بسمه تعالی

دانشکده مهندسی کامپیوتر

یادگیری عمیق

نام استاد: دکتر محمدی

تمرین هشتم

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

آذر ۱۴۰۰

۱. پاسخ سوال اول

الف) استفاده اصلی لایه dropout برای جلوگیری از overfit شدن مدل است. کار این لایه این است که به صورت رندوم در هنگام آموزش، وزن تعدادی از نورون ها را صفر میکند. پارامتر نگهداری نورون ها عددی بین ۰ و ۱ است که درواقع احتمال صفر شدن وزن نورون در هر epoch زمان آموزش را نشان میدهد. مثلا اگر بعد از یک لایه که ۱۰۰ نورون دارد یک لایه dropout با p=0.4 قرار دهیم، انگار در هر آموزش به صورت رندوم ۴۰ تا از نورون ها تاثیری روی خروجی نمیگذارند.

این لایه ازین که تمام نورون ها به صورت موازی به یک هدف آپدیت شوند جلوگیری میکند و در نتیجه شبکه کمتر به مقادیر دقیق هر نورون وابسته شده و overfit نمیشود. همچنین در صورت کار نکردن یکی از نورون ها به هر دلیلی، مدل قبلا چنین شرایطی را داشته و موقع تست دچار خطای زیادی نمیشود.

ب) اگر مقدار پارامتر نگهداری شبکه در dropout با همان P، زیاد شود به معنی صفر شدن بیشتر نورون های آن لایه است. و در نتیجه capacity شبکه که به معنی تعداد نورون های شبکه (تعداد لایه ها و نورون های هر لایه) است کاهش میابد. و با استدلال مشابه با کاهش p، Capacity زیاد می شود و درواقع رابطه معکوس دارند.

معمولا احتمال dropout را عددی بین ۰.۲ تا ۰.۵ میگذارند. چون مقادیر کمتر عملا تاثیری ندارند و مقادیر بیشتر هم استفاده از نورون های زیاد هر لایه را بیهوده میکنند. چون اگر dropout زیاد باشد، انگار نورون های شبکه کم هستند و شبکه به خوبی نمی تواند روی داده آموزشی fit شود.

منابع: medium.com، keras.io، oreilly.com

۲. پاسخ سوال دوم

- Fullty connected layer: نوع لایه هایی که تا اینجای درس با آن ها سر و کار داشتیم است. که در tensorflow به عنوان Dense شناخته می شود. این لایه تعدادی نورون دارد که هر کدام وزن مخصوص به خود را دارند و یک bias هم برای آن ها وجود دارد. در این نوع لایه، تمام نورون(پیکسل) های ورودی به تمام نورون های لایه تماما متصل وصل می شوند. برای مسائل classification و به خصوص لایه های انتهایی یک شبکه عمیق بسیار کاربردی هستند.
- Convolutional layer: نقش یک لایه کانولوشن شناسایی ویژگی های محلی در موقعیت های مختلف از نقشه های ویژگی ورودی با filterها و kernel ها قابل یادگیری است. با این لایه درواقع روی مرتبط بودن اجزای مختلف یک ورودی حساب میکنیم و یک فیلتر خاص را روی نقاط مختلف آن پیاده میکنیم. مثلا برای شناسایی ویژگی خاصی در یک تصویر یا یک صوت میتوان از یک لایه کانوولوشنی استفاده کرد. ویژگی مهم آن ها پارامترهای کم قابل یادگیری هم هست. پارامترهای stride و padding و kernel size این لایه باید مشخص باشد.
- Locally connected layer: این نوع لایه کاملاً مشابه لایه Convolutional است اما تنها با یک تفاوت (مهم). در لایه Convolutional فیلتر در بین تمام نورون های خروجی (پیکسل) مشترک بود. به عبارت دیگر، ما از یک فیلتر واحد برای محاسبه تمام نورون ها (پیکسل) یک کانال خروجی استفاده کردیم. با این حال، در Locally-Connected Layer هر نورون (پیکسل) فیلتر مخصوص به خود را دارد. این بدان معناست که تعداد پارامترها در تعداد نورون های خروجی ضرب می شود. این می تواند به شدت تعداد پارامترها را افزایش دهد و اگر داده کافی نداشته باشید، ممکن است با overfitting مواجه شوید. با این حال، این نوع لایه به شبکه شما اجازه می دهد تا انواع مختلفی از ویژگی ها را برای مناطق مختلف ورودی یاد بگیرد. محققان از این ویژگی مفید لایه های محلی متصل به ویژه در تأیید چهره مانند DeepFace و DeepID3 استفاده کردند. با این حال، برخی دیگر از محققان از یک فیلتر مجزا برای هر ناحیه از ورودی به جای هر نورون (پیکسل) استفاده می‌کنند تا از لایه‌های محلی متصل با تعداد پارامترهای کمتر بهره ببرند.

منابع: medium

۳. پاسخ سوال سوم

کدهای مربوط به این سوال در HW8.ipynb ضمیمه شده است. (لینک گوگل کولب)

الف) در این بخش فقط باید داخل تابع flow_from_directory را برای validation_datagen و train_datagen پر میکردیم تا داده های آموزشی و ارزیابی را بدون مشکل تشکیل دهیم. این کار را فعلا بدون data augmentation انجام دادیم.

در این سوال داده ارزیابی و تست را یکی در نظر گرفته ایم.

ب) مدل را با ۴ لایه کانوولوشنی که پس از هر کدام یک لایه maxpooling با ابعاد و stride، (2,2) پس از هر کدام میسازیم تا پارامترها و زمان اجرا خیلی زیاد نشود. به نوعی از overfit هم جلوگیری کند. بعد هم مطابق صورت سوال از دو لایه Dense استفاده میکنیم. نکته مهم این است که دومی به تعداد کلاس ها (۵) نورون داشته باشد و تابع فعالسازی آن softmax باشد. مدل را با بهینه ساز آدام که بهترین است و تابع ضرر categorical crossentropy چون مسئله چندین کلاسه است، اجرا میکنیم.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 16)      592

max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
)

conv2d_1 (Conv2D)            (None, 125, 125, 32)      4640

max_pooling2d_1 (MaxPooling  (None, 62, 62, 32)       0
2D)

conv2d_2 (Conv2D)            (None, 60, 60, 32)        9248

max_pooling2d_2 (MaxPooling  (None, 30, 30, 32)       0
2D)

conv2d_3 (Conv2D)            (None, 28, 28, 64)        18496

max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
2D)

flatten (Flatten)            (None, 12544)             0

dense (Dense)                (None, 64)                802880

dense_1 (Dense)              (None, 5)                 325

=================================================================
Total params: 836,181
Trainable params: 836,181
```

با این هایپرپارامتر های معقول میبینیم که مدل overfit میکند. از نتایج که دقت آموزش به عدد بالایی رسیده و دقت ارزیابی پایین است مشخص است:

```
Epoch 1/15
14/14 [==============================] - 35s 349ms/step - loss: 1.6365 - accuracy: 0.1872 - val_loss: 1.6073 - val_accuracy: 0.1967
Epoch 2/15
14/14 [==============================] - 4s 315ms/step - loss: 1.5951 - accuracy: 0.2603 - val_loss: 1.5759 - val_accuracy: 0.4426
Epoch 3/15
14/14 [==============================] - 4s 314ms/step - loss: 1.5036 - accuracy: 0.3653 - val_loss: 1.6002 - val_accuracy: 0.2787
Epoch 4/15
14/14 [==============================] - 4s 312ms/step - loss: 1.4192 - accuracy: 0.3744 - val_loss: 1.4366 - val_accuracy: 0.4262
Epoch 5/15
14/14 [==============================] - 4s 307ms/step - loss: 1.2725 - accuracy: 0.4338 - val_loss: 1.3540 - val_accuracy: 0.4262
Epoch 6/15
14/14 [==============================] - 4s 318ms/step - loss: 1.0477 - accuracy: 0.5297 - val_loss: 1.4247 - val_accuracy: 0.4262
Epoch 7/15
14/14 [==============================] - 4s 314ms/step - loss: 0.9196 - accuracy: 0.6849 - val_loss: 1.3093 - val_accuracy: 0.4754
Epoch 8/15
14/14 [==============================] - 4s 313ms/step - loss: 0.6669 - accuracy: 0.7489 - val_loss: 1.4057 - val_accuracy: 0.5246
Epoch 9/15
14/14 [==============================] - 4s 314ms/step - loss: 0.5267 - accuracy: 0.7991 - val_loss: 1.4234 - val_accuracy: 0.4918
Epoch 10/15
14/14 [==============================] - 4s 305ms/step - loss: 0.3237 - accuracy: 0.8995 - val_loss: 1.6460 - val_accuracy: 0.5246
Epoch 11/15
14/14 [==============================] - 4s 305ms/step - loss: 0.2982 - accuracy: 0.8995 - val_loss: 1.5981 - val_accuracy: 0.4426
Epoch 12/15
14/14 [==============================] - 4s 305ms/step - loss: 0.2622 - accuracy: 0.9224 - val_loss: 1.9615 - val_accuracy: 0.3770
Epoch 13/15
14/14 [==============================] - 4s 305ms/step - loss: 0.1350 - accuracy: 0.9726 - val_loss: 1.9244 - val_accuracy: 0.4262
Epoch 14/15
14/14 [==============================] - 4s 303ms/step - loss: 0.1062 - accuracy: 0.9726 - val_loss: 2.4362 - val_accuracy: 0.3443
Epoch 15/15
14/14 [==============================] - 4s 303ms/step - loss: 0.0991 - accuracy: 0.9635 - val_loss: 2.4487 - val_accuracy: 0.4426
```

و در نهایت نتیجه سلولی که برای ارزیابی اجرا میکنیم:

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

```
4/4 [==============================] - 1s 225ms/step - loss: 2.4487 - accuracy: 0.4426
[2.4487485885620117, 0.44262295961380005]
```

که دقت پایینی است و در مقایسه با ۹۶ درصد داده آموزشی، overfit مدل واضح است.

پ) تفاوت این بخش در دادن ورودی به ImageDataGenerator است. میخواهیم داده بیشتری از هر کلاس داشته باشیم و این کار را با چرخاندن عکس، جابجایی عرضی آن، جابجایی افقی آن، زوم کردن روی بخشی از عکس، بریدن افقی بخشی از عکس انجام میدهیم. به این طریق مدل نسبت به عکس های کم bias نمیشود و نویز ها اثر کمتری روی تصمیم گیری آن میگذارند. نوع data augmentation ما به این صورت است:

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
```

```
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

با ۱۵ بار اجرا (مثل قسمت قبلی) میبینیم که مدل همچنان fit نشده است و نیاز به اجرای بیشتری دارد. که با توجه به افزایش قابل توجه داده های آموزشی امری طبیعی است.

```
Epoch 1/15
14/14 [==============================] - 11s 721ms/step - loss: 1.6575 - accuracy: 0.2146 - val_loss: 1.5996 - val_accuracy: 0.2787
Epoch 2/15
14/14 [==============================] - 9s 666ms/step - loss: 1.5830 - accuracy: 0.2968 - val_loss: 1.5218 - val_accuracy: 0.3443
Epoch 3/15
14/14 [==============================] - 9s 667ms/step - loss: 1.5354 - accuracy: 0.2877 - val_loss: 1.4985 - val_accuracy: 0.2787
Epoch 4/15
14/14 [==============================] - 9s 669ms/step - loss: 1.5113 - accuracy: 0.3014 - val_loss: 1.4972 - val_accuracy: 0.3115
Epoch 5/15
14/14 [==============================] - 9s 673ms/step - loss: 1.5092 - accuracy: 0.3242 - val_loss: 1.5051 - val_accuracy: 0.2787
Epoch 6/15
14/14 [==============================] - 9s 662ms/step - loss: 1.4813 - accuracy: 0.3333 - val_loss: 1.4967 - val_accuracy: 0.3115
Epoch 7/15
14/14 [==============================] - 9s 664ms/step - loss: 1.3857 - accuracy: 0.4110 - val_loss: 1.4234 - val_accuracy: 0.3443
Epoch 8/15
14/14 [==============================] - 9s 664ms/step - loss: 1.4135 - accuracy: 0.3425 - val_loss: 1.3927 - val_accuracy: 0.3443
Epoch 9/15
14/14 [==============================] - 9s 669ms/step - loss: 1.4464 - accuracy: 0.3379 - val_loss: 1.5269 - val_accuracy: 0.2951
Epoch 10/15
14/14 [==============================] - 9s 661ms/step - loss: 1.4108 - accuracy: 0.3425 - val_loss: 1.5035 - val_accuracy: 0.2459
Epoch 11/15
14/14 [==============================] - 9s 662ms/step - loss: 1.4107 - accuracy: 0.3836 - val_loss: 1.4290 - val_accuracy: 0.3279
Epoch 12/15
14/14 [==============================] - 9s 659ms/step - loss: 1.4125 - accuracy: 0.3744 - val_loss: 1.3984 - val_accuracy: 0.3770
Epoch 13/15
14/14 [==============================] - 9s 662ms/step - loss: 1.2736 - accuracy: 0.4658 - val_loss: 1.4593 - val_accuracy: 0.3279
Epoch 14/15
14/14 [==============================] - 9s 656ms/step - loss: 1.2368 - accuracy: 0.4886 - val_loss: 1.6238 - val_accuracy: 0.2787
Epoch 15/15
14/14 [==============================] - 9s 653ms/step - loss: 1.2742 - accuracy: 0.4429 - val_loss: 1.3893 - val_accuracy: 0.4098
```

پس آموزش را ادامه میدهیم:

```
Epoch 16/45
14/14 [==============================] - 9s 668ms/step - loss: 1.1777 - accuracy: 0.5114 - val_loss: 1.5476 - val_accuracy: 0.5246
Epoch 17/45
14/14 [==============================] - 9s 672ms/step - loss: 1.1612 - accuracy: 0.5434 - val_loss: 1.1954 - val_accuracy: 0.5410
Epoch 18/45
14/14 [==============================] - 9s 674ms/step - loss: 1.0786 - accuracy: 0.5068 - val_loss: 1.2503 - val_accuracy: 0.4754
Epoch 19/45
14/14 [==============================] - 9s 674ms/step - loss: 0.9889 - accuracy: 0.5982 - val_loss: 1.1811 - val_accuracy: 0.4754
Epoch 20/45
14/14 [==============================] - 9s 685ms/step - loss: 0.9453 - accuracy: 0.6027 - val_loss: 1.2213 - val_accuracy: 0.4590
Epoch 21/45
14/14 [==============================] - 9s 668ms/step - loss: 1.0313 - accuracy: 0.5936 - val_loss: 1.1917 - val_accuracy: 0.5246
Epoch 22/45
14/14 [==============================] - 9s 665ms/step - loss: 0.9311 - accuracy: 0.6027 - val_loss: 1.2369 - val_accuracy: 0.4754
Epoch 23/45
14/14 [==============================] - 9s 658ms/step - loss: 0.8816 - accuracy: 0.6210 - val_loss: 0.9908 - val_accuracy: 0.5902
Epoch 24/45
14/14 [==============================] - 9s 665ms/step - loss: 0.9266 - accuracy: 0.6301 - val_loss: 1.2488 - val_accuracy: 0.4590
Epoch 25/45
14/14 [==============================] - 9s 661ms/step - loss: 0.8857 - accuracy: 0.6073 - val_loss: 1.1193 - val_accuracy: 0.5902
Epoch 26/45
14/14 [==============================] - 9s 653ms/step - loss: 0.8258 - accuracy: 0.6530 - val_loss: 1.1349 - val_accuracy: 0.4590
Epoch 27/45
14/14 [==============================] - 9s 652ms/step - loss: 0.8713 - accuracy: 0.6393 - val_loss: 1.2546 - val_accuracy: 0.5246
Epoch 28/45
14/14 [==============================] - 9s 644ms/step - loss: 0.8334 - accuracy: 0.6621 - val_loss: 1.4385 - val_accuracy: 0.4426
Epoch 29/45
14/14 [==============================] - 9s 654ms/step - loss: 0.8472 - accuracy: 0.6621 - val_loss: 1.1749 - val_accuracy: 0.5574
```

```
Epoch 30/45
14/14 [==============================] - 9s 649ms/step - loss: 0.8319 - accuracy: 0.6758 - val_loss: 1.1512 - val_accuracy: 0.5246
Epoch 31/45
14/14 [==============================] - 9s 650ms/step - loss: 0.7807 - accuracy: 0.6712 - val_loss: 1.1353 - val_accuracy: 0.6230
Epoch 32/45
14/14 [==============================] - 9s 654ms/step - loss: 0.7834 - accuracy: 0.6849 - val_loss: 1.0415 - val_accuracy: 0.5902
Epoch 33/45
14/14 [==============================] - 9s 653ms/step - loss: 0.7263 - accuracy: 0.6895 - val_loss: 1.5702 - val_accuracy: 0.5574
Epoch 34/45
14/14 [==============================] - 9s 654ms/step - loss: 0.7582 - accuracy: 0.6941 - val_loss: 1.0984 - val_accuracy: 0.5738
Epoch 35/45
14/14 [==============================] - 9s 653ms/step - loss: 0.6615 - accuracy: 0.6986 - val_loss: 1.0312 - val_accuracy: 0.6393
Epoch 36/45
14/14 [==============================] - 9s 652ms/step - loss: 0.6468 - accuracy: 0.7534 - val_loss: 1.3330 - val_accuracy: 0.5410
Epoch 37/45
14/14 [==============================] - 9s 645ms/step - loss: 0.6656 - accuracy: 0.7352 - val_loss: 1.2192 - val_accuracy: 0.5246
Epoch 38/45
14/14 [==============================] - 9s 646ms/step - loss: 0.8773 - accuracy: 0.6347 - val_loss: 1.3991 - val_accuracy: 0.5082
Epoch 39/45
14/14 [==============================] - 9s 647ms/step - loss: 0.7076 - accuracy: 0.7215 - val_loss: 1.3076 - val_accuracy: 0.5246
Epoch 40/45
14/14 [==============================] - 9s 646ms/step - loss: 0.6377 - accuracy: 0.7534 - val_loss: 1.3634 - val_accuracy: 0.5410
Epoch 41/45
14/14 [==============================] - 9s 646ms/step - loss: 0.6537 - accuracy: 0.7397 - val_loss: 1.3590 - val_accuracy: 0.4918
Epoch 42/45
14/14 [==============================] - 9s 645ms/step - loss: 0.6736 - accuracy: 0.7078 - val_loss: 1.0986 - val_accuracy: 0.6066
Epoch 43/45
14/14 [==============================] - 9s 655ms/step - loss: 0.6940 - accuracy: 0.7306 - val_loss: 1.2069 - val_accuracy: 0.5410
Epoch 44/45
14/14 [==============================] - 9s 655ms/step - loss: 0.6620 - accuracy: 0.7215 - val_loss: 1.3501 - val_accuracy: 0.5574
Epoch 45/45
14/14 [==============================] - 9s 653ms/step - loss: 0.7170 - accuracy: 0.6667 - val_loss: 1.3868 - val_accuracy: 0.5410
<keras.callbacks.History at 0x7f8e9c6765d0>
```

مشخص است که شبکه از نظر overfit شدن نسبت به قسمت الف شرایط بسیار بهتری دارد. دقت روی داده آموزشی در اینجا بالا میرود چون شبکه موارد بیشتری را میبیند. دقت validation هم در جاهایی از آموزش کمی بهتر شده اما بازهم عدد قابل توجهی نیست.

حال برای مقایسه، نوع ساختن دیتاهای جدیدمان را تغییر میدهیم. و به این صورت تعریف میکنیم:

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.3,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)
```

در واقع داده های تست را دیگر تغییری نمیدهیم و سعی میکنیم داده های آموزشی هم با تغییرات کمتری generate کنیم. چون شاید در مثال قبلی در این امر زیاده روی کرده بودیم. نتایج به این صورت به دست آمد:

```
Epoch 36/45
14/14 [==============================] - 8s 550ms/step - loss: 0.8215 - accuracy: 0.6530 - val_loss: 1.5622 - val_accuracy: 0.5082
Epoch 37/45
14/14 [==============================] - 8s 557ms/step - loss: 0.7586 - accuracy: 0.6758 - val_loss: 1.6092 - val_accuracy: 0.5574
Epoch 38/45
14/14 [==============================] - 8s 561ms/step - loss: 0.6997 - accuracy: 0.7078 - val_loss: 1.7154 - val_accuracy: 0.5574
Epoch 39/45
14/14 [==============================] - 8s 544ms/step - loss: 0.7801 - accuracy: 0.7306 - val_loss: 1.7644 - val_accuracy: 0.5246
Epoch 40/45
14/14 [==============================] - 8s 557ms/step - loss: 0.6923 - accuracy: 0.7215 - val_loss: 1.5025 - val_accuracy: 0.5082
Epoch 41/45
14/14 [==============================] - 8s 573ms/step - loss: 0.7294 - accuracy: 0.7443 - val_loss: 1.5776 - val_accuracy: 0.4590
Epoch 42/45
14/14 [==============================] - 8s 575ms/step - loss: 0.6477 - accuracy: 0.7626 - val_loss: 1.4929 - val_accuracy: 0.4754
Epoch 43/45
14/14 [==============================] - 8s 564ms/step - loss: 0.6083 - accuracy: 0.7854 - val_loss: 1.6057 - val_accuracy: 0.5574
Epoch 44/45
14/14 [==============================] - 8s 595ms/step - loss: 0.5717 - accuracy: 0.7626 - val_loss: 1.4401 - val_accuracy: 0.5574
Epoch 45/45
14/14 [==============================] - 8s 577ms/step - loss: 0.6214 - accuracy: 0.7671 - val_loss: 1.4676 - val_accuracy: 0.5246

model.evaluate(validation_generator, batch_size=BATCH_SIZE)

4/4 [==============================] - 1s 233ms/step - loss: 1.4676 - accuracy: 0.5246
[1.4676457643508911, 0.5245901346206665]
```

اوضاع از حالات قبلی بهتر شد اما همچنان با توجه به اختلاف دقت بالای داده آموزشی و ارزیابی، overfit شدن را داریم. در قسمت بعدی با اضافه کردن dropout سعی میکنیم این مشکل را تا حدی حل کنیم.

ت) پس از تمام لایه های کانوولوشن، یک لایه dropout اضافه میکنیم. با احتمالات مختلفی برای dropout این سوال را حل میکنیم. البته data augmentation و سایر هاپرپارامترها را ثابت در نظر میگیریم و فقط احتمال dropout را به عنوان ورودی به تابع build_model میدهیم. ابتدا احتمال را برابر ۰.۲۵ میگذاریم تا هر بار موقع آموزش فقط یک چهارم نورون ها صفر شوند. نتایج به این صورت به دست می آید:

```
Epoch 53/60
14/14 [==============================] - 8s 578ms/step - loss: 0.1011 - accuracy: 0.9726 - val_loss: 1.3029 - val_accuracy: 0.5246
Epoch 54/60
14/14 [==============================] - 8s 581ms/step - loss: 0.0859 - accuracy: 0.9772 - val_loss: 1.2614 - val_accuracy: 0.5410
Epoch 55/60
14/14 [==============================] - 8s 571ms/step - loss: 0.1493 - accuracy: 0.9498 - val_loss: 1.5661 - val_accuracy: 0.4754
Epoch 56/60
14/14 [==============================] - 8s 569ms/step - loss: 0.1810 - accuracy: 0.9315 - val_loss: 1.3441 - val_accuracy: 0.5082
Epoch 57/60
14/14 [==============================] - 8s 565ms/step - loss: 0.1030 - accuracy: 0.9726 - val_loss: 1.3691 - val_accuracy: 0.4918
Epoch 58/60
14/14 [==============================] - 8s 581ms/step - loss: 0.0785 - accuracy: 0.9772 - val_loss: 1.3966 - val_accuracy: 0.4754
Epoch 59/60
14/14 [==============================] - 8s 568ms/step - loss: 0.0766 - accuracy: 0.9589 - val_loss: 1.4344 - val_accuracy: 0.5246
Epoch 60/60
14/14 [==============================] - 8s 568ms/step - loss: 0.1000 - accuracy: 0.9635 - val_loss: 1.4836 - val_accuracy: 0.5410
4/4 [==============================] - 1s 230ms/step - loss: 1.4836 - accuracy: 0.5410
[1.4836385250091553, 0.5409836173057556]
```

نتایج از دفعات قبل بهتر شده است اما همچنان overfit را داریم. پس احتمال dropout را به ۰.۵ افزایش میدهیم:

```
Epoch 51/60
14/14 [==============================] - 8s 568ms/step - loss: 0.4577 - accuracy: 0.8721 - val_loss: 1.5330 - val_accuracy: 0.3443
Epoch 52/60
14/14 [==============================] - 8s 565ms/step - loss: 0.4339 - accuracy: 0.8082 - val_loss: 1.5321 - val_accuracy: 0.3607
Epoch 53/60
14/14 [==============================] - 8s 574ms/step - loss: 0.4976 - accuracy: 0.8311 - val_loss: 1.5089 - val_accuracy: 0.2951
Epoch 54/60
14/14 [==============================] - 8s 568ms/step - loss: 0.4555 - accuracy: 0.8082 - val_loss: 1.4574 - val_accuracy: 0.4098
Epoch 55/60
14/14 [==============================] - 8s 571ms/step - loss: 0.3526 - accuracy: 0.8356 - val_loss: 1.4496 - val_accuracy: 0.3934
Epoch 56/60
14/14 [==============================] - 8s 568ms/step - loss: 0.3288 - accuracy: 0.8539 - val_loss: 1.4408 - val_accuracy: 0.4426
Epoch 57/60
14/14 [==============================] - 8s 571ms/step - loss: 0.4047 - accuracy: 0.8265 - val_loss: 1.4505 - val_accuracy: 0.3770
Epoch 58/60
14/14 [==============================] - 8s 572ms/step - loss: 0.3799 - accuracy: 0.8721 - val_loss: 1.4049 - val_accuracy: 0.4262
Epoch 59/60
14/14 [==============================] - 8s 575ms/step - loss: 0.3412 - accuracy: 0.8995 - val_loss: 1.5847 - val_accuracy: 0.3443
Epoch 60/60
14/14 [==============================] - 8s 578ms/step - loss: 0.3487 - accuracy: 0.8630 - val_loss: 1.4009 - val_accuracy: 0.4426
4/4 [==============================] - 1s 245ms/step - loss: 1.4009 - accuracy: 0.4426
[1.4008538722991943, 0.44262295961380005]
```

همچنین با نرخی بین این دو عدد یعنی ۰.۴ هم امتحان کردم و به چنین نتیجه ای رسیدم:

```
Epoch 59/60
14/14 [==============================] - 8s 616ms/step - loss: 0.3274 - accuracy: 0.8721 - val_loss: 1.3407 - val_accuracy: 0.5246
Epoch 60/60
14/14 [==============================] - 8s 602ms/step - loss: 0.4456 - accuracy: 0.8447 - val_loss: 1.1937 - val_accuracy: 0.4590
4/4 [==============================] - 1s 237ms/step - loss: 1.1937 - accuracy: 0.4590
[1.1936877965927124, 0.4590163826942444]
```

که متاسفانه در هیچکدام ازین حالات مشکل overfit شدن حل نشد. پس با عددی بیش از مقدار معمول یعنی ۰.۷ امتحان کردم که به نتایج زیر رسید:

```
Epoch 54/60
14/14 [==============================] - 8s 601ms/step - loss: 0.7345 - accuracy: 0.7123 - val_loss: 1.5063 - val_accuracy: 0.4754
Epoch 55/60
14/14 [==============================] - 8s 585ms/step - loss: 0.7285 - accuracy: 0.7123 - val_loss: 1.5139 - val_accuracy: 0.3934
Epoch 56/60
14/14 [==============================] - 8s 600ms/step - loss: 0.7257 - accuracy: 0.7306 - val_loss: 1.5201 - val_accuracy: 0.3443
Epoch 57/60
14/14 [==============================] - 8s 582ms/step - loss: 0.6211 - accuracy: 0.7580 - val_loss: 1.5090 - val_accuracy: 0.4262
Epoch 58/60
14/14 [==============================] - 8s 579ms/step - loss: 0.6185 - accuracy: 0.7352 - val_loss: 1.5015 - val_accuracy: 0.4262
Epoch 59/60
14/14 [==============================] - 8s 581ms/step - loss: 0.5853 - accuracy: 0.7580 - val_loss: 1.5078 - val_accuracy: 0.4098
Epoch 60/60
14/14 [==============================] - 8s 588ms/step - loss: 0.5291 - accuracy: 0.7854 - val_loss: 1.4963 - val_accuracy: 0.4426
4/4 [==============================] - 1s 234ms/step - loss: 1.4963 - accuracy: 0.4426
[1.496298909187317, 0.44262295961380005]
```

نتایج روی validation بهبود خاصی نداشتند و صرفا دقت آموزشی کم شده است.

برای حل مشکل overfit در این سوال، با ساختن دیتا ها و اضافه کرد dropout پیشرفت کردیم اما همچنان به ظور کامل رفع نشد. احتمالا باید از روش های دیگری مانند kernel regularizer استفاده میکردیم.

د) برای این قسمت بهترین مدل قسمت قبل را در نظر میگیریم. (آخرین مدلی که تعریف شد و dropout=0.25)

- Precision: معیار خوبی برای تعیین زمانی است که هزینه های مثبت کاذب زیاد است. به عنوان مثال، تشخیص هرزنامه ایمیل. در تشخیص اسپم ایمیل، مثبت کاذب به این معنی است که ایمیلی که غیر اسپم است (منفی واقعی) به عنوان اسپم (اسپم پیش بینی شده) شناسایی شده است. اگر دقت در مدل تشخیص اسپم بالا نباشد، کاربر ایمیل ممکن است ایمیل های مهم را از دست بدهد.
- Recall: در واقع محاسبه می کند که مدل ما چه تعداد از مثبت های واقعی را از طریق برچسب گذاری آن به عنوان مثبت (مثبت واقعی) گرفته است. با استفاده از همین درک، می دانیم که Recall معیاری است که برای انتخاب بهترین مدل خود در زمانی که هزینه زیادی با False Negative وجود دارد، استفاده می کنیم.

به عنوان مثال، در تشخیص تقلب بانکی. اگر تراکنش متقلبانه (Actual Positive) به عنوان غیر متقلبانه (Predicted Negative) پیش بینی شود، عواقب آن می تواند برای بانک بسیار بد باشد.

به طور مشابه، در تشخیص بیماری حاد. اگر یک بیمار (Actual Positive) آزمایش را انجام دهد و پیش بینی شود که بیمار نباشد (Predicted Negative). اگر بیماری مسری باشد، هزینه مربوط به منفی کاذب بسیار بالا خواهد بود.

- F1-score: زمانی مورد نیاز است که می‌خواهید تعادلی بین دقت و فراخوانی ایجاد کنید. قبلاً دیده‌ایم که دقت می‌تواند تا حد زیادی توسط تعداد زیادی از منفی‌های واقعی ایجاد شود که در بیشتر شرایط تجاری، ما روی آن تمرکز زیادی نداریم، در حالی که منفی کاذب و مثبت کاذب معمولاً هزینه‌های تجاری دارند (محسوس و ناملموس) بنابراین امتیاز F1 ممکن است یک امتیاز باشد. اگر نیاز به تعادل بین دقت و یادآوری داریم و توزیع کلاسی ناهمواری وجود دارد (تعداد زیادی از منفی های واقعی)، معیار بهتری برای استفاده است.

در این سوال با تابعی که پیشنهاد شده بود این مقادیر را حساب کردم و به این صورت به دست آمد:

```
from sklearn.metrics import classification_report
y_predict = model.predict(validation_generator)
y_predict = np.argmax(y_predict, axis=1)
y_true = validation_generator.classes
print(classification_report(y_true, y_predict, target_names = categories))
```

```
              precision    recall  f1-score   support

     Leopard       0.19      0.25      0.21        12
        bird       0.20      0.25      0.22        12
         car       0.17      0.18      0.17        11
         cat       0.00      0.00      0.00        13
         dog       0.00      0.00      0.00        13

    accuracy                           0.13        61
   macro avg       0.11      0.14      0.12        61
weighted avg       0.11      0.13      0.12        61
```

منابع: towardsdatascience

ه) confusion matrix خلاصه ای از نتایج پیش بینی در مورد یک مسئله طبقه بندی است. تعداد پیش‌بینی‌های صحیح و نادرست با مقادیر شمارش خلاصه شده و بر اساس هر کلاس تجزیه می‌شوند. ماتریس Confusion یک ماتریس N x N است که برای ارزیابی عملکرد یک مدل طبقه‌بندی استفاده می‌شود، که در آن N تعداد کلاس‌های هدف است. ماتریس مقادیر هدف واقعی را با مقادیر پیش بینی شده توسط مدل یادگیری ماشین مقایسه می کند.

در این مثال با استفاده از تابعی که صورت سوال پیشنهاد داد این ماتریس را به این صورت به دست آوردم:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_true, y_predict)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=categories)
disp.plot(cmap=plt.cm.Greens)
plt.show()
```