소규모 데이터셋에서 CNN 네트워크 학습하기

Youngtaek Hong, PhD

1. 사전 훈련된 네트워크 사용하기

Pre trained model

- 작은 이미지 데이터셋에 딥 러닝을 적용하는 **일반적이고 매우 효과적인** 방법은 사전 훈련된 네트워크 pretrained model 를 사용하는 것입니다.
- 사전 훈련된 네트워크는 일반적으로 대규모 이미지 분류 문제를 위해 대량의 데이터셋에서 미리 훈련되어 저장된 네트워크입니다.
- 원본 데이터셋이 충분히 크고 일반적이라면 사전 훈련된 네트워크에 의해 학습된 특성의 계층 구조는 실제 세상에 대한 일반적인 **모델로 효율적인 역할**을 할 수 있습니다.

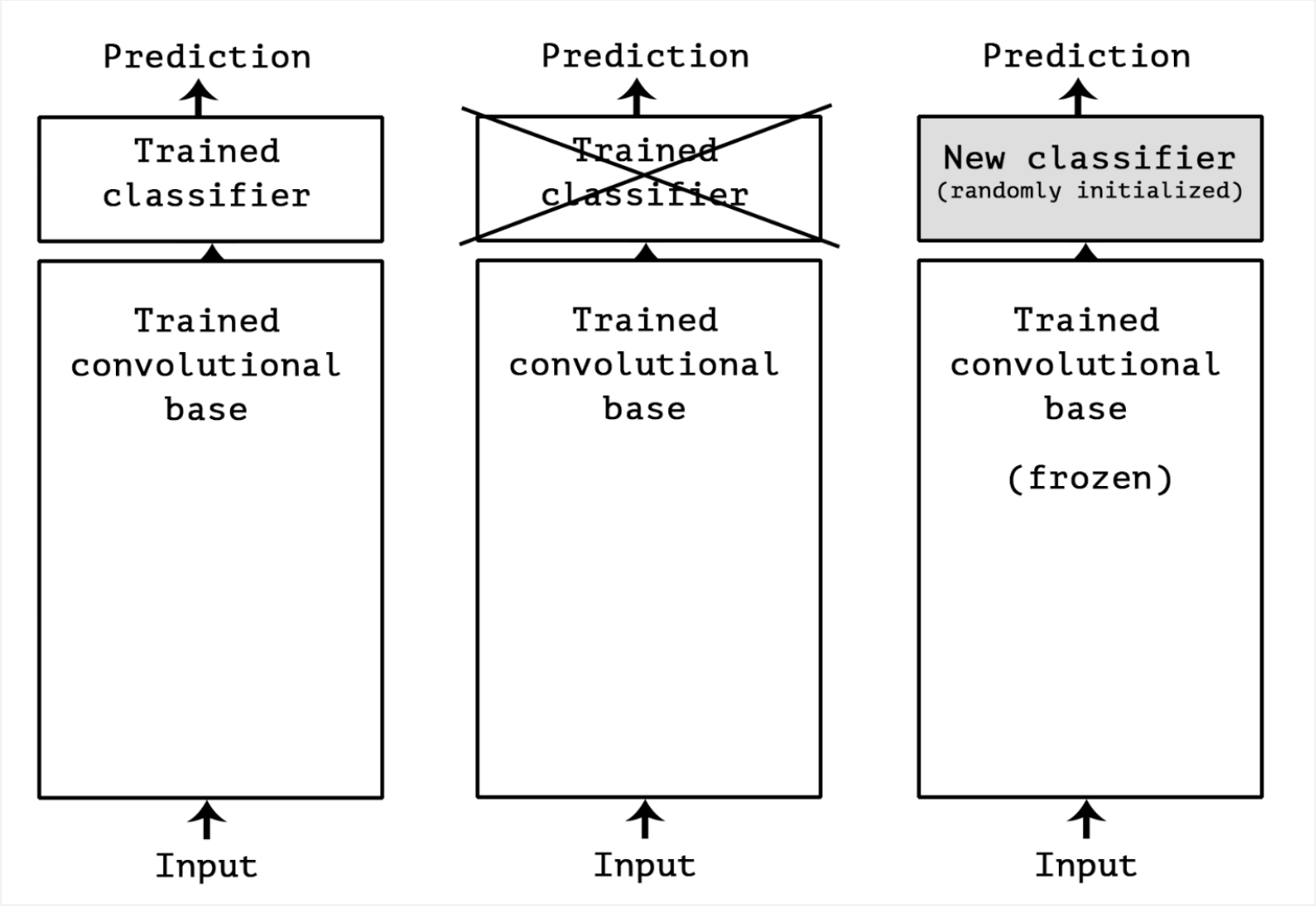
Pre trained model

- 새로운 문제가 원래 작업과 완전히 다른 클래스에 대한 것이더라도 이런 특성은 많은 컴퓨터 비전 문제에 유용합니다.
- 예를 들어 (대부분 동물이나 생활 용품으로 이루어진) **ImageNet 데이터셋**에 네트워크를 훈련합니다.
- 그 다음 이 네트워크를 이미지에서 가구 아이템을 식별하는 것 같은 다른 용도로 사용할 수 있습니다.
- 학습된 특성을 다른 문제에 적용할 수 있는 이런 **유연성**은 이전의 많은 얕은 학습 방법과 비교했을 때 **딥 러닝의 핵심 장점**입니다.
- 이런 방식으로 작은 데이터셋을 가진 문제에도 딥 러닝이 효율적으로 작동할 수 있습니다.

Pre trained model

- 여기에서는 (1.4백만 개의 레이블된 이미지와 1,000개의 클래스로 이루어진) ImageNet 데이터셋에서 훈련된 대규모 컨브넷을 사용해 보겠습니다.
- ImageNet 데이터셋은 다양한 종의 강아지와 고양이를 포함해 많은 동물들을 포함하고 있습니다.
- 그래서 강아지 vs. 고양이 분류 문제에 좋은 성능을 낼 것 같습니다.
- VGG16은 간단하고 ImageNet 데이터셋에 널리 사용되는 컨브넷 구조입니다.
- 이런 이름에는 VGG, ResNet, Inception, Inception-ResNet, Xception 등이 있습니다.

Pre trained model: feature extraction



Pre trained model: feature extraction

- 합성곱 층에 의해 **학습된 표현이 더 일반적**이어서 재사용 가능합니다.
- 컨브넷의 특성 맵은 사진에 대한 일반적인 컨셉의 존재 여부를 기록한 맵입니다. 그래서 주어진 컴퓨터 비전 문제에 상관없이 유용하게 사용할 수 있습니다.
- 하지만 **분류기에서 학습한 표현**은 모델이 **훈련된 클래스 집합에 특화**되어 있습니다.
- 분류기는 전체 사진에 어떤 클래스가 존재할 **확률에 관한 정보**만을 담고 있습니다.
- 더군다나 완전 연결 층에서 찾은 표현은 더 이상 입력 이미지에 있는 객체의 위치 정보를 가지고 있지 않습니다.
- 완전 연결 층들은 공간 개념을 제거하지만 합성곱의 특성 맵은 객체의 위치를 고려합니다. 객체의 위치가 중요한 문제라면 완전 연결 층에서 만든 특성은 크게 쓸모가 없습니다.

Pre trained model: feature extraction

- 특정 합성곱 층에서 **추출한 표현의 일반성(그리고 재사용성)**의 수준은 모델에 있는 층의 깊이에 달려 있습니다.
- 모델의 하위 층은 **(에지, 색깔, 질감 등과 같이) 지역적이고 매우 일반적인 특성** 맵을 추출합니다. 반면 **상위 층은 ('강아지 눈'이나 '고양이 귀'와 같이) 좀 더 추상적인 개념**을 추출합니다.
- 만약 새로운 데이터셋이 원본 모델이 훈련한 데이터셋과 많이 다르다면 전체 합성곱 기반층을 사용하는 것보다는 모델의 하위 층 몇 개만 특성 추출에 사용하는 것이 좋습니다.

1. VGG on small datasets

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

# 모든 이미지를 1/255로 스케일을 조정합니다
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

base_dir = '/content/gdrive/My Drive/datasets/cats_and_dogs_small'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')


train_generator = train_datagen.flow_from_directory(
        # 타깃 디렉터리
        train_dir,
        # 모든 이미지를 224 × 224 크기로 바꿉니다
        target_size=(224, 224),
        batch_size=20,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요합니다
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(224, 224),
        batch_size=20,
        class_mode='binary')
```

1. VGG on small datasets

```
#  This code cell shows how to utilize VGG16 model by combining Dense layer at the er
from keras.layers import Input, Dense, GlobalAveragePooling2D, Flatten
from keras.models import Model

from keras.applications.vgg16 import VGG16
from keras import layers
from keras import models

VGGNet = VGG16()

# VGGNet.summary()

# We will not update VGG pre-trained model, only added Dense layers will be trained f
for layer in VGGNet.layers:
  layer.trainable = False

vgg_maxpool5 = VGGNet.get_layer('block5_pool').output

Feature_Flatten = Flatten()(vgg_maxpool5)
dense = Dense(10, name='dense', activation='relu')(Feature_Flatten)
predictions = Dense(1, activation='sigmoid')(dense)
New_VGGmodel = Model(inputs=VGGNet.input, outputs=predictions)
New_VGGmodel.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur

New_VGGmodel.summary()
```

1. VGG on small datasets

```
#  This code cell shows how to utilize VGG16 model by combining Dense layer at the er
from keras.layers import Input, Dense, GlobalAveragePooling2D, Flatten
from keras.models import Model

from keras.applications.vgg16 import VGG16
from keras import layers
from keras import models

VGGNet = VGG16()

# VGGNet.summary()

# We will not update VGG pre-trained model, only added Dense layers will be trained f
for layer in VGGNet.layers:
  layer.trainable = False

vgg_maxpool5 = VGGNet.get_layer('block5_pool').output

Feature_Flatten = Flatten()(vgg_maxpool5)
dense = Dense(10, name='dense', activation='relu')(Feature_Flatten)
predictions = Dense(1, activation='sigmoid')(dense)
New_VGGmodel = Model(inputs=VGGNet.input, outputs=predictions)
New_VGGmodel.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur

New_VGGmodel.summary()
```

1. VGG on small datasets

```
history = New_VGGmodel.fit_generator(
      train_generator,
      steps_per_epoch=10,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

```
Epoch 1/30
10/10 [==============================] - 407s 41s/step - loss: 1.1114 - acc: 0.5750
Epoch 2/30
10/10 [==============================] - 7s 651ms/step - loss: 0.6297 - acc: 0.7000
Epoch 3/30
10/10 [==============================] - 68s 7s/step - loss: 0.5477 - acc: 0.7800 -
Epoch 4/30
10/10 [==============================] - 77s 8s/step - loss: 0.5195 - acc: 0.7650 -
Epoch 5/30
10/10 [==============================] - 80s 8s/step - loss: 0.5840 - acc: 0.7350 -
```

2. 사전 훈련된 Inception 네트워크 활용하기

2.Inception on small datasets

```
from keras.applications.inception_v3 import InceptionV3

Incep= InceptionV3(weights='imagenet',include_top = False,input_shape = (224, 224, 3)
for layer in Incep.layers:
  layer.trainable = False
```

2.Inception on small datasets

```
incep_maxpool4 = Incep.layers[-2].output

Feature_Flatten = Flatten()(incep_maxpool4)
dense = Dense(10, name = 'dense', activation = 'relu')(Feature_Flatten)
predictions = Dense(1, activation='sigmoid')(dense)
New_incep = Model(inputs=Incep.input, outputs = predictions)
New_incep.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy

New_incep.summary()
```

3. Cifar-10 Dataset Challenge

3. CIFAR10 small image classification

Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

Usage:

```
from keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

- **Returns:**
  - 2 tuples:
    - **x_train, x_test**: uint8 array of RGB image data with shape (num_samples, 3, 32, 32) or (num_samples, 32, 32, 3) based on the `image_data_format` backend setting of either `channels_first` or `channels_last` respectively.
    - **y_train, y_test**: uint8 array of category labels (integers in range 0-9) with shape (num_samples,).