



CNN: 사전학습모형 사용하기

Sang Yup Lee



Pre-trained model 사용 방법



Pre-trained models

- Tensorflow

- <https://keras.io/api/applications/>

- PyTorch

- <https://pytorch.org/vision/stable/models.html>



Pre-trained models

- 사전학습모형을 이용한 이미지 분류
- 주요 방법 2가지
 - 방법1: pre-trained 모형의 결과값을 그대로 사용하여 예측
 - 원래 모형의 구조를 그대로 사용
 - 원래의 학습데이터를 학습에서 얻어진 파라미터의 값을 그대로 사용
 - 새롭게 풀고자 하는 문제가 pre-trained 모형이 적용된 문제와 동일하거나 유사도가 큰 경우 사용
 - 예) ImageNet 1000
 - Classes의 많은 부분이 동물에 대한 것 → 동물 사진을 분류하고자 하는 경우 ImageNet 사전학습모형을 그대로 사용할 수 있음



Pre-trained models

- 주요 방법 (cont'd)
 - 방법2: Transfer learning (전이 학습)
 - Pre-trained 모형의 일부만을 그대로 사용하고 나머지 일부를 변형하여 사용하기
 - 모형의 구조를 변경하기: 기존의 층 제거 또는 새로운 층을 추가하기
 - 일부 또는 전체 파라미터를 다시 학습하여 사용하기
 - 이를 위해서는 새로운 학습 데이터가 필요
 - 하지만, 사전학습모형을 활용하기 때문에 그렇게 많은 학습데이터가 필요하지는 않음



Pre-trained 모형의 결과를 그대로 사용하기

Pre-trained model 사용하기 1

- Pre-trained model 결과 그대로 사용하여 새로운 이미지 분류하기
 - 이는 pre-trained model의 구조를 변경하지 않고, 특정 학습 데이터를 학습하여 가지고 있는 weights의 값을 그대로 사용하는 것을 의미
 - 이러한 weights값은 원래의 문제를 풀기 위해 optimal하게 계산된 값들임
 - 예) ImageNet 문제: 분류 문제 (1000개의 classes)
 - 따라서 우리가 풀고자 하는 문제가 원래의 문제와 유사하거나 동일한 경우에 사용 적합

풀고자하는 문제: 아래 사진 속의 동물이 무엇인지 맞추는 문제





Pre-trained model 사용하기 1

- 구체적 예: VGG16 & ResNet50
 - 이 둘 모두 ImageNet 학습 데이터를 사용해서 학습
 - 1000개의 image classes를 가지고 있음
 - Example 1) VGG16 사용하기
 - 파이썬 코드: VGG16_example.ipynb
 - Example 2) ResNet50 사용하기
 - 파이썬 코드: ResNet50_example.ipynb

Pre-trained model 사용하기 1

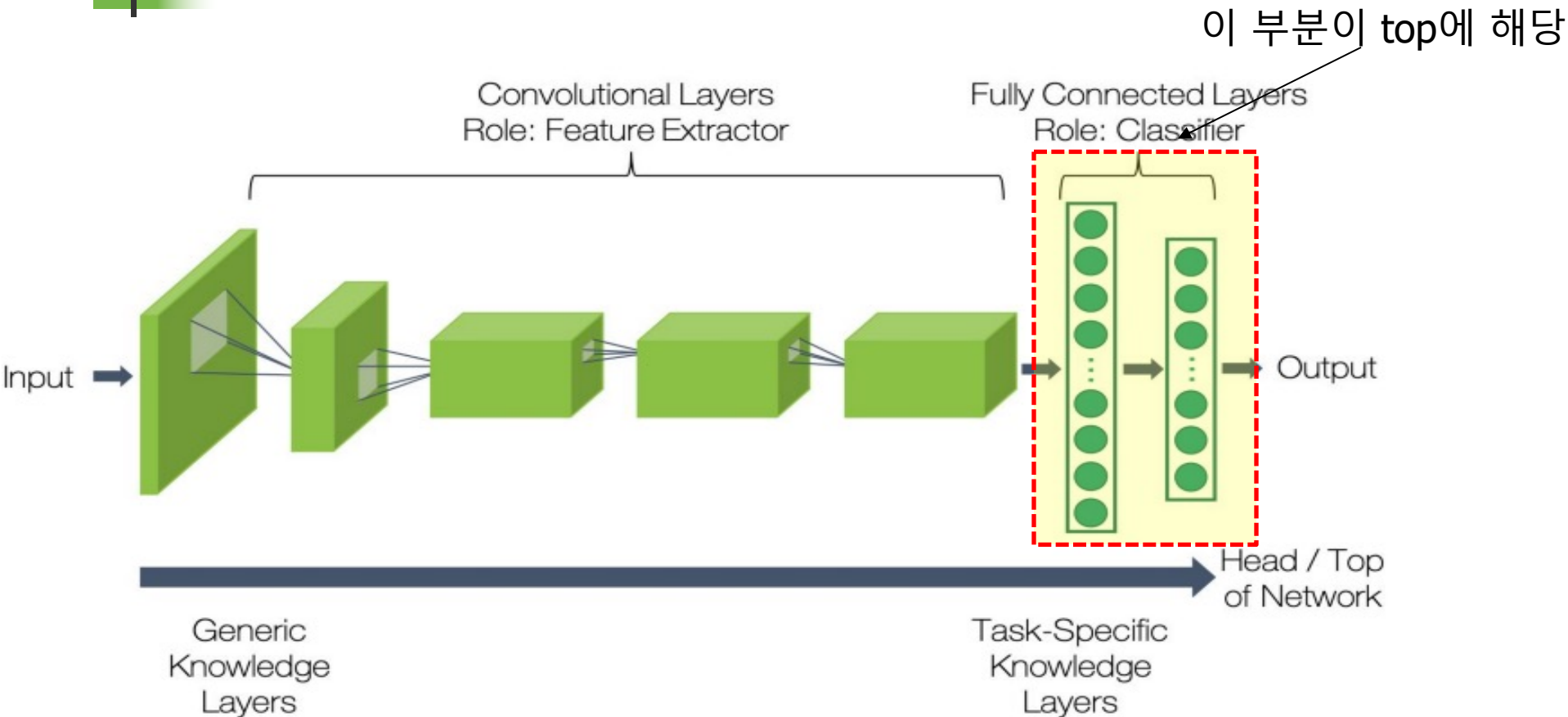
■ VGG16_example.ipynb 코드 보기

```
from tensorflow.keras.applications.vgg16 import VGG16
model = VGG16(weights='imagenet', include_top=True)
model.summary()
```

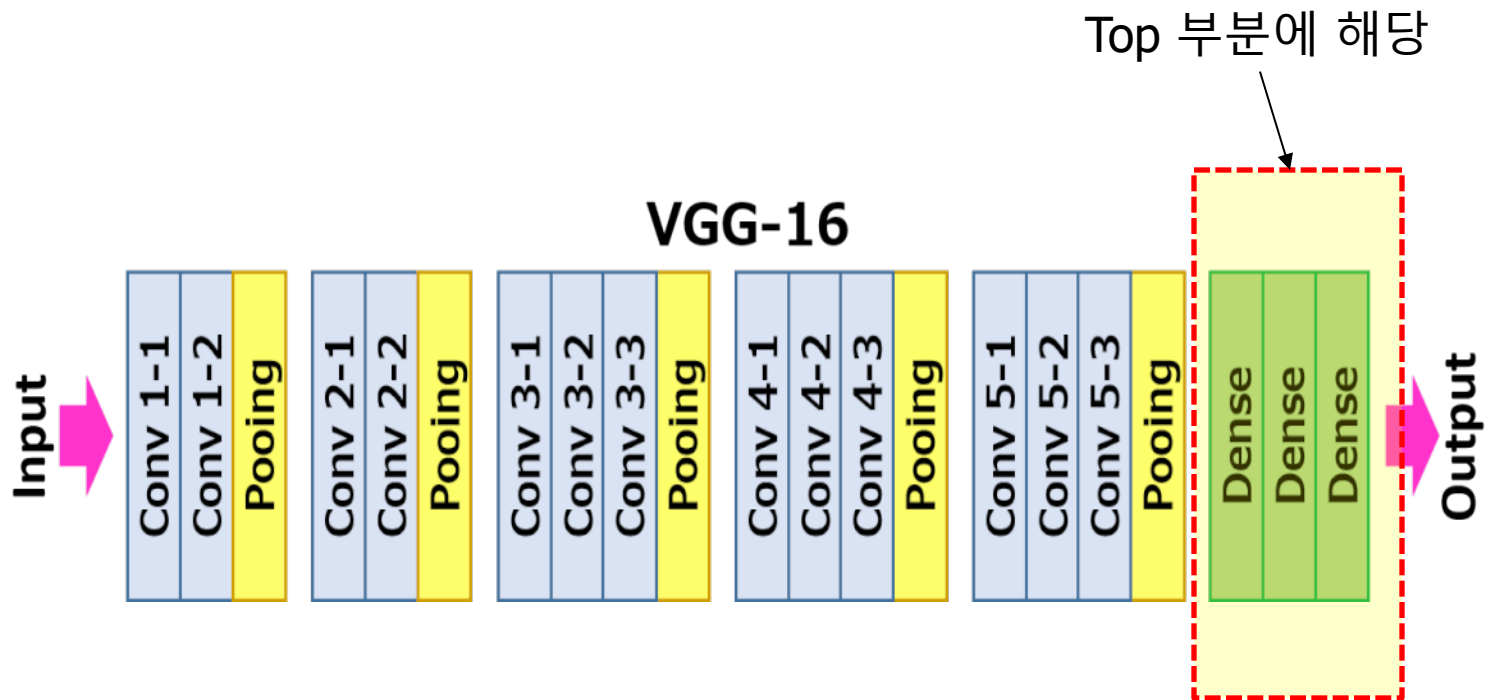
imagenet 데이터를 사용해 학습한 결과의 파라미터값을 사용한다는 뜻
또는 None (random initialization)

include_top: whether to include the 3 fully-connected
layers at the top of the network.
만약 해당 dense layers를 포함하지 않고자 하는
경우에는 값을 False로 설정

CNN 기반 모형의 일반적 구조



CNN 기반 모형의 일반적 구조



Top 부분을 제거하기 위해서는 `include_top=False` 설정

VGG16 이용하기

- Image 준비하기
 - 여기서는 하나의 이미지를 분류
 - Pre-trained 모델을 사용해서 새로운 이미지를 분류하기 위해서는 데이터를 pre-trained 모델에 맞게 준비하는게 필요
 - 보통, 특정한 크기의 정사각형 형태로 이미지를 입력 받음
 - VGG16의 경우는 224x224 정사각형 컬러이미지

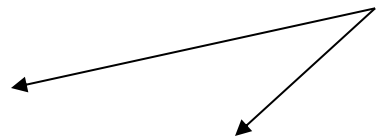




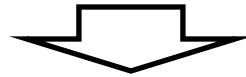
VGG16 이용하기

- 이미지를 특정 크기의 정사각형으로 준비하기
 - 주요 과정
 - 정사각형으로 만들기
 - (원하는 크기로) 이미지 축소/확대 (resize) 하기
- Image를 정사각형으로 만들기
 - 주요 방법 3가지
 - (가운데를 중심으로) Cropping
 - Warping: 가로와 세로를 다른 비율로 확대 또는 축소 ⇒ 이미지 왜곡 발생
 - Padding

동시에 수행도 가능



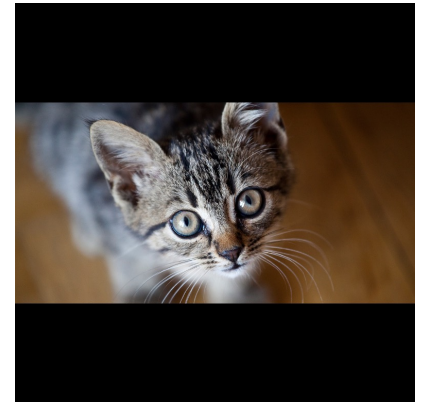
정사각형으로 만들기



center cropping



warping



padding

See `Image_preprocessing.ipynb`



Cropping 관련

- How to crop?
 - 분석하고자 하는 물체가 중간에 위치하는 경우에는, center cropping is often the best strategy.
 - 하지만, 이미지의 특성에 따라서 다른 방법 사용 필요
 - 예, a tall image 이러한 경우에는 padding을 해서 정사각형으로 만드는 것이 더 바람직
 - 특정한 물체만 crop하고자 하는 경우, SSD, YOLO 등을 이용해서 bounding box를 찾고 Open CV 등을 이용해서 crop하는 방법



VGG16 이용하기

■ Cropping

- 원본 이미지의 가로, 세로 길이 중 더 짧은 길이를 이용해서 정사각형으로 만든다.
- 이를 위해 `crop()` 를 이용 (아래는 center cropping)

```
img = Image.open('cat.jpg')
w, h = img.size
s = min(w, h)
y = (h - s) // 2
x = (w - s) // 2
#print(w, h, x, y, s)
img = img.crop((x, y, x+s, y+s))
# 4-tuple defining the left, upper, right, and lower pixel coordinate
imshow(np.asarray(img))
```

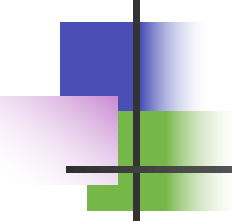

VGG16 이용하기

- Image resize (확대/축소하기)
 - 정사각형으로 crop을 한 다음에 (VGG16) 모형에서 입력받는 크기로 resize
 - 이를 위해 모형에 입력되는 이미지의 크기를 확인

```
In: model.layers[0].input_shape  
Out: [(None, 224, 224, 3)]
```

224x224의 칼라 이미지를
입력받는다는 것을 의미

```
target_size = 224  
img = img.resize((target_size, target_size))
```



VGG16 이용하기

- Image data를 array로 변환
 - 이미지를 array 형태로 변환하는 과정이 필요. 이는 CNN 모형에 입력되는 형태는 array 이기 때문.

```
np_img = image.img_to_array(img)
np_img.shape # (224, 224, 3)
```

- 4D array 형태로 차원을 확장해 주는 것이 필요
 - 원래의 모형은 3차원의 이미지를 여러개 입력 받기 때문 즉, 4차원 array로 데이터가 구성

```
img_batch = np.expand_dims(np_img, axis=0)
img_batch.shape
Out: (1, 224, 224, 3)
```



VGG16 이용하기

- Feature normalization
 - `img_batch`의 각 원소값은 0 ~ 255 사이의 숫자로 구성
 - 학습의 속도와 모형의 성능을 높이기 위해서 normalization 필요
 - 이를 위해 `preprocess_input()` 사용
 - 모형마다 `preprocess_input()` 함수의 역할의 다름
 - VGG16의 경우, 원래의 값들이 0을 기준으로 centering



VGG16 이용하기

- 예측하기

- predict() 함수를 이용
- 별도의 학습이 필요 없음
 - Why? 이미 학습이 된 가중치를 사용하기 때문

```
In: features = model.predict(pre_processed)
In: features.shape
Out: (1, 1000)
```

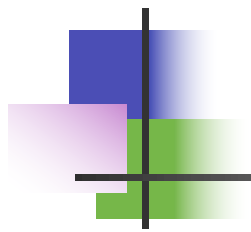
- 예측 확률이 높은 상위 k 개의 결과 보기

```
decode_predictions(features, top=5)
```



ResNet 이용하기

- ResNet50_example.ipynb
- 이 코드에는 cropping 없이 resize만 해줌
 - 이미지 왜곡이 어느정도 발생할 수 있음
- 나머지 과정은 거의 동일



Transfer Learning (전이학습)



Transfer learning

- What is it?
 - 사전학습모형의 구조를 변경하거나 파라미터의 일부(또는 전체)를 새롭게 학습하여 사용하는 방법

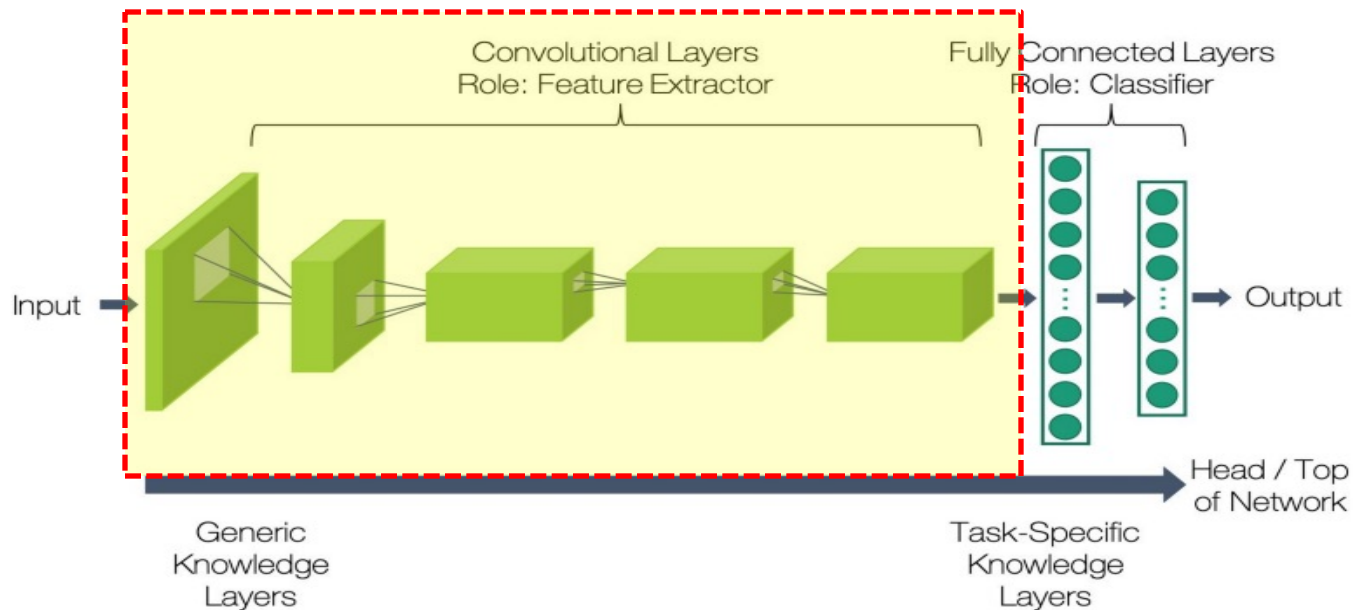


Transfer learning

- 사전학습모형의 구조를 변경하거나 파라미터의 일부를 새롭게 학습하여 사용하는 방법
- 주요 방법
 - 방법1: 사전학습모형이 출력하는 값을 features로 사용해서 전통적인 기계학습 알고리즘 (예, Logistic regression, SVM, DT 기반 ensemble 방법 (Light GBM, XGBoost, CatBoost 등)) 사용 (최근에는 많이 사용되지는 않음)
 - 방법2: 기존 모형의 일부 layer를 제거 (보통 top layer 제거) + 다른 레이어를 추가하여 분류하기
 - 방법3: Fine tuning
 - 사전학습모형이 가지고 있는 파라미터의 일부 또는 전체를 새로운 학습데이터를 이용해서 새롭게 학습시키는 것
 - 사전학습모형을 이용하는 경우, 초기값이 사전학습모형이 가지고 있는 최적값이 됨

Transfer learning

- 방법1: Features from convolutional layers+ 전통적인 기계학습 분류 모형
 - Pre-trained model이 feature extractor의 역할을 한다고 생각할 수 있음



하지만 이러한 경우, pre-trained 모형이 원래 적용된 문제와 우리가 풀고자 하는 문제와 다른 경우, 추출된 features가 우리가 풀고자 하는 문제와 관련된 각 이미지의 특성을 제대로 반영하지 못하고 있을 수 있다.

Transfer learning: Features extraction

- Example1: Inception_ML-classifiers.ipynb
 - 풀고자 하는 문제: 사진 속의 동물이 강아지인지 고양이인지를 맞추는 문제
 - 준비사항: 자체적으로 구축한 학습 데이터 필요
 - 학습데이터를 클래스(label)에 따라 별도의 폴더의 저장해야 함

cats

dogs

- 이미지의 크기를 299x299로 불러옴 (warping 방법)

Transfer learning: Features extraction

- Inception_ML-classifiers.ipynb (cont'd)
 - 마지막 출력층 (출력노드의수 = 1000) 바로 직전의 층 (avg_pool)의 출력값을 각 이미지의 feature 정보로 사용 (다른 층의 값을 이용하는 것도 가능)
 - avg_pool층이 출력하는 값의 수 = 2048 (즉, 원소가 2048개인 벡터라고 생각할 수 있음)
 - 이는 각 이미지의 특성을 2048의 (독립)변수를 가지고 표현했다고 생각할 수 있음
 - 이러한 2048개의 feature 정보를 독립변수 정보로 하여 전통적인 기계학습 classifier 적용

Transfer learning: Features extraction

- Inception_ML-classifiers.ipynb (cont'd)
 - 종속변수 생성
 - 'cats' 폴더에 있는 이미지의 종속변수 값을 1로 coding (즉, 정답이 고양이인 경우 $y = 1$), 'dogs' 폴더에 있는 이미지의 종속변수 값을 0으로 coding
 - 정답 데이터를 학습 데이터와 평가 데이터로 구분
 - 이를 위해 sklearn 모듈에서 제공하는 train_test_split() 함수 사용

```
labels = [1] * 1000 + [0] * 1000 # 종속변수 생성
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(vectors, labels, test_size=0.2)
```

Transfer learning: Features extraction



- Inception_ML-classifiers.ipynb (cont'd)
 - 전통적인 기계학습 분류기
 - Logistic regression model
 - Support vector machine
 - Ensemble methods
 - XGBoost
 - Gradient Boosting

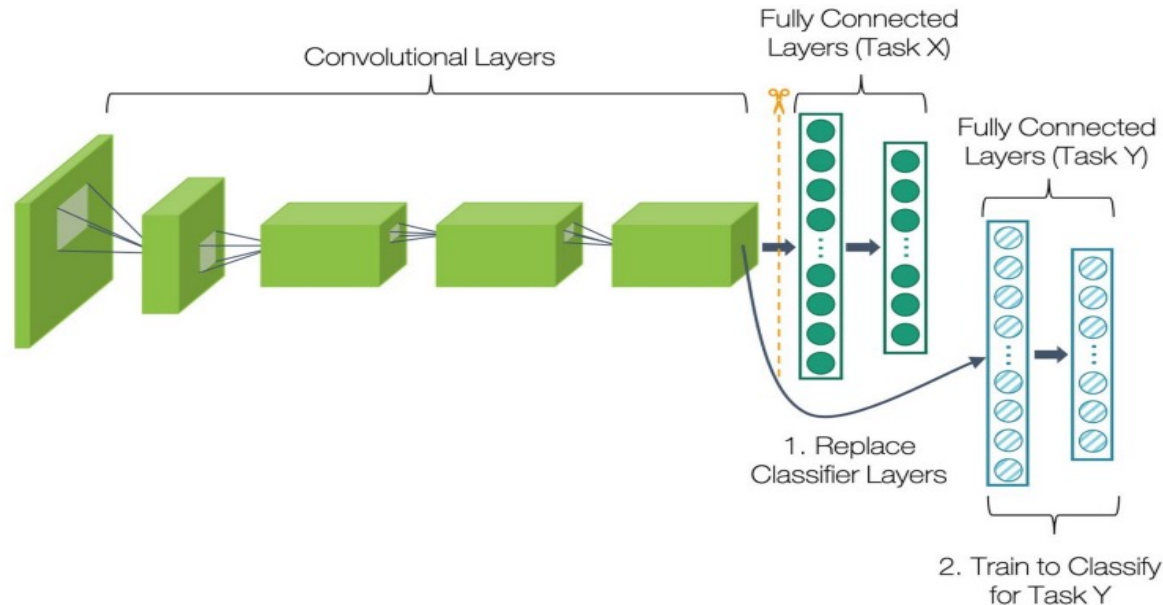
Transfer learning: Features extraction



- Example2: VGG16_ML-classifiers.ipynb
 - 전반적인 과정은 InceptionV3를 사용하는 경우와 유사
 - 이번에도 최종 출력층 바로 직전의 층에서 출력하는 값을 각 이미지의 feature 정보로 사용
 - 바로 직전 층: fc2 (4096의 값을 출력)

Transfer learning

- 방법2: 기존 모형의 일부 제거 (보통은 top layer를 제거) + 다른 레이어를 추가하여 분류하기



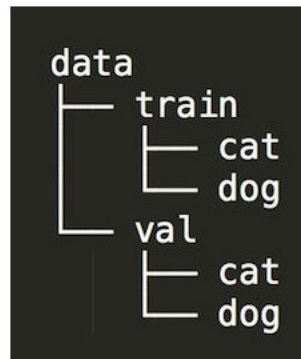


Transfer learning

- 기존 모형의 top layer를 제거 + 다른 레이어를 추가하여 분류하기
 - Python code
 - Transfer_learning_MobileNet.ipynb
 - Transfer_learning_VGG16.ipynb

Transfer learning

- 학습 데이터 준비하기
 - 새롭게 추가된 layer에 존재하는 파라미터를 새로운 학습 데이터를 이용해서 학습하는게 필요
 - 학습 데이터와 평가 데이터 (경우에 따라서는 검증 데이터셋)을 별도의 폴더에 저장
 - 각 폴더안에 이미지들을 클래스별로 별도의 폴더에 저장 (cats_and_dogs_small 폴더 참고)





Data augmentation

- 새로운 층을 추가하는 경우
 - 새롭게 학습해야 하는 많은 수의 파라미터 증가
 - 뒤이 나오는 fine tuning의 경우도 동일
- 학습 데이터의 양이 많지 않은 경우
 - 학습 데이터의 양이 많지 않은 경우!! ⇒ 과적합 문제 발생
 - 학습 데이터의 양 증가시키는 방법
 - ① 정답이 있는 이미지를 추가 준비 ⇒ 쉽지 않음
 - ② Augment existing data (known as data augmentation)



Data augmentation

■ Data augmentation

- 원본 이미지를 약간 변형하여 새로운 이미지 생성, 컴퓨터 관점에서는 다른 이미지로 인식
- 주요 방법
 - Affine transformation
 - Rotation
 - Random Shift: Shift the images to the left, or to the right.
 - Zoom: Zoom in and out slightly of the image.
 - Flipping: 이미지를 뒤집는 것
 - Deep learning based methods
 - Feature space augmentation (autoencoder 이용)
 - GAN-based methods



Data augmentation

- Data augmentation (cont'd)
 - Keras의 경우
 - ImageDataGenerator 사용
 - https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

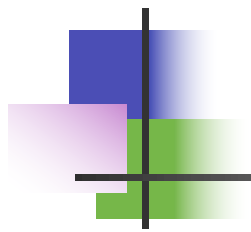
```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,  
                                   rotation_range=20,  
                                   width_shift_range=0.2,  
                                   height_shift_range=0.2,  
                                   zoom_range=0.2,  
                                   horizontal_flip=True,  
                                   vertical_flip=True)
```



Transfer learning

- 기존 모형에 새로운 층 추가
 - 여기서는 Functional 방법을 사용하여 새로운 층 추가

```
input = Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3))  
custom_model = base_model(input)  
custom_model = GlobalAveragePooling2D()(custom_model)  
custom_model = Dense(64, activation='relu')(custom_model)  
predictions = Dense(NUM_CLASSES, activation='softmax')(custom_model)
```

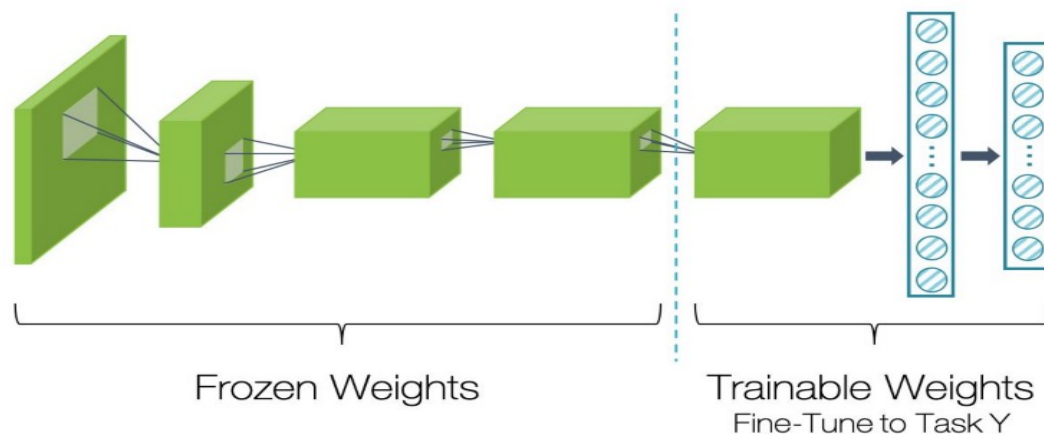


Fine tuning

Fine tuning

- Fine tuning

- Top layers만을 unfreeze하는 것(혹은 제거하는 것)이 아니라 generic layers의 일부도 unfreeze해서 직접 학습을 하는 것
- 보다 많은 layers를 우리가 가지고 있는 데이터를 이용해서 직접 학습을 하기 때문에 더 좋은 성능을 낼 수 있다.



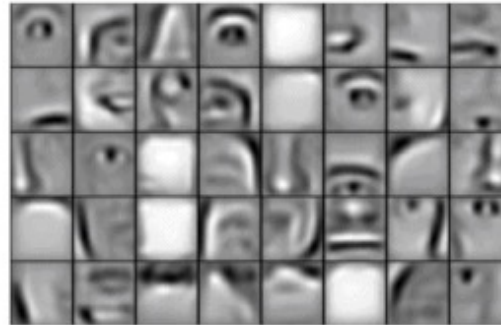
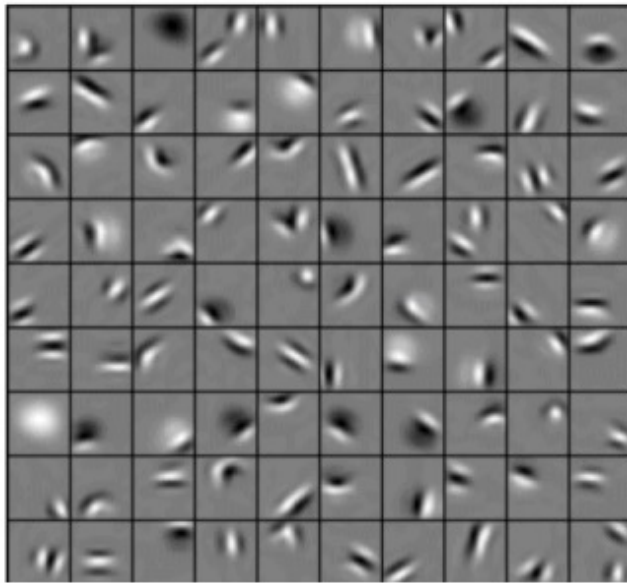


Fine tuning

- 어떠한 층의 파라미터를 새롭게 학습하느냐?
 - 보통 출력층에 가까운 파라미터부터 새롭게 학습
 - 이는 입력층에 가까운 층일수록 이미지의 일반적인 정보를 추출하고, 출력층에 가까운 층일수록 주어진 task를 푸는데 중요한 정보를 추출

Fine tuning

- 사람의 얼굴을 인식하는 문제의 경우



입력층

출력층

위의 그림에서 보이는 것 처럼 출력층에 가까울수록 주어진 문제 (사람 얼굴 인식)에 가까운 정보가 추출된다.



Fine tuning

- 얼마나 많은 layers를 fine tuning 할 것인가?
 - 학습에 사용가능한 데이터의 양과 task의 유사한 정도에 의해 결정
 - 학습 데이터의 양이 많을수록, task의 유사도가 작을수록 더 많은 layer를 학습한다.



Fine tuning

- Python code
 - DL_transfer_learning_fine_tuning.ipynb

```
base_model = MobileNet(include_top=False,  
                        input_shape=(IMG_WIDTH, IMG_HEIGHT, 3))  
for layer in base_model.layers[:-2]:  
    layer.trainable = False
```

- Top layers가 아닌 layers 중에서 마지막 두계층을 새롭게 학습
- 새롭게 학습을 하더라도 사전학습 결과를 초기값으로 사용