تمرین هشتم یادگیری عمیق

محمدعلی فراهت **۹۷۵۲۱۴۲۳**

سوال ۱)

الف) وقتی مدل ما در حال overfit شدن باشد ، می‌توانیم از dropout استفاده کنیم که یکی از روش‌های Regularization می‌باشد. در این روش تعدادی از نورون‌ها را به صورت تصادفی از شبکه خارج می‌کنیم و شبکه با ویژگی‌های کمتری آپدیت می‌شود تا کمتر به یکدیگر وابستگی داشته باشند. هر چه پارامتر نگهداری نورون ها بیشتر باشد، تعداد نورون‌های بیشتری در هر مرحله وجود دارد. معمولا این مقدار را ۰.۵ در نظر می‌گیرند. اگر این عدد را خیلی کم کنیم، تعداد نورون ها انقدری کم می‌شود که شبکه به هم میریزد و underfit می‌شود و درست آموزش نمی‌بیند و اگر خیلی زیاد باشد باز دوباره مشکل overfitting را خواهیم داشت. در شبکه‌های CNN بهتر است این مقدار برای لایه‌های ابتدایی خیلی کم نباشد و هر چه به جلو پیش می‌رویم، مقدار آن کاهش یابد (نه خیلی زیاد)

ب) طبیعتا وقتی از تعداد نورون بیشتری استفاده شود، مدل ظرفیت بیشتری دارد. وقتی مقدار این پارامتر را بیشتر کنیم، تعداد کمتری نورون حذف می‌شوند و این یعنی ظرفیت شبکه بیشتر است ، و اگر پارامتر را کاهش دهیم ، تعداد بیشتری نورون حذف می‌شود و این یعنی ظرفیت کمتر.

سوال ۲)

الف) fully connected : در این لایه نورون ها به همه‌ی نورون‌های قبلی خود متصلند. این لایه همیشه آخرین لایه شبکه عصبی می‌باشد ، چه CNN باشد، چه نباشد. آخرین لایه fc در شبکه، وظیفه طبقه بندی را دارد، تعدادی ورودی می‌گیرد و در خروجی به تعداد کلاس‌های مسئله‌مان نورون دارد که نشان‌دهنده جواب مسئله هستند.

ابتدا لایه Convolutional را بررسی می‌کنیم: این لایه همانطور که در اسم شبکه های CNN هم وجود دارد ، مهم‌ترین لایه است. وقتی ما یک تصویر در ورودی داریم، این لایه اولین لایه‌ای است که تصویر را به آن می‌دهیم. این لایه تعداد زیادی Kernel را بر روی عکس آموزش می‌دهد و از آنها ویژگی استخراج می‌کند (مثلا لبه‌ها در یک تصویر). نکته‌ای که وجود دارد این است که این لایه از یک وزن برای kernel استفاده می‌کند و آپدیت کردن آن وزن‌ها را آنقدر ادامه می‌دهد تا دقت بهینه شود. پس مثلا اگر سایز kernel ما ۵ باشد و همچنین تصویر ورودی ۱۲۸ در ۱۲۸ باشد، مقدار پارامترهای قابل آموزش $256 \times 5 \times 5$ خواهد بود.

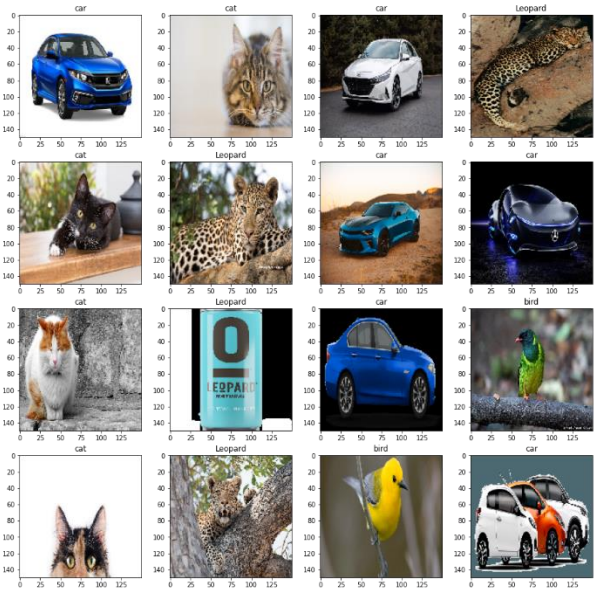
locally connected : این لایه درست مثل لایه Convolutional خواهد بود ولی از کرنل های متفاوت استفاده می‌کند ، یعنی هر feature ای که استخراج می‌شود از دیگری جدا است و یک کرنل نداریم که بین همه آن‌ها مشترک باشد(sharing of weights). با این کار تعداد پارامترهای قابل آموزش بسیار زیاد می‌شود، برای مثال ، اگر همان ورودی مثال قبل را در نظر بگیریم، تعداد پارامتر ها برابر است با $124 \times 124 \times 256 \times 5 \times 5$

اما اگر از GPU استفاده کنیم این مشکل کمتر می‌شود و میتواند همزمان آن را اجرا کند.

سوال ۳)

الف) ابتدا داده‌ها را در drive ذخیره می‌کنیم و سپس بعد از لود کردن آنها، در ImageDataGenerator از آن‌ها استفاده میکنیم، نمونه ای از تصاویر ذخیره شده را می‌بینید:

```
from keras.preprocessing.image import ImageDataGenerator
batch_size = 16
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory("train", target_size=(150, 150), batch_size=batch_size)
validation_generator = test_datagen.flow_from_directory("test", target_size=(150, 150), batch_size=batch_size)
```



ب) همانطور که خواسته شد، یک شبکه کانولوشنی شامل چهار لایه کانوولوشنی و دو لایه dense ساخته شد که سایز کرنل، ۵ است (epoch=50 و batch_size=16). هیچ داده افزایی استفاده نشد و نتیجه آن بسیار بد بود و دقت تست روی ۴۰٪ ایستاد ولی دقت آموزش ۱۰۰٪ بود :) این یعنی مدل overfit شده است و باید جلویش را گرفت.

```
[47] model.evaluate(validation_generator, batch_size=16)

/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Transparency expressed in by
  "Palette images with Transparency expressed in bytes should be "
4/4 [==============================] - 1s 129ms/step - loss: 5.2953 - accuracy: 0.4068
[5.295302867889404, 0.4067796468734741]
```

```
3s 187ms/step - loss: 2.8597e-04 - accuracy: 1.0000 - val_loss: 5.2953 - val_accuracy: 0.4068
90>
```

```python
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.losses import categorical_crossentropy

def build_model():
  kernel_size = 5
  model = Sequential()
  model.add(Conv2D(8, kernel_size, activation ="relu", input_shape=(150, 150, 3)))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Conv2D(16, kernel_size, activation ="relu"))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Conv2D(32, kernel_size, activation ="relu"))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Conv2D(64, kernel_size, 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
```

```python
model = build_model()
loss = tf.keras.losses.CategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()
model.compile(loss= loss, optimizer= optimizer, metrics=["accuracy"])
model.fit(train_generator, epochs=50, batch_size=16, validation_data=validation_generator)
```

پ) حالا می‌خواهیم data augmentation را اضافه کنیم ، برای اولین روش از سه ویژگی استفاده می‌کنیم : rotation_range و width_shift_range و height_shift_range . با این کار دیتاهای جدیدی تولید می‌شود که از نظر زاویه چرخش، جابجا شدن در طول و عرض، با تصاویر اولیه متفاوت هستند. و احتمالا باید مشکل overfitting را حل کند:

```python
train_datagen_aug1 = ImageDataGenerator(rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2)
train_generator1 = train_datagen_aug1.flow_from_directory("train", target_size=(150, 150), batch_size=batch_size)

model = build_model()
model.compile(loss= loss, optimizer= optimizer, metrics=["accuracy"])
model.fit(train_generator1, epochs=50, batch_size=16, validation_data=validation_generator)
```

```
==================] - 4s 277ms/step - loss: 1.0878 - accuracy: 0.5160 - val_loss: 0.9712 - val_accuracy: 0.5932
story at 0x7fcea03f3f90>
```

```
- loss: 1.3421 - accuracy: 0.5085
```

می‌بینیم که اختلاف دیتا آموزش و ولیدیشن کم شد (۸٪ اختلاف دارند) ، ولی هنوز جا دارد که بهتر شود. توجه شود که مدل را عوض نکردم و مانند قبل است، فقط دیتای ورودی را augment کردم. و دلیل پایین بودن نسبی دقت همین است ، شاید اگر مدل پیچیده‌تر بود ، مدل بهتر train می‌شد. ولی برای مقایسه صحیح ، به آن دست نمی‌زنم.

حالا دوباره دیتا بیشتری اضافه می‌کنیم، این بار علاوه بر قبلی ها ، از zoom_range ، horizontal_flip و shear_range هم استفاده می‌کنم. کار این سه به ترتیب ، تغییر زوم در عکس ، قرینه کردن عکس، و تغییر زاویه عکس می‌باشد. با افزودن این موارد ، هم دقت ما بالاتر رفت ، هم overfitting به طور کامل از بین رفت و حالا دقت تست و آموزش بسیار نزدیک می‌باشند. (۶۱ و ۶۰)

```
train_datagen_aug2 = 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,)
train_generator2 = train_datagen_aug1.flow_from_directory("train", target_size=(150, 150), batch_size=batch_size)

model = build_model()
model.compile(loss= loss, optimizer= optimizer, metrics=["accuracy"])
model.fit(train_generator2, epochs=50, batch_size=16, validation_data=validation_generator)
```

```
==================] - 4s 268ms/step - loss: 0.9485 - accuracy: 0.6027 - val_loss: 0.9469 - val_accuracy: 0.6102
tory at 0x7fcea0a9b2d0>
```

```
loss: 0.9469 - accuracy: 0.6102
```

ت) حالا از dropout استفاده می‌کنیم تا شرایط را بهتر کنیم. ابتدا با احتمال 0.5 استفاده می‌کنیم. بعد از هر لایه pooling یک dropout گذاشتم.

```
def build_model(d_rate):
  kernel_size = 5
  model = Sequential()
  model.add(Conv2D(8, kernel_size, activation ="relu", input_shape=(150, 150, 3)))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Dropout(d_rate))
  model.add(Conv2D(16, kernel_size, activation ="relu"))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Dropout(d_rate))
  model.add(Conv2D(32, kernel_size, activation ="relu"))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Dropout(d_rate))
  model.add(Conv2D(64, kernel_size, activation ="relu"))
  model.add(MaxPooling2D(pool_size = (2,2)))
  model.add(Dropout(d_rate))
  model.add(Flatten())
  model.add(Dense(64, activation = "relu"))
  model.add(Dense(5, activation = "softmax"))
  return model
```

حالا خروجی را برای 0.5 می‌بینیم:

```
==================] - 4s 273ms/step - loss: 1.2559 - accuracy: 0.4201 - val_loss: 1.1426 - val_accuracy: 0.5593
tory at 0x7fce90020750>
```

کاهش زیاد دقت و همچنین بیشتر شدن overfitting را شاهد بودیم، پس استفاده از این تعداد augmentation و همزمان با dropout روش خوبی نیست . حالا آن را به 0.8 تغییر می‌دهیم تا ببینیم چه اتفاقی می‌افتد:

```
model = build_model(0.8)
loss = tf.keras.losses.CategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()
model.compile(loss= loss, optimizer= optimizer, metrics=["accuracy"])
model.fit(train_generator2, epochs=50, batch_size=16, validation_data=validation_generator)
```

```
==============] - 4s 271ms/step - loss: 1.2883 - accuracy: 0.3653 - val_loss: 1.8509 - val_accuracy: 0.1864
y at 0x7fcdae7e59d0>
```

در این حالت هم دقت ما بسیار بسیار کاهش یافته ، این یعنی لایه dropout را به درستی انتخاب نکردیم و وقتی از آن استفاده می‌کنیم دقت کاهش می‌یابد. اما استفاده از dropout باعث شد کمتر overfit شویم (به نسبت اول)

در کل ما توانستیم با data augmentation و dropout شبکه خودمان را بدون دیتای اضافه‌ای از بیرون، بهبود ببخشیم و هدف اصلی این تمرین هم همین بوده است.

با تشکر