11. 합성곱 신경망

인공 신경망은 파라미터만 잘 조정해주면 매우 다양한 함수를 나타낼 수 있다. 그래서 우리가 원하는 결과를 얻기 위해 모형의 형태에 제약을 주기도 한다. 이러한 방법 중에 대표적인 것으로 합성곱 신경망과 순환신경망이 있다.

이미지를 기계학습으로 처리할 경우, 이미지의 점 하나가 변수가 된다. 가로 28 X 세로 28인 이미지가 있다면, 이 이미지는 784개의 점으로 이뤄져 있다. 즉 784개의 변수로된 데이터를 분석하는 것과 같게 된다.

이미지는 점들이 모여 작은 형태를 이루고, 작은 형태들이 모여 더 큰 형태를 이루는 특징이 있다. 따라서 이미지들을 단순히 여러 개의 점들로 보는 것보다, 이런 점들이 모여 만드는 형태들을 모형에 포함시킨다면 이미지 기계학습의 성능을 향상시킬 수 있다. 실제로 생물의 시각도 같은 방식으로 작동한다. 생물에서 작은 영역에 반응하는 세포들을 receptive field라고 부른다. 여기에서 착안한 것이 합성곱 신경망(convolutional neural network, CNN)이다.

11.1. 합성곱

합성곱(convolution)은 원래 이미지 처리에 많이 사용되는 방법이다.예를 들어 다음과 같은 이미지에

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

아래와 같은 필터를

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

합성곱 해보자. 위의 필터는 수직선에 반응하는 비슷한 기능을 한다.

계산은 다음과 같이 이뤄진다. 먼저 왼쪽 위의 3x3 부분만을 본다.

$$\begin{bmatrix} 1 & 0 & \cdot \\ 1 & 0 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

그 다음 같은 위치에 있는 값들끼리 곱해준다.

$$\begin{bmatrix} 1 \times 1 & -1 \times 0 \\ 1 \times 1 & -1 \times 0 \end{bmatrix}$$

그러면 계산 결과가 다음과 같다.

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

마지막으로 이들의 합을 구하면 2가 된다.

다음으로는 한 칸 오른쪽으로 옮겨서 다음 3x3 부분에

$$\begin{bmatrix} \cdot & 0 & 0 \\ \cdot & 0 & 0 \\ \cdot & \cdot & \cdot \end{bmatrix}$$

똑같은 계산을 적용하면 0이 된다. 이렇게 계산을 반복하고 그 결과를 아래와 같은 feature map으로 만든다.

$$\begin{bmatrix} 2 & 0 \\ 1 & 0 \end{bmatrix}$$

위의 행렬을 보면 그림의 왼쪽 위가 수직선 패턴이 가장 강하다는 것을 알 수 있다. 이번에는 수평선 패턴에 반응하는 아래와 같은 필터를

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

똑같은 이미지에 합성곱을 하면 다음과 같은 feature map을 만들 수 있다.

$$\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix}$$

위의 행렬을 보면 그림의 오른쪽 아래가 수평선 패턴이 가장 강하다는 것을 알 수 있다.

이런 식으로 다양한 필터를 만들어 합성곱을 해주면 이미지에서 특정 패턴을 추출할 수 있다. 합성곱은 패턴 추출에만 사용되는 것이 아니고 흐리게 하기(blur)나 날카롭게 하기(sharpen) 등 이미지 편집에 사용되는 여러 필터 적용에도 쓸 수 있다.

합성곱 신경망에서 필터 역할을 하는 레이어를 합성곱 레이어라고 한다. 기존의 이미지 처리가 미리 만들어진 필터를 적용하는 방식이라면, CNN은 이러한 필터 자체를 학습한다는 것이 특징이다. 필터라는 것은 결국 그림의 일부에 일정한 계수를 곱해주고 그 결과를 더하는 것이로 일반적인 신경망의 구조와 다를 것이 없다. 다만, 같은 계수를 그림의 부분들에 반복적으로 적용한다는 점만 차이가 있을 뿐이다.

11.2. Pooling

합성곱을 하면 feature map에서 근처의 값들끼리 비슷해지는 경향을 띈다. 그래서 근방의 영역에 있는 값들을 묶어 하나의 값으로 줄이는 pooling을 한다. pooling의 방법에는 다음과 같은 것들이 있다.

- max: 영역의 최대값
- average: 영역의 평균
- L2: 영역의 제곱합의 제곱근

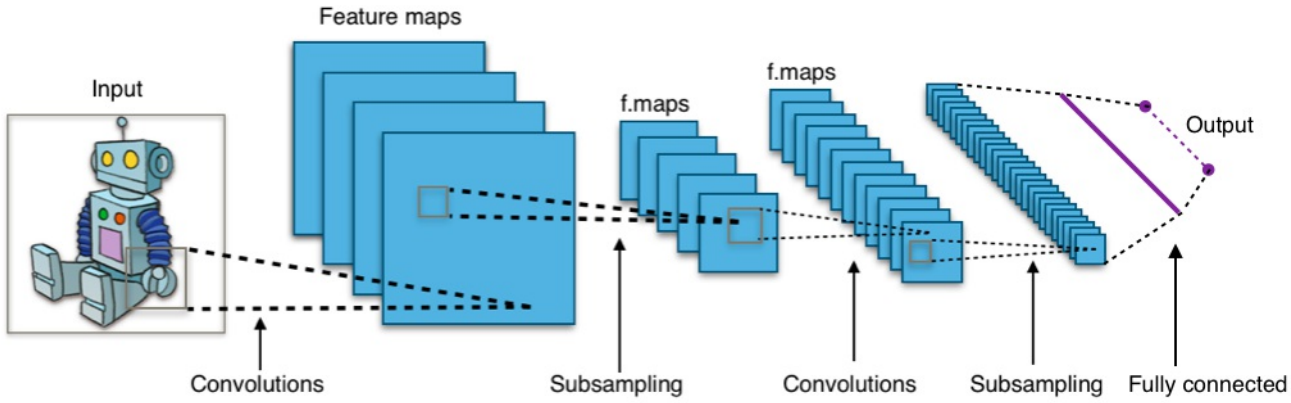
앞에서 예로 들었던 feature map에 2x2 max pooling을 적용하면 똑같이 2가 나오게 된다.

따라서 우리는 정확한 위치는 알 수 없지만 3x3 그림에 수직선과 수평선이 하나 있다는 것을 알 수 있게 된다. 이러한 pooling은 정보를 줄이게 되지만, 과적합도 줄어들 뿐만 아니라 그림이 일부 변형되더라도 학습결과를 유지할 수 있게 해준다. 왜냐하면 그림이 수직이나 수평 방향으로 1픽셀 이동하더라도 pooling을 거친 결과는 동일하기 때문이다.

합성곱 레이어와 달리 풀링 레이어는 별도의 학습이 이뤄지지 않는다.

11.3. 합성곱 신경망

이제 이미지를 입력 받아 그 위로 합성곱 레이어와 풀링 레이어를 반복해서 쌓으면 CNN이 된다. 두 가지 레이어를 반복해서 쌓는 이유는 작은 형태를 바탕으로 다시 큰 형태를 처리하게 하기 위해서다.



11.4. 실습

11.4.1. 데이터 준비

```
from urllib.request import urlretrieve
from zipfile import ZipFile


urlretrieve('http://doc.mindscale.kr/km/unstructured/dog-vs-cat.zip',
            'dog-vs-cat.zip')

with ZipFile('dog-vs-cat.zip') as z:
    z.extractall()

from keras.preprocessing.image import ImageDataGenerator

train = ImageDataGenerator(rescale=1.0/255).flow_from_directory(
    'dog-vs-cat/train',
    target_size=(100, 100),
    class_mode='binary')

valid = ImageDataGenerator(rescale=1.0/255).flow_from_directory(
    'dog-vs-cat/validation',
    target_size=(100, 100),
    class_mode='binary',
    shuffle=False)
```

```
/home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
/home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/matplotlib/__init__.py:1067: UserWarning: Duplicate key in file "/home/u
  (fname, cnt))
/home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/matplotlib/__init__.py:1067: UserWarning: Duplicate key in file "/home/u
  (fname, cnt))

Found 2000 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
```

11.4.2. 합성곱 신경망과 텐서보드

```
from keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from keras.models import Sequential

model1 = Sequential()
model1.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model1.add(MaxPooling2D((2, 2)))
model1.add(Flatten())
model1.add(Dense(1, activation='sigmoid'))

model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 98, 98, 32)        896
_________________________________________________________________
```

```
max_pooling2d_6 (MaxPooling2 (None, 49, 49, 32)        0
_________________________________________________________________
flatten_6 (Flatten)          (None, 76832)             0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 76833
=================================================================
Total params: 77,729
Trainable params: 77,729
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.optimizers import Adam, RMSprop

model1.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=Adam())
```

텐서보드

학습 기록을 `log_model1`로 저장하고 텐서보드로 모니터한다.

```
from keras.callbacks import EarlyStopping, TensorBoard

history1 = model1.fit_generator(
    train, validation_data=valid, epochs=30,
    callbacks=[
        EarlyStopping(monitor = "val_loss", patience=2),
        TensorBoard(log_dir='log_model1')
    ])
```

```
Epoch 1/30
63/63 [==============================] - 15s 236ms/step - loss: 0.7803 - acc: 0.5347 - val_loss: 0.6694 - val_acc: 0.5950
Epoch 2/30
63/63 [==============================] - 14s 228ms/step - loss: 0.6180 - acc: 0.6627 - val_loss: 0.6166 - val_acc: 0.6837
Epoch 3/30
63/63 [==============================] - 13s 206ms/step - loss: 0.5118 - acc: 0.7654 - val_loss: 0.5970 - val_acc: 0.6587
Epoch 4/30
63/63 [==============================] - 13s 208ms/step - loss: 0.4174 - acc: 0.8323 - val_loss: 0.6203 - val_acc: 0.6488
Epoch 5/30
63/63 [==============================] - 13s 209ms/step - loss: 0.3370 - acc: 0.8814 - val_loss: 0.5938 - val_acc: 0.6813
Epoch 6/30
63/63 [==============================] - 13s 209ms/step - loss: 0.2762 - acc: 0.9152 - val_loss: 0.6249 - val_acc: 0.6687
Epoch 7/30
63/63 [==============================] - 13s 207ms/step - loss: 0.2249 - acc: 0.9459 - val_loss: 0.6427 - val_acc: 0.6713
```

작업 디렉토리 명령창에 `tensorboard --logdir=log_model1`이라고 입력하여 텐서보드를 실행하고 웹 브라우저 주소창에 http://127.0.0.1:6006 또는 http://localhost:6006 으로 텐서보드에 접속한다.

11.4.3. 더 깊은 신경망

신경망의 층을 더 추가한다.

```
model2 = Sequential()
model2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model2.add(MaxPooling2D((2, 2)))
model2.add(Conv2D(32, (3, 3), activation='relu'))
model2.add(MaxPooling2D((2, 2)))
model2.add(Flatten())
model2.add(Dense(1, activation='sigmoid'))

model2.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=Adam())

history2 = model2.fit_generator(
    train, validation_data=valid, epochs=30,
    callbacks=[
        EarlyStopping(monitor = "val_loss", patience=2),
        TensorBoard(log_dir='log_model2')
    ])
```

```
Epoch 1/30
63/63 [==============================] - 15s 240ms/step - loss: 0.6971 - acc: 0.5238 - val_loss: 0.6859 - val_acc: 0.5012
Epoch 2/30
63/63 [==============================] - 13s 208ms/step - loss: 0.6428 - acc: 0.6215 - val_loss: 0.6346 - val_acc: 0.6288
Epoch 3/30
63/63 [==============================] - 13s 207ms/step - loss: 0.5838 - acc: 0.6895 - val_loss: 0.6014 - val_acc: 0.6813
Epoch 4/30
63/63 [==============================] - 13s 208ms/step - loss: 0.5297 - acc: 0.7386 - val_loss: 0.6111 - val_acc: 0.6538
Epoch 5/30
63/63 [==============================] - 13s 208ms/step - loss: 0.5052 - acc: 0.7579 - val_loss: 0.6211 - val_acc: 0.6625
```

11.4.4. 데이터 증강

회전, 상하좌우로 이동, 기울이거나 확대 또는 뒤집는 방식으로 데이터를 증강한다.

```
img_gen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,      # 40도까지 회전
        width_shift_range=0.2,  # 20%까지 좌우 이동
        height_shift_range=0.2, # 20%까지 상하 이동
        shear_range=0.2,        # 20%까지 기울임
        zoom_range=0.2,         # 20%까지 확대
        horizontal_flip=True,   # 좌우 뒤집기
    )

train_ag = img_gen.flow_from_directory(
        'dog-vs-cat/train',
        target_size=(100, 100),
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
```

```
model3 = Sequential()
model3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model3.add(MaxPooling2D((2, 2)))
model3.add(Conv2D(32, (3, 3), activation='relu'))
model3.add(MaxPooling2D((2, 2)))
model3.add(Flatten())
```

```
model3.add(Dense(1, activation='sigmoid'))

model3.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 47, 47, 32)        9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 23, 23, 32)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 16928)             0
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 16929
=================================================================
Total params: 27,073
Trainable params: 27,073
Non-trainable params: 0
_________________________________________________________________
```

```
model3.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=Adam())

history3 = model3.fit_generator(
    train, validation_data=valid, epochs=30,
    callbacks=[
        EarlyStopping(monitor = "val_loss", patience=2),
        TensorBoard(log_dir='log_model3')
    ])
```

```
Epoch 1/30
63/63 [==============================] - 15s 241ms/step - loss: 0.6946 - acc: 0.5491 - val_loss: 0.6848 - val_acc: 0.5975
Epoch 2/30
63/63 [==============================] - 13s 207ms/step - loss: 0.6686 - acc: 0.6077 - val_loss: 0.6632 - val_acc: 0.6138
Epoch 3/30
63/63 [==============================] - 13s 208ms/step - loss: 0.6247 - acc: 0.6711 - val_loss: 0.6253 - val_acc: 0.6687
Epoch 4/30
63/63 [==============================] - 13s 209ms/step - loss: 0.5750 - acc: 0.7054 - val_loss: 0.6199 - val_acc: 0.6538
Epoch 5/30
63/63 [==============================] - 13s 208ms/step - loss: 0.5187 - acc: 0.7391 - val_loss: 0.5950 - val_acc: 0.6913
Epoch 6/30
63/63 [==============================] - 13s 208ms/step - loss: 0.4896 - acc: 0.7584 - val_loss: 0.6121 - val_acc: 0.6787
Epoch 7/30
63/63 [==============================] - 13s 208ms/step - loss: 0.4358 - acc: 0.7986 - val_loss: 0.6036 - val_acc: 0.6863
```

11.4.5. 드롭아웃과 학습률 조정 그리고 ModelCheckpoint

```
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint
```

드롭아웃 레이어를 넣는다.

```
model4 = Sequential()
model4.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model4.add(MaxPooling2D((2, 2)))
model4.add(Conv2D(64, (3, 3), activation='relu'))
model4.add(MaxPooling2D((2, 2)))
model4.add(Conv2D(128, (3, 3), activation='relu'))
model4.add(MaxPooling2D((2, 2)))
model4.add(Flatten())
model4.add(Dropout(0.5))
model4.add(Dense(512, activation='relu'))
model4.add(Dense(1, activation='sigmoid'))
```

학습률을 0.0001로 낮춘다.

```
model4.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=RMSprop(lr=0.0001))
```

체크포인트

가장 성능이 좋은 모델을 `model4-00.hdf5`와 같은 파일 명으로 저장한다.

```
history4 = model4.fit_generator(
    train, validation_data=valid, epochs=30,
    callbacks=[
        ModelCheckpoint('model4-{epoch:02d}.hdf5', save_best_only=True),
        TensorBoard(log_dir='log_model4')
    ])
```

```
Epoch 1/30
63/63 [==============================] - 15s 239ms/step - loss: 0.6939 - acc: 0.5233 - val_loss: 0.6753 - val_acc: 0.6175
Epoch 2/30
63/63 [==============================] - 13s 209ms/step - loss: 0.6719 - acc: 0.5848 - val_loss: 0.6801 - val_acc: 0.5312
Epoch 3/30
63/63 [==============================] - 13s 209ms/step - loss: 0.6341 - acc: 0.6523 - val_loss: 0.7012 - val_acc: 0.5487
Epoch 4/30
63/63 [==============================] - 13s 209ms/step - loss: 0.6112 - acc: 0.6751 - val_loss: 0.6531 - val_acc: 0.5962
Epoch 5/30
63/63 [==============================] - 13s 212ms/step - loss: 0.5871 - acc: 0.6994 - val_loss: 0.6376 - val_acc: 0.6350
Epoch 6/30
63/63 [==============================] - 13s 208ms/step - loss: 0.5559 - acc: 0.7227 - val_loss: 0.6355 - val_acc: 0.6450
Epoch 7/30
63/63 [==============================] - 13s 209ms/step - loss: 0.5334 - acc: 0.7475 - val_loss: 0.5898 - val_acc: 0.6775
Epoch 8/30
63/63 [==============================] - 13s 209ms/step - loss: 0.4967 - acc: 0.7713 - val_loss: 0.5948 - val_acc: 0.6787
Epoch 9/30
63/63 [==============================] - 13s 209ms/step - loss: 0.4963 - acc: 0.7699 - val_loss: 0.5929 - val_acc: 0.6863
Epoch 10/30
63/63 [==============================] - 13s 209ms/step - loss: 0.4676 - acc: 0.7798 - val_loss: 0.5835 - val_acc: 0.7050
Epoch 11/30
63/63 [==============================] - 13s 208ms/step - loss: 0.4451 - acc: 0.7936 - val_loss: 0.5802 - val_acc: 0.7037
Epoch 12/30
63/63 [==============================] - 13s 209ms/step - loss: 0.4339 - acc: 0.7976 - val_loss: 0.5723 - val_acc: 0.7063
```

```
Epoch 13/30
63/63 [==============================] - 13s 210ms/step - loss: 0.4100 - acc: 0.8140 - val_loss: 0.5779 - val_acc: 0.7000
Epoch 14/30
63/63 [==============================] - 13s 208ms/step - loss: 0.4103 - acc: 0.8075 - val_loss: 0.5696 - val_acc: 0.7113
Epoch 15/30
63/63 [==============================] - 13s 209ms/step - loss: 0.3846 - acc: 0.8343 - val_loss: 0.5732 - val_acc: 0.7163
Epoch 16/30
63/63 [==============================] - 13s 208ms/step - loss: 0.3754 - acc: 0.8378 - val_loss: 0.5744 - val_acc: 0.7175
Epoch 17/30
63/63 [==============================] - 13s 211ms/step - loss: 0.3565 - acc: 0.8408 - val_loss: 0.5867 - val_acc: 0.7100
Epoch 18/30
63/63 [==============================] - 13s 208ms/step - loss: 0.3446 - acc: 0.8507 - val_loss: 0.6245 - val_acc: 0.6925
Epoch 19/30
63/63 [==============================] - 13s 211ms/step - loss: 0.3307 - acc: 0.8566 - val_loss: 0.6485 - val_acc: 0.6937
Epoch 20/30
63/63 [==============================] - 13s 209ms/step - loss: 0.3282 - acc: 0.8571 - val_loss: 0.5875 - val_acc: 0.7125
Epoch 21/30
63/63 [==============================] - 14s 221ms/step - loss: 0.3145 - acc: 0.8715 - val_loss: 0.6014 - val_acc: 0.7150
Epoch 22/30
63/63 [==============================] - 13s 208ms/step - loss: 0.2983 - acc: 0.8760 - val_loss: 0.5960 - val_acc: 0.7063
Epoch 23/30
63/63 [==============================] - 15s 230ms/step - loss: 0.2809 - acc: 0.8805 - val_loss: 0.6126 - val_acc: 0.7150
Epoch 24/30
63/63 [==============================] - 13s 208ms/step - loss: 0.2761 - acc: 0.8869 - val_loss: 0.7494 - val_acc: 0.6750
Epoch 25/30
63/63 [==============================] - 13s 209ms/step - loss: 0.2781 - acc: 0.8824 - val_loss: 0.6123 - val_acc: 0.7137
Epoch 26/30
63/63 [==============================] - 13s 208ms/step - loss: 0.2498 - acc: 0.9003 - val_loss: 0.6248 - val_acc: 0.7262
Epoch 27/30
63/63 [==============================] - 13s 210ms/step - loss: 0.2390 - acc: 0.9122 - val_loss: 0.6527 - val_acc: 0.7100
Epoch 28/30
63/63 [==============================] - 13s 210ms/step - loss: 0.2333 - acc: 0.9048 - val_loss: 0.6399 - val_acc: 0.7238
Epoch 29/30
63/63 [==============================] - 13s 208ms/step - loss: 0.2241 - acc: 0.9132 - val_loss: 0.7072 - val_acc: 0.6950
Epoch 30/30
63/63 [==============================] - 13s 209ms/step - loss: 0.2234 - acc: 0.9157 - val_loss: 0.6412 - val_acc: 0.7188
```

저장된 모델 목록 보기

```
import glob

model_files = glob.glob('model4-*.hdf5')

model_files
```

```
['model4-05.hdf5',
 'model4-10.hdf5',
 'model4-01.hdf5',
 'model4-04.hdf5',
 'model4-14.hdf5',
 'model4-11.hdf5',
 'model4-07.hdf5',
 'model4-06.hdf5',
 'model4-12.hdf5']
```

```
last_model = sorted(model_files)[-1]

last_model
```

```
'model4-14.hdf5'
```

Colab에서 다운로드 받기

Google Colab을 쓰는 경우 아래 코드를 이용해 모델 파일을 다운로드 받을 수 있다.

```
from google.colab import files

files.download(last_model)
```

모델 다시 불러오기

```
import keras.models

_model)
```