵۲۲۱۶۶

1. **الف) dropout یکی از روش های منظم سازی است و برای جلوگیری از overfit شدن شبکه استفاده می شود. باعث می شود اگر در یک شبکه بعضی ویژگی ها حذف شد، مدل بتواند با سایر ویژگی ها تصمیم گیری کند. با استفاده کردن از dropout در لایه های میانی، یک ensemble بزرگ آموزش میبیند که دارای پارامترهای مشترک هستند. در شرایطی که مدل به سمت overfit شدن میرود، یعنی دقت آموزش و دقت اعتبارسنجی اختلاف چشمگیری دارند، خوب است از آن استفاده کنیم. در مقاله ای برای لایه کاملا متصل توصیه میشود پارامتر نگهداری نورون ها را با استفاده از مجموعه اعتبارسنجی tune کنیم و یا از مقدار 0.5 برای لایه های میانی استفاده کنیم. اما به نظر این مقدار برای لایه convolution نمیتواند مقدار مناسبی باشد و بهتر است مقدار بزرگتری داشته باشد. بدلیل اینکه در شبکه با معماری کانولوشن تعداد فیلترها در طی لایه ها معمولا زیاد میشود، میتوانیم از یک پارامتر بزرگ شروع کنیم و در طی لایه ها آن را کمی کاهش دهیم. برای لایه ورودی، باید پارامتر نگهداری مقداری نزدیک به 1، مثلا 0.8 باشد تا از بیشتر نورون های لایه ورودی برای طبقه بندی استفاده کنیم. در پایان، باید دقت داشته باشیم که پارامتر نگهداری را آنقدر کوچک نکنیم که شبکه دچار underfit شود.**

   ب) هرچه مقدار پارامتر بیشتر باشد، اثر dropout کمتر میشود. یعنی مدل به حالت قبلی خود (overfit شدن) نزدیکتر میشود. پس هرچه این مقدار بیشتر باشد شبکه بهتر الگوها را حفظ میکند و ظرفیت یادگیری آن بیشتر است و برعکس.

2. **لایه کاملا متصل: در این لایه هر نورون، به تمامی نورون های لایه قبل متصل است و به همین دلیل در تمامی خروجی های لایه های بعدی تاثیرگذار است. هر نورون از نورون بعدی مستقل است. این لایه الگوهای سراسری را در فضای ویژگی ورودی می آموزد.**

لایه متصل محلی: هر نورون تنها بخش کوچکی از لایه قبل را میبیند زیرا در داده هایی مثل تصویر، میتوانیم لبه ها، بافت ها و شکل ها را با استفاده از شدت پیکسل ها در ناحیه کوچکی از تصویر تشخیص دهیم. در این نوع لایه هر نورون فیلتر خودش را دارد و تعداد پارامترهای آن نسبت به کانولوشن بسیار بیشتر است که در صورت نداشتن داده کافی، همین امر میتواند موجب overfit شدن شبکه شود. با کمک این لایه شبکه ویژگی های مختلفی برای نواحی مختلف ورودی می آموزد. در یک شبکه عمیق، receptive field، یعنی اثر یک نورون در نورون های لایه بعدی زیاد است. این کار به شبکه اجازه می‌دهد تا بتواند ویژگی‌های پیچیده را به صورت سلسله‌مراتبی بیاموزد.

ویژگی ها:

- **دارای اتصالات تنک: یعنی با بخش کوچکی از تصویر ویژگی های خوبی استخراج میکند.**
- **توانایی کار با ورودی های با ابعاد مختلف: در این نوع لایه برخلاف لایه کاملا متصل، نیازی نیست ورودی ها ابعاد یکسان داشته باشند.**

لایه کانولوشنی: مشابه لایه متصل محلی است. با این تفاوت که از ویژگی parameter sharing استفاده میکند. یعنی برای تشخیص یک ویژگی، از یک فیلتر (یک سری وزن) برای تمامی نورون های لایه قبل استفاده میشود. دارای بازنمایی های هم تغییر نسبت به جا به جایی است.

کاربرد این لایه ها:

در داده هایی که ویژگی آنها به یکدیگر مربوط نیست، از لایه کاملا متصل استفاده میشود. مثلا در مسئله تخمین قیمت خانه، ویژگی ها از جمله موقعیت جغرافیایی، اندازه، قدمت، تعداد اتاق ها و ... بر یکدیگر تاثیر ندارند. پس از لایه کاملا متصل استفاده میکنیم تا در هر لایه، یک نورون بر اساس تمامی نورون های قبلی آموزش ببیند. همچنین در مسائلی که از لایه های دیگر همچون کانولوشن استفاده میشود، از لایه dense برای دسته بندی نهایی استفاده میشود.

در داده هایی که نواحی مجاور به هم وابسته هستند (مثل صوت و تصویر که در آن پیکسل های همسایه بهم وابستگی دارند) استفاده از کانولوشن میتواند انتخاب مناسبی باشد زیرا کانولوشن به الگوهای محلی داده دقت میکند و از آنها ویژگی استخراج میکند.

در صورتی که ویژگی های داده به یکدیگر وابسته باشند ولی نخواهیم پارامترها را به اشتراک بگذاریم، از لایه متصل محلی استفاده میکنیم. یعنی در جایی که مکان و زمان اهمیت پیدا میکند و بدنبال ویژگی های سطح بالا

هستیم. مثلا در پردازش تصویر، در لایه های پایانی شبکه از locally connected استفاده میکنیم. چون مطمئن هستیم در ناحیه ای یک تصویر خاص هست پس ویژگی خاص آن را یاد میگیریم. مثلا در تصویر زیر، در لایه های پایانی شبکه میدانیم در باکس 1 چشم قرار دارد، پس ویژگی مربوط به چشم را آموزش میبینند.

3. ابتدا سلول های 1 تا 3 را اجرا میکنیم تا دیتاستی از تصاویر دانلود شده و در درایو ذخیره شوند.

الف) برای قسمت اول، ImageDataGenerator را به شکل ساده و بدون داده افزایی پیاده سازی میکنیم تا بتوانیم دقت مدل را بدون augmentation بسنجیم. تنها از rescale استفاده میکنیم تا داده ها نرمالایز شوند. سپس با استفاده از متد flow_from_directory تصاویر را از فولدر مربوطه میخوانیم و Generator را بر روی آنها اعمال میکنیم.

```
from keras.preprocessing.image import ImageDataGenerator
batch_size = 16
shape = (150, 150)
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('train',
        target_size=shape,
        batch_size=batch_size)
validation_generator = test_datagen.flow_from_directory('test',
        target_size=shape,
        batch_size=batch_size)
```

سلول بعدی را اجرا میکنیم تا نمونه ای از داده ها را مشاهده کنیم.

ب) یک مدل دارای 4 لایه کانولوشنی با تابع فعالسازی relu و سایز کرنل 3، 4 لایه maxpooling پیاده سازی میکنیم. مطابق معماری کانولوشنی، تعداد فیلترها در طی لایه ها را افزایش میدهیم. برای لایه های classification از دو لایه Dense، اولی دارای 64 نورون با تابع فعالسازی relu، دومی دارای 5 نورون خروجی با تابع فعالسازی sigmoid، برای دسته بندی نهایی استفاده میکنیم.

```
def build_model():
  kernel = 3
  model = Sequential()
  model.add(Conv2D(16, kernel, activation='relu', input_shape=(shape[0], shape[1], 3)))
  model.add(MaxPooling2D(pool_size=(2, 2)))

  model.add(Conv2D(32, kernel, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))

  model.add(Conv2D(64, kernel, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))

  model.add(Conv2D(128, kernel, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))

  model.add(Flatten())
  model.add(Dense(64, activation='relu'))
  model.add(Dense(5, activation='softmax'))
  return model
```

مدل اول را با batch size = 32,Epochs = 15 را ابتدا کامپایل کرده و سپس train میکنیم. از تابع ضرر categorical cross-entropy استفاده میکنیم زیرا مسئله چند کلاسه و دارای یک برچسب (multi-class, single-label classification) است. از بهینه ساز Adam استفاده میکنیم چون سرعت همگرایی آن بیشتر است.

```
EPOCHS=15
BATCH=32
model = build_model()
loss = CategoricalCrossentropy(from_logits=False)
optimizer = tf.keras.optimizers.Adam()

model.compile(loss= loss, optimizer= optimizer, metrics=['accuracy'])
model.fit(train_generator, epochs=EPOCHS, batch_size=BATCH, validation_data=validation_generator)
```

نتیجه آموزش: این شبکه شدیدا دچار overfit شده است (میان دقت آموزش و اعتبارسنجی اختلاف چشمگیری وجود دارد). اندازه دیتاست بسیار کوچک بوده و شبکه صرفا به حفظ کردن الگوها پرداخته است و generalization ندارد. به همین دلیل روی داده اعتبارسنجی عملکرد مناسبی ندارد.

```
Epoch 15/15
14/14 [==============================] - 7s 480ms/step - loss: 0.3120 - accuracy: 0.8991 - val_loss: 2.3177 - val_accuracy: 0.3793
```

```
model.evaluate(validation_generator, batch_size=BATCH)
```

```
2/4 [==============>..............] - ETA: 0s - loss: 2.0941 - accuracy: 0.4062/usr/l
  "Palette images with Transparency expressed in bytes should be "
4/4 [==============================] - 1s 179ms/step - loss: 2.3177 - accuracy: 0.3793
[2.3176822662353516, 0.37931033968925476]
```

پ) اکنون با استفاده از روش data augmentation، سایز دیتاست را افزایش میدهیم تا مدل داده های بیشتری برای آموزش دیدن در اختیار داشته باشد و احتمال overfit کاهش یابد. در اینجا برای امتحان کردن نتیجه داده افزایی، از دو ImageDataGenerator استفاده شد.

- **در اولین روش داده افزایی از width shift range و height shift range استفاده شد. بدین معنی که برای هر تصویر موجود در دیتاست، آن را با توجه به بازه داده شده، آن را به راست یا چپ، بالا یا پایین شیفت دادیم تا داده جدید تولید کنیم.**

```
train_datagen_aug1 = ImageDataGenerator(rescale=1./255,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2)

train_generator_aug1 = train_datagen_aug1.flow_from_directory('train',
        target_size=shape,
        batch_size=batch_size)
```

در این شبکه، قدرت تعمیم بسیار بیشتر شد. اما سرعت همگرایی بسیار کاهش یافت. به همین دلیل تعداد epochها را 50 میکنیم تا اثر داده افزایی را به خوبی مشاهده کنیم. و همچنین سایز batch را 64 قرار میدهیم. اما اصل معماری شبکه به همان صورت قبل است.

```
EPOCHS2 = 50
BATCH2 = 64

model2 = build_model()

model2.compile(loss= loss, optimizer= optimizer, metrics=['accuracy'])
model2.fit(train_generator_aug1, epochs=EPOCHS2, batch_size=BATCH2, validation_data=validation_generator)
```

نتیجه آموزش: اختلاف دقت آموزش و تست از 52 به 19 کاهش پیدا کرد. با وجود اینکه دقت آموزش کاهش یافت، اما در مسائل طبقه بندی هدف اصلی افزایش دقت داده هایی است که شبکه بر روی آنها آموزش ندیده است. که در این مدل به هدف خود رسیدیم.

```
Epoch 50/50
14/14 [==============================] - 4s 276ms/step - loss: 0.6602 - accuracy: 0.7523 - val_loss: 1.5009 - val_accuracy: 0.5690
```

```
model2.evaluate(validation_generator, batch_size=BATCH)
```

```
2/4 [==============>..............] - ETA: 0s - loss: 1.3016 - accuracy: 0.6562/usr/lo
  "Palette images with Transparency expressed in bytes should be "
4/4 [==============================] - 1s 142ms/step - loss: 1.5009 - accuracy: 0.5690
[1.5009173154830933, 0.568965494632721]
```

- در دومین روش داده افزایی سعی کردیم روش قبل را بهبود ببخشیم. برای این کار، علاوه بر شیفت عمودی و افقی، از horizontal flip هم استفاده شد. بدین معنی که هر تصویر را علاوه بر شیفت، به صورت افقی فلیپ کردیم. سپس شبکه را با دیتاست تولید شده کامپایل و train کردیم.

```
train_datagen_aug2 = ImageDataGenerator(rescale=1./255,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        horizontal_flip=True)

train_generator_aug2 = train_datagen_aug2.flow_from_directory('train',
        target_size=shape,
        batch_size=batch_size)

model3 = build_model()
model3.compile(loss= loss, optimizer= optimizer, metrics=['accuracy'])
model3.fit(train_generator_aug2, epochs=EPOCHS2, batch_size=BATCH2, validation_data=validation_generator)
```

نتیجه آموزش: دقت تست بسیار افزایش یافت. یعنی با تولید کردن دیتاست مناسب، توانستیم دقت اعتبارسنجی را از 37 به 74 برسانیم. اما دقت داده آموزش زیاد مطلوب نیست و شبکه مقداری دچار underfit شده است. با توجه به اینکه شبکه در طی 50 ایپوک مرتبا در حال پیشرفت بوده، شبکه هنوز به همگرایی نرسیده است و در صورت آموزش طولانی تر میتوانیم underfit را برطرف کنیم.

```
Epoch 50/50
14/14 [==============================] - 4s 268ms/step - loss: 0.8603 - accuracy: 0.6239 - val_loss: 0.9824 - val_accuracy: 0.7414
```

```
model3.evaluate(validation_generator, batch_size=BATCH)
```

```
4/4 [==============================] - 1s 151ms/step - loss: 0.9824 - accuracy: 0.7414
/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images wi
  "Palette images with Transparency expressed in bytes should be "
[0.9823940992355347, 0.7413793206214905]
```

ت) در این بخش برای برطرف کردن مشکل overfit، از روش dropout استفاده میکنیم. در معماری شبکه بعد از هر لایه maxpooling، یک لایه dropout با احتمال p اضافه میکنیم.

```python
def build_model2(p):
  kernel = 3
  model = Sequential()
  model.add(Conv2D(16, kernel, activation='relu', input_shape=(shape[0], shape[1], 3)))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(p))

  model.add(Conv2D(32, kernel, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(p))

  model.add(Conv2D(64, kernel, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(p))

  model.add(Conv2D(128, kernel, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(p))

  model.add(Flatten())
  model.add(Dense(64, activation='relu'))
  model.add(Dense(5, activation='softmax'))
  return model
```

dropout را با دو احتمال مختلف امتحان میکنیم.

- **در مدل اول، احتمال حذف شدن نورون ها را 0.1 قرار میدهیم. مدل را در 20 ایپوک و سایز batch 32 آموزش میدهیم.**

```python
EPOCHS4=20
BATCH4=32

model4 = build_model2(0.1)
loss4 = CategoricalCrossentropy(from_logits=False)
optimizer4 = tf.keras.optimizers.Adam()
model4.compile(loss=loss4, optimizer= optimizer4, metrics=['accuracy'])
model4.fit(train_generator, epochs=EPOCHS4, batch_size=BATCH4, validation_data=validation_generator)
```

نتیجه آموزش: دقت تست نسبت به حالت اولیه (بدون هیچ روش منظم سازی) بسیار افزایش پیدا کرد و از 37 به 62 رسید. اما شبکه همچنان درگیر overfit است زیرا بر روی داده آموزش دقت 98 را کسب کرده است.

```
Epoch 20/20
14/14 [==============================] - 7s 510ms/step - loss: 0.0823 - accuracy: 0.9817 - val_loss: 2.1222 - val_accuracy: 0.6207
```

```python
model4.evaluate(validation_generator, batch_size=BATCH4)
```

```
1/4 [======>.....................] - ETA: 0s - loss: 2.4888 - accuracy: 0.6250/usr/lo
  "Palette images with Transparency expressed in bytes should be "
4/4 [==============================] - 1s 196ms/step - loss: 2.1222 - accuracy: 0.6207
[2.122199058532715, 0.6206896305084229]
```

- **در مدل دوم، احتمال حذف شدن نورون ها را 0.2 قرار میدهیم. مدل را با شرایط قبل آموزش میدهیم.**

```
model5 = build_model2(0.2)

model5.compile(loss=loss4, optimizer= optimizer4, metrics=['accuracy'])
model5.fit(train_generator, epochs=EPOCHS4, batch_size=BATCH4, validation_data=validation_generator)
```

نتیجه آموزش: دقت تست نسبت به حالت اولیه (بدون هیچ روش منظم سازی) افزایش پیدا کرد و از 37 به 48 رسید. اما نتیجه مطلوب نیست و شبکه underfit شد چون دقت آموزش کم است. پس احتمال حذف 0.2، نسبت به سایر روش ها تاثیر چندانی بر عملکرد شبکه نداشت.

```
Epoch 20/20
14/14 [==============================] - 7s 523ms/step - loss: 0.6750 - accuracy: 0.7294 - val_loss: 1.4875 - val_accuracy: 0.4828
```

```
model5.evaluate(validation_generator, batch_size=BATCH4)
```

```
/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images wit
  "Palette images with Transparency expressed in bytes should be "
4/4 [==============================] - 1s 207ms/step - loss: 1.4875 - accuracy: 0.4828
[1.487547516822815, 0.48275861144065857]
```

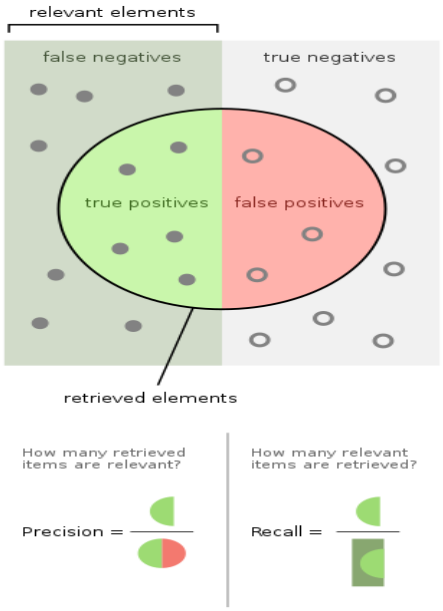
نتیجه نهایی: در این مسئله، توانستیم با استفاده از روش های منظم سازی دقت شبکه (بر روی داده تست) را افزایش دهیم. با استفاده از روش data augmentation، از دقت 37 به 74 و با استفاده از روش dropout به دقت 62 رسیدیم.

بخش امتیازی

د) به معرفی این معیارها میپردازیم:

Precision: درصدی از پیش‌بینی‌های مدل که مرتبط هستند

recall: درصدی از کل پیش‌بینی‌هایی که توسط مدل درست دسته‌بندی‌شده‌اند.



به عنوان مثال در یک الگوریتم یادگیری ماشینی از میان ۱۲ تصویر سگ و گربه ۸ مورد سگ را تشخیص می‌دهد. از ۸ سگ ۵ مورد واقعاً سگ و ۳ مورد دیگر گربه هستند. مقدار precision **برابر با** 5/8 **است و مقدار** recall **برابر با** 5/12 **است.**

F1 score**: میانگین وزن دار از دو معیار** precision **و** recall **است و فرمول آن به شرح زیر است:**

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

با استفاده از کتابخانه Scikit **این معیارها را برای قسمت 1 و 3 محاسبه میکنیم.**

- **قسمت 1: برای محاسبه آنها از متد** classification_report **استفاده میکنیم. ورودی های آن برچسب واقعی داده ها و پیش بینی مدل از آن است. برای اینکه پیش بینی مدل فرمت صحیحی داشته باشد، از** np.argmax **استفاده میکنیم تا ایندکس نورون با بیشترین احتمال را پیدا کنیم. برای برچسب، از** validation_generator.labels **استفاده میکنیم.**

```
yhat_probs = model.predict(validation_generator)
predicted = np.argmax(yhat_probs, axis=1)

print("\n")
print(
    f"{classification_report(validation_generator.labels, predicted)}\n"
)
```

نتیجه آن را برای هر 5 کلاس چاپ میکنیم.

	precision	recall	f1-score	support
0	0.33	0.08	0.12	13
1	0.14	0.40	0.21	10
2	0.07	0.09	0.08	11
3	0.27	0.27	0.27	11
4	0.50	0.08	0.13	13
accuracy			0.17	58
macro avg	0.26	0.18	0.16	58
weighted avg	0.28	0.17	0.16	58

- **قسمت 3: پیاده سازی آن مشابه حالت قبل است. تنها بردار پیش بینی مدل تغییر کرده است.**

احتمال 0.1

```
yhat_probs4 = model4.predict(validation_generator)
predicted4 = np.argmax(yhat_probs4, axis=1)

print("\n")
print(
    f"{classification_report(validation_generator.labels, predicted4)}\n"
)
```

	precision	recall	f1-score	support
0	0.25	0.31	0.28	13
1	0.00	0.00	0.00	10
2	0.17	0.27	0.21	11
3	0.12	0.09	0.11	11
4	0.25	0.23	0.24	13
accuracy			0.19	58
macro avg	0.16	0.18	0.17	58
weighted avg	0.17	0.19	0.17	58

احتمال 0.2

```
print("\n")
print(
    f"{classification_report(validation_generator.labels, predicted5)}\n"
)
```

	precision	recall	f1-score	support
0	0.10	0.08	0.09	13
1	0.00	0.00	0.00	10
2	0.17	0.36	0.24	11
3	0.00	0.00	0.00	11
4	0.20	0.23	0.21	13
accuracy			0.14	58
macro avg	0.09	0.13	0.11	58
weighted avg	0.10	0.14	0.11	58

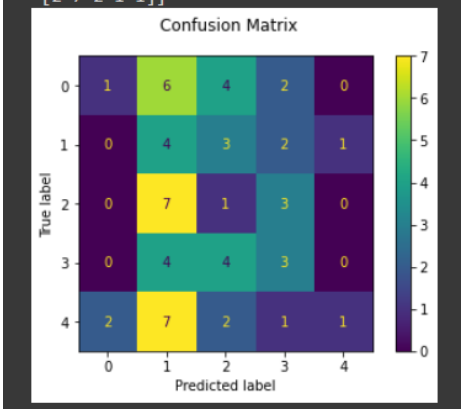
ه) در این سوال با استفاده از ConfustionMatrixDisplay **(موجود در کتابخانه** sklearn**) به رسم ماتریس** confusion **می‌پردازیم.**

در ماتریس confusion هرچه اعداد روی قطر اصلی ماتریس بزرگتر باشند بهتر است.

قسمت اول: ورودی تابع from_predictions برچسب اصلی داده ها و پیش بینی مدل است.

```
24 disp = ConfusionMatrixDisplay.from_predictions(validation_generator.labels, predicted)
25 disp.figure_.suptitle("Confusion Matrix")
26 print(f"Confusion matrix:\n{disp.confusion_matrix}")
27
28 plt.show()
```

```
Confusion matrix:
[[1 6 4 2 0]
 [0 4 3 2 1]
 [0 7 1 3 0]
 [0 4 4 3 0]
 [2 7 2 1 1]]
```



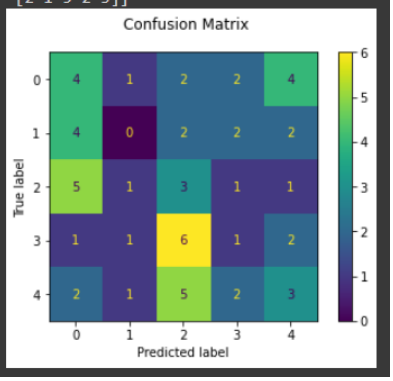
به طور مثال به بررسی سطر اول ماتریس داده تست می‌پردازیم. در سطر سوم لیبل اصلی داده‌ها از نوع کلاس 0 است. طبق پیش‌بینی مدل، یکی از داده‌ها متعلق به کلاس 0، 6 تا از داده‌ها متعلق به کلاس 1 و 4 تا از داده‌ها به کلاس2، 2تا از داده ها به کلاس 3 و 0 داده به کلاس 4 تعلق دارند. یعنی مدل بسیار بد پیش بینی کرده و تنها یکی از داده ها را درست پیش بینی کرده است.

قسمت سوم:

احتمال 0.1

```
18 disp = ConfusionMatrixDisplay.from_predictions(validation_generator.labels, predicted4)
19 disp.figure_.suptitle("Confusion Matrix")
20 print(f"Confusion matrix:\n{disp.confusion_matrix}")
21
22 plt.show()
```

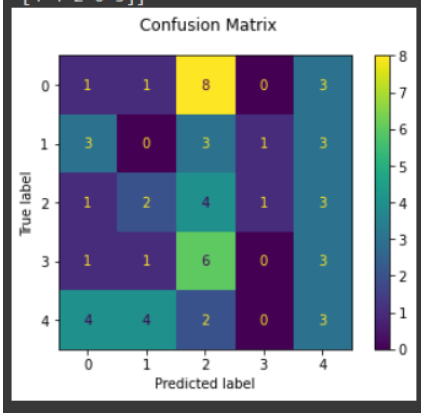
```
Confusion matrix:
[[4 1 2 2 4]
 [4 0 2 2 2]
 [5 1 3 1 1]
 [1 1 6 1 2]
 [2 1 5 2 3]]
```



احتمال 0.2

```
15 disp = ConfusionMatrixDisplay.from_predictions(validation_generator.labels, predicted5)
16 disp.figure_.suptitle("Confusion Matrix")
17 print(f"Confusion matrix:\n{disp.confusion_matrix}")
18
19 plt.show()
```

```
Confusion matrix:
[[1 1 8 0 3]
 [3 0 3 1 3]
 [1 2 4 1 3]
 [1 1 6 0 3]
 [4 4 2 0 3]]
```



منابع

https://stackoverflow.com/questions/47892505/dropout-rate-guidance-for-hidden-layers-in-a-convolution-neural-network

https://datascience.stackexchange.com/questions/85582/dense-layer-vs-convolutional-layer-when-to-use-them-and-how/85587

https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py

https://en.wikipedia.org/wiki/Precision_and_recall

https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd