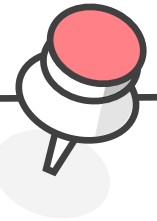


합성곱 신경망을 사용한 컴퓨터 비전(2)

By Hands-On

경제학과 하지민

케라스에서 제공하는 사전 훈련된 모델 사용하기

keras.application 패키지에 준비되어 있는 사전 훈련된 모델을 코드 한 줄로 불러 올 수 있음

ResNet-50 모델 로드 한 코드

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102973440/102967424 [==============================] - 1s 0us/step
102981632/102967424 [==============================] - 1s 0us/step
```

케라스에서 제공하는 사전 훈련된 모델 사용하기

tf. Image.resize() 이용한 이미지 크기 변환

가로세로 비율을 유지 하지 않음

```
images_resized = tf.image.resize(images, [224, 224])
plot_color_image(images_resized[0])
plt.show()
```

케라스에서 제공하는 사전 훈련된 모델 사용하기

tf. Image.resize_with_pad() 이용한 이미지 크기 변환

왜곡없이 종횡비를 동일하게 유지하여 이미지를 대상 폭과 높이로 조정
대상 치수가 이미지 치수와 일치하지 않으면 이미지 크기가 조정 된 다음 요청 된 치수와 일치하도록 0으로 채워짐

```
images_resized = tf.image.resize_with_pad(images, 224, 224, antialias=True)
plot_color_image(images_resized[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

케라스에서 제공하는 사전 훈련된 모델 사용하기

tf. Image.resize_with_crop_or_pad() 이용한 이미지 크기 변환

위 두 가지 동시에 수행
이미지를 중앙에서 자르거나 0으로 균등하게 채워 이미지를 대상 폭과 높이로 조정

```
images_resized = tf.image.resize_with_crop_or_pad(images, 224, 224)
plot_color_image(images_resized[0])
plt.show()
```

케라스에서 제공하는 사전 훈련된 모델 사용하기

```
china_box = [0, 0.03, 1, 0.68]
flower_box = [0.19, 0.26, 0.86, 0.7]
images_resized = tf.image.crop_and_resize(images, [china_box, flower_box], [0, 1], [224, 224])
plot_color_image(images_resized[0])
plt.show()
plot_color_image(images_resized[1])
plt.show()
```

케라스에서 제공하는 사전 훈련된 모델 사용하기

사전훈련된 모델은 이미지가 적절한 방식으로 전처리 되었다고 가정

이 함수는 픽셀값이 0에서 255사이라고 가정

Image_resized에 255 곱한 후 예측 수행

```
[ ]  inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)
     Y_proba = model.predict(inputs)
```

```
Y_proba.shape
```

```
(2, 1000)
```

케라스에서 제공하는 사전 훈련된 모델 사용하기

```
top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
for image_index in range(len(images)):
    print("Image #{}".format(image_index))
    for class_id, name, y_proba in top_K[image_index]:
        print("  {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
    print()
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
49152/35363 [==========================================] - 0s 0us/step
Image #0
  n03877845 - palace       43.39%
  n02825657 - bell_cote    43.07%
  n03781244 - monastery    11.70%

Image #1
  n04522168 - vase         53.96%
  n07930864 - cup          9.52%
  n11939491 - daisy        4.97%
```

사전훈련된 모델을 사용한 전이 학습

텐서플로 데이터셋 활용해 데이터 적재

```
[ ] import tensorflow_datasets as tfds

    dataset, info = tfds.load("tf_flowers", as_supervised=True, with_info=True)
```

```
Downloading and preparing dataset tf_flowers/3.0.1 (download: 218.21 MiB, generated: 221.83 MiB, total: 440.05 MiB) to /root/tensorflow_datasets/tf_flowers/3.0.1...
WARNING:absl:Dataset tf_flowers is hosted on GCS. It will automatically be downloaded to your
local data directory. If you'd instead prefer to read directly from our public
GCS bucket (recommended if you're running on GCP), you can instead pass
`try_gcs=True` to `tfds.load` or set `data_dir=gs://tfds-data/datasets`.

Dl Completed...: 100% 5/5 [00:02<00:00, 1.54 file/s]

Dataset tf_flowers downloaded and prepared to /root/tensorflow_datasets/tf_flowers/3.0.1. Subsequent calls will reuse this data.
```

```
[ ] info.splits
```

```
{'train': <tfds.core.SplitInfo num_examples=3670>}
```

```
[ ] info.splits["train"]
```

```
<tfds.core.SplitInfo num_examples=3670>
```

```
[ ] class_names = info.features["label"].names
    class_names
```

```
['dandelion', 'daisy', 'tulips', 'sunflowers', 'roses']
```

```
[ ] n_classes = info.features["label"].num_classes
```

```
[ ] dataset_size = info.splits["train"].num_examples
    dataset_size
```

```
3670
```

사전훈련된 모델을 사용한 전이 학습

데이터 예시

사전훈련된 모델을 사용한 전이 학습

이 데이터셋에는 'train'셋만 있기 때문에 테스트 세트나 검증 세트를 나누어야 함

TFDS의 split API는 책이 출간된 후에 바뀌었음. 새로운 split API(S3 슬라이싱 API)는 사용하기 훨씬 간단

```
test_set_raw, valid_set_raw, train_set_raw = tfds.load(
    "tf_flowers",
    split=["train[:10%]", "train[10%:25%]", "train[25%:]"],
    as_supervised=True)
```

사전훈련된 모델을 사용한 전이 학습

앞에 했던 것과 마찬가지로 resize 후 preprocess_input 활용하여 기본 전처리

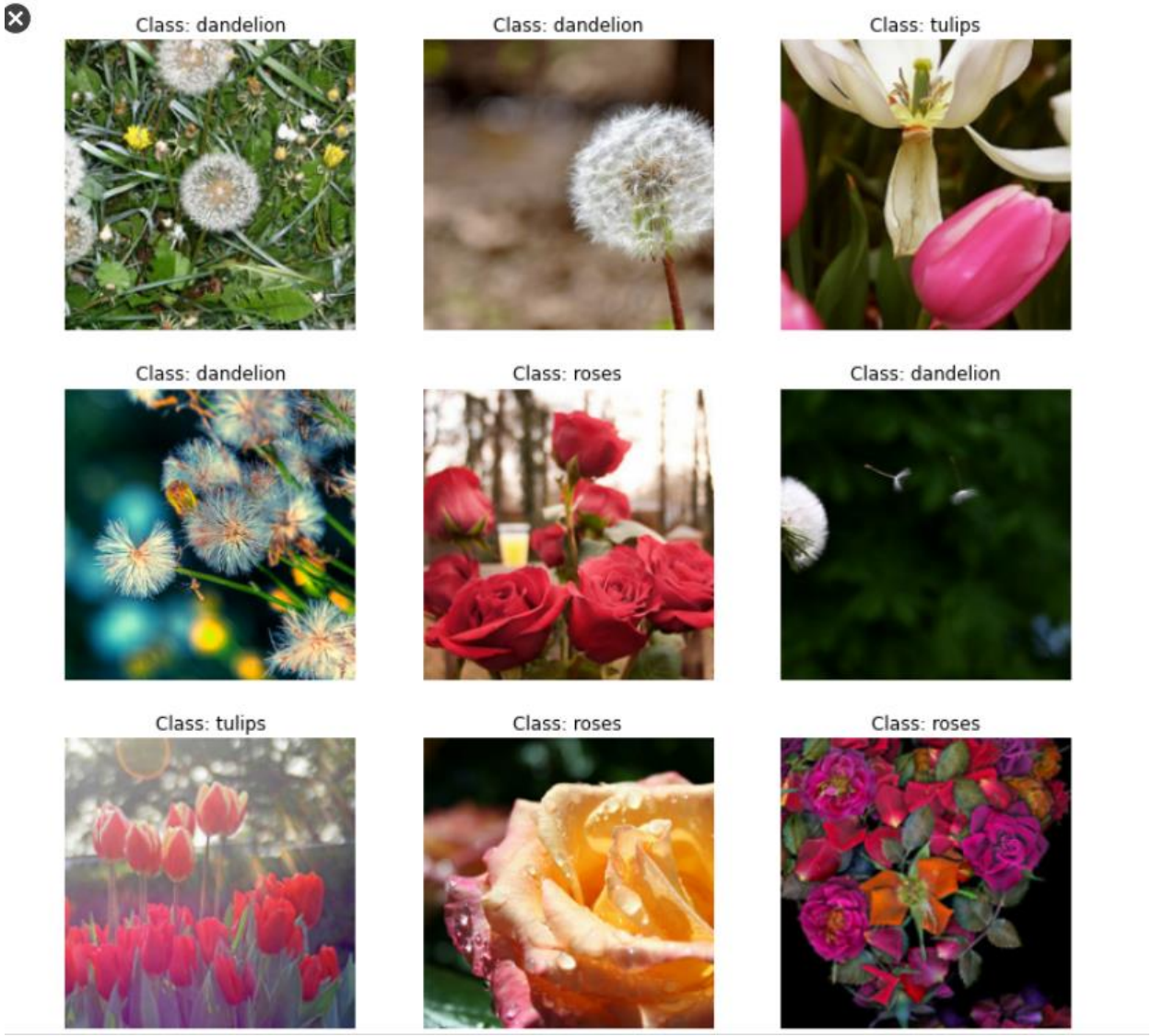
```
[ ]  def preprocess(image, label):
         resized_image = tf.image.resize(image, [224, 224])
         final_image = keras.applications.xception.preprocess_input(resized_image)
         return final_image, label
```

훈련 세트를 섞고 전처리 함수를 3개의 데이터셋에 모두 적용
그 다음 배치 크기를 지정하고 프리패치를 적용

```
batch_size = 32
train_set = train_set_raw.shuffle(1000).repeat()
train_set = train_set.map(partial(preprocess, randomize=True)).batch(batch_size).prefetch(1)
valid_set = valid_set_raw.map(preprocess).batch(batch_size).prefetch(1)
test_set = test_set_raw.map(preprocess).batch(batch_size).prefetch(1)
```

사전훈련된 모델을 사용한 전이 학습

전처리 끝난 데이터



사전훈련된 모델을 사용한 전이 학습

이미지넷에서 사전훈련된 Xception 모델을 로드

include_top = False로 지정하여 네트워크 최상층에 해당하는 전역 평균 풀링 층과 밀집 출력 층을 제외

이 기반 모델 출력 바탕으로 새로운 전역 풀링 층을 추가

그 뒤에 클래스마다 하나의 유닛과 소프트맥스 활성화 함수를 가진 밀집 출력층 설치

마지막으로 케라스의 Model 객체 생성

```
base_model = keras.applications.xception.Xception(weights="imagenet",
                                                  include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(n_classes, activation="softmax")(avg)
model = keras.models.Model(inputs=base_model.input, outputs=output)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
```

사전훈련된 모델을 사용한 전이 학습

모델 컴파일 후 훈련

```
[ ]  for layer in base_model.layers:
         layer.trainable = False

     optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9, decay=0.01)
     model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
                   metrics=["accuracy"])
     history = model.fit(train_set,
                         steps_per_epoch=int(0.75 * dataset_size / batch_size),
                         validation_data=valid_set,
                         validation_steps=int(0.15 * dataset_size / batch_size),
                         epochs=5)
```

```
Epoch 1/5
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/optimizer_v2.py:356: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
86/86 [==============================] - 38s 371ms/step - loss: 1.4720 - accuracy: 0.7925 - val_loss: 1.4752 - val_accuracy: 0.8070
Epoch 2/5
86/86 [==============================] - 31s 360ms/step - loss: 0.6576 - accuracy: 0.8914 - val_loss: 0.9815 - val_accuracy: 0.8732
Epoch 3/5
86/86 [==============================] - 31s 359ms/step - loss: 0.3623 - accuracy: 0.9248 - val_loss: 0.8554 - val_accuracy: 0.8732
Epoch 4/5
86/86 [==============================] - 31s 359ms/step - loss: 0.2582 - accuracy: 0.9335 - val_loss: 0.7072 - val_accuracy: 0.8842
Epoch 5/5
86/86 [==============================] - 31s 359ms/step - loss: 0.2029 - accuracy: 0.9440 - val_loss: 0.7266 - val_accuracy: 0.8732
```

분류와 위치 추정

사진에서 물체의 위치를 추정하는 것은 회귀 작업으로 나타낼 수 있음

물체 주위의 바운딩 박스를 예측하는 일방적인 방법은 물체 중심의 수평, 수직 좌표와 높이, 너비를 예측하는 것

네 개의 유닛을 가진 두번째 밀집층을 추가하고 MSE 손실을 사용해 훈련

```
[ ]  base_model = keras.applications.xception.Xception(weights="imagenet",
                                                       include_top=False)
     avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
     class_output = keras.layers.Dense(n_classes, activation="softmax")(avg)
     loc_output = keras.layers.Dense(4)(avg)
     model = keras.models.Model(inputs=base_model.input,
                                outputs=[class_output, loc_output])
     model.compile(loss=["sparse_categorical_crossentropy", "mse"],
                   loss_weights=[0.8, 0.2], # 어떤 것을 중요하게 생각하느냐에 따라
                   optimizer=optimizer, metrics=["accuracy"])
```

분류와 위치 추정

사진에서 물체의 위치를 추정하는 것은 회귀 작업으로 나타낼 수 있음

물체 주위의 바운딩 박스를 예측하는 일방적인 방법은 물체 중심의 수평, 수직 좌표와 높이, 너비를 예측하는 것

네 개의 유닛을 가진 두번째 밀집층을 추가하고 MSE 손실을 사용해 훈련

```
[ ]  base_model = keras.applications.xception.Xception(weights="imagenet",
                                                       include_top=False)
     avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
     class_output = keras.layers.Dense(n_classes, activation="softmax")(avg)
     loc_output = keras.layers.Dense(4)(avg)
     model = keras.models.Model(inputs=base_model.input,
                                outputs=[class_output, loc_output])
     model.compile(loss=["sparse_categorical_crossentropy", "mse"],
                   loss_weights=[0.8, 0.2], # 어떤 것을 중요하게 생각하느냐에 따라
                   optimizer=optimizer, metrics=["accuracy"])
```

바운딩 박스 : 오브젝트의 형태를 모두 포함할 수 있는 최소 크기의 박스

분류와 위치 추정

바운딩 박스 직접 추가 후 훈련

```
def add_random_bounding_boxes(images, labels):
    fake_bboxes = tf.random.uniform([tf.shape(images)[0], 4])
    return images, (labels, fake_bboxes)

fake_train_set = train_set.take(5).repeat(2).map(add_random_bounding_boxes)
```

```
model.fit(fake_train_set, steps_per_epoch=5, epochs=2)
```

```
Epoch 1/2
5/5 [==============================] - 11s 1s/step - loss: 1.3813 - dense_5_loss: 1.6242 - dense_6_loss: 0.4098 - dense_5_accuracy: 0.1562 - dense_6_accuracy: 0.2188
Epoch 2/2
5/5 [==============================] - 7s 1s/step - loss: 1.3013 - dense_5_loss: 1.5382 - dense_6_loss: 0.3537 - dense_5_accuracy: 0.3187 - dense_6_accuracy: 0.2625
<keras.callbacks.History at 0x7f289925d9d0>
```

지금은 간단하게 만들어져 있지만 원래 가장 비용이 많이 들고 어려운 작업 중 하나라고 함

바운딩 박스에 널리 사용되는 지표로 IoU 존재

예측한 바운딩 박스와 타깃 바운딩 박스 사이에 중첩되는 영역을 전체 영역으로 나눈 것

객체 탐지

하나의 이미지에서 여러 물체를 분류하고 위치를 추정하는 작업

주로 이미지를 슬라이딩 시키며 탐지

중복되거나 존재여부 점수가 낮은 바운딩 박스를 제거하는 과정 거침

완전 합성곱 신경망을 사용하면 CNN을 훨씬 빠르게 이미지에 슬라이딩 시킬 수 있음

객체 탐지

YOLO(You Only Look Once)

딥러닝을 통한 객체 탐지 모델은 크게 R-CNN, SSD, YOLO가 존재

객체탐지 모델들이 우선적으로 당면한 문제 중 하나는, 실제 서비스를 할 수 있을 만큼 탐지 속도와 정확도를 올려야 한다는 것

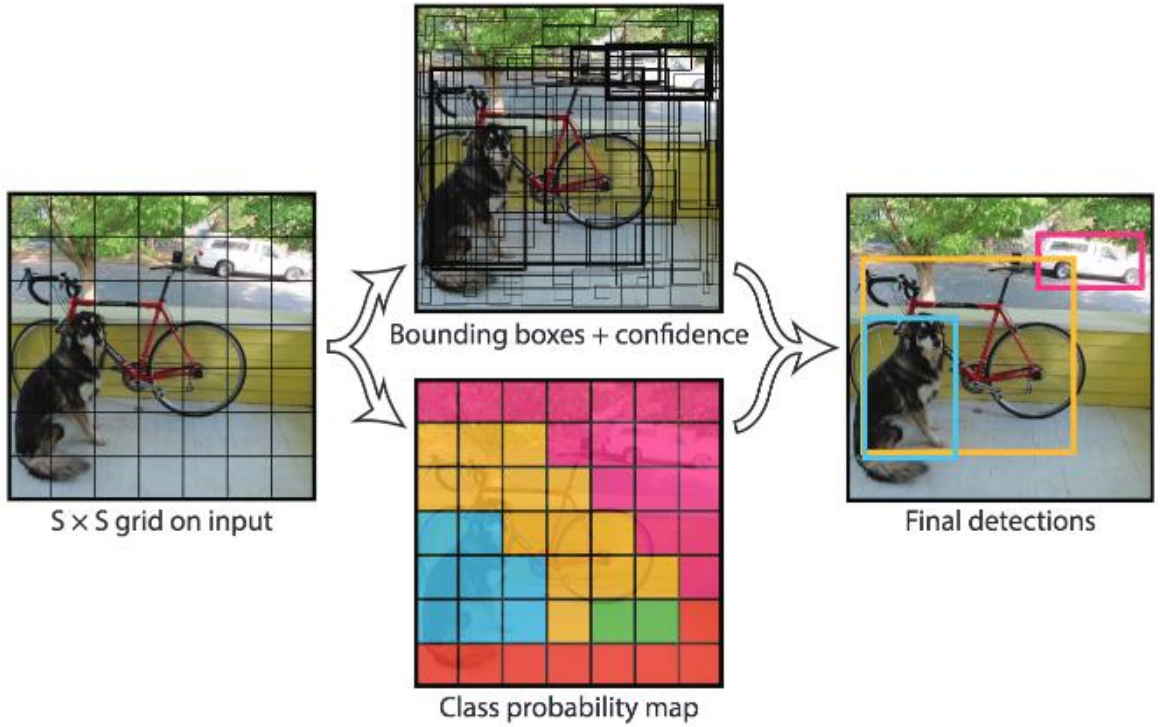
정확도와 탐지 속도(mAP : mean Average Precision)는 trade-off 관계
탐지 속도가 높으면 그만큼 정확도는 낮아지고, 정확도가 낮아지면 그만큼 탐지 속도가 올라감
YOLO는 괜찮은 수준의 mAP와 FPS를 가짐. 개인 레벨에서 시범적으로 사용해보기에는 최적의 모델

YOLO의 속도 상승의 비결은,

기존에 1) region proposal 2) classification 이렇게 두 단계로 나누어서 진행하던 방식에서 region proposal 단계를 제거하고 한번에 Object Detection을 수행하는 구조를 갖는다는 점

객체 탐지

YOLO(You Only Look Once) 의 원리



Example of application. The input image is divided into an SxS grid, B bounding boxes are predicted (regression) and a class is predicted among C classes (classification) over the most confident ones. Source: J. Redmon and al. (2016)

객체 탐지

YOLO(You Only Look Once) 의 원리

먼저 이미지를 SxS 그리드 영역으로 나눔
이 그리드를 기준으로 객체 후보군의 위치를 나누는 것
저 그리드가 연산량을 줄이는 역할을 하며, 그리드를 촘촘하게 만들면 보다 정교하게 b-box가 예측되지만, 그리드를 적당히 설정하면, 검색 구역이 줄어들며 후보군 조금 허술해지겠지만 연산량이 줄어듦

하나의 이미지에서 두개의 모델 프로세스로 나뉨
위에는 그리드를 기반으로 객체를 탐지하는 b-box 탐지 모델로, IoU를 손실함수 기준으로 사용
아래는 동시에 각 그리드별로 클래스 분류를 수행하며 이것이 개인지 자전거인지 배경인지를 판단

위에는 b-box가 출력되는데, 아래로는 각 그리드별로 클래스 분류를 진행
클래스 분류 자체야 오히려 연산량이 많아진 것일수도 있는데, 직렬처리를 병렬로 바꾼 것으로 단점을 장점으로 만들 수 있음

시멘틱 분할

픽셀이 속한 객체의 클래스로 분류됨

이미지가 일반적으로 CNN을 통과할 때 위치 정보 손실

해상도를 늘리는 업샘플링층을 추가

주로 전치 합성곱을 사용