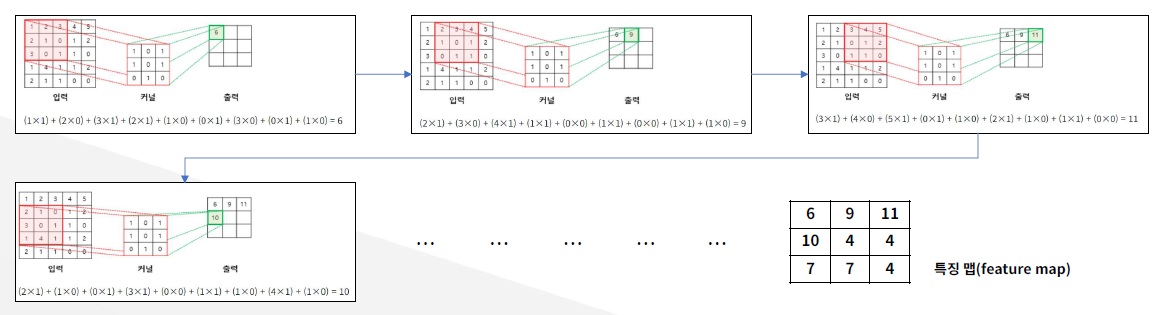
**4.1 CNN Layer의 구조**

* CNN(Convolutional Neural Nets)은 이미지 및 영상 인식을 위한 딥러닝의 기본 모델이다
* MLP의 개선형
  + MLP의 신경망 구조의 경우 완전연결층(Fully-Connected Layer)을 구성하기 떄문에 은닉층을 추가할수록 학습해야 할 파라미터(weights)의 수가 많아지고, 학습 복잡도가 높아진다
  + Convolution 연산을 활용하는 CNN의 경우 MLP와 달리 인접 층의 노드들이 Fully Connected를 이루지 않는다. 따라서 신경망의 복잡도가 낮아지고, 효율적인 학습이 가능하게 된다
* CNN 레이어의 구조
  + convolutional layer : convolution 연산을 수행하여 신경망의 복잡도를 낮추는 층
  + pulling layer : pulling 연산을 수행하여 이미지의 특징을 추출하는 층
  + Fully Connected Layer : MLP와 같은 구조의 최종 출력층

**4.1.1 Convolution Layer**

**4.1.1.1 convolution 연산**



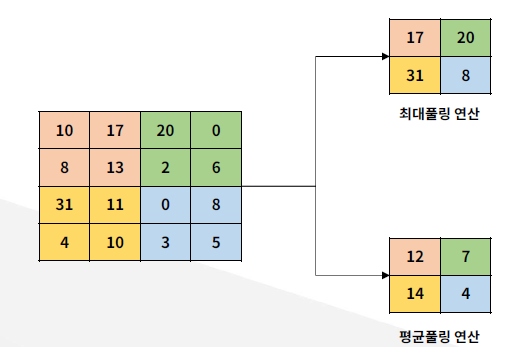
* 입력 데이터는 kernel(혹은 filter)라고 불리는 행렬과 convolution 연산(\*)을 수행한다
* convolution 연산을 통해 이미지 차원의 축소되고(Padding을 하지 않을 경우) 특징 맵(featrue map)이 만들어지게 된다

**4.1.1.2 convolution 연산에서 조정 가능한 파라미터**

* 필터 크기(Kernel Size) : convolution 연산에서 사용되는 필터의 크기를 정한다
* 필터의 크기가 커질수록 더 많은 공간적 특징을 포착할 수 있다
* 너무 큰 필터의 경우 연산량이 증가하고 모델이 복잡해 질 수 있다
* 스트라이드(stride) : 필터가 입력 데이터에서 convolution 연산을 수행하는 간격
* 스트라이드가 커질수록 convolution연산을 수행횟수가 줄어들면서 모델의 크기가 줄어들지만 이미지의 정보 손실이 발생하게 된다
* 패딩(Padding) : 출력 크기 조절을 위해 사용하는 기법. convolution연산을 수행하면서 작아진 특징 맵의 크기를 입력 맵과 동일하게 유지하기 위해 크기를 늘리고 주변을 ‘0’과 같은 무의미한 데이터로 채워 넣는다
* 정보 보존 및 출력 크기 조절 : 패딩을 수행할 경우 이미지의 중요한 특징이 가장자리에서 손실되지 않고 유지될 수 있다
* 스트라이드와 함께 사용하면 모델의 크기를 유지하면서 연산량을 줄일 수 있다
  + 패딩 처리한 데이터는 학습 시 가중치 업데이트가 더 안정적으로 진행된다고 알려져 있다

**4.1.2 Pooling Layer**

**4.1.2.1 Pooling 연산**



* pulling 연산은 input 이미지를 정해진 경계 만큼 격자 형태로 분할한 뒤, 해당 분할된 영역 안에서 가장 큰 값(최대 풀링) 하나만을 선택하거나, 분할 영역 안의 요소들의 평균값(평균 풀링 연산)을 취한다
* 일반적으로는 거의 대부분의 경우 최대 풀링 연산을 많이 사용한다

**4.1.2.2 Pooling 연산의 역할**

* 특징 다운샘플링(Feature Downsampling) : 모델 데이터의 크기를 감소시키면서도 모델의 중요한 특징을 유지한다
* 변이에 대한 강건함(Intensivity) 유지 : 풀링 연산이 진행되는 지역 안에서 가장 두드러지는 특징만 추출함으로서 입력 이미지의 작은 변화나 왜곡에 대해서도 모델이 안정성을 유지하도록 만든다
* 과적합(Overfitting) 방지 : 특징 개수를 감소시키면서 모델의 파라미터를 줄이고, 과적합의 위험을 감소시킨다
* 추상화 수준 증가 : 이미지의 추상화 수준을 높여 더 고차원의 특징을 추출할 수 있게 한다

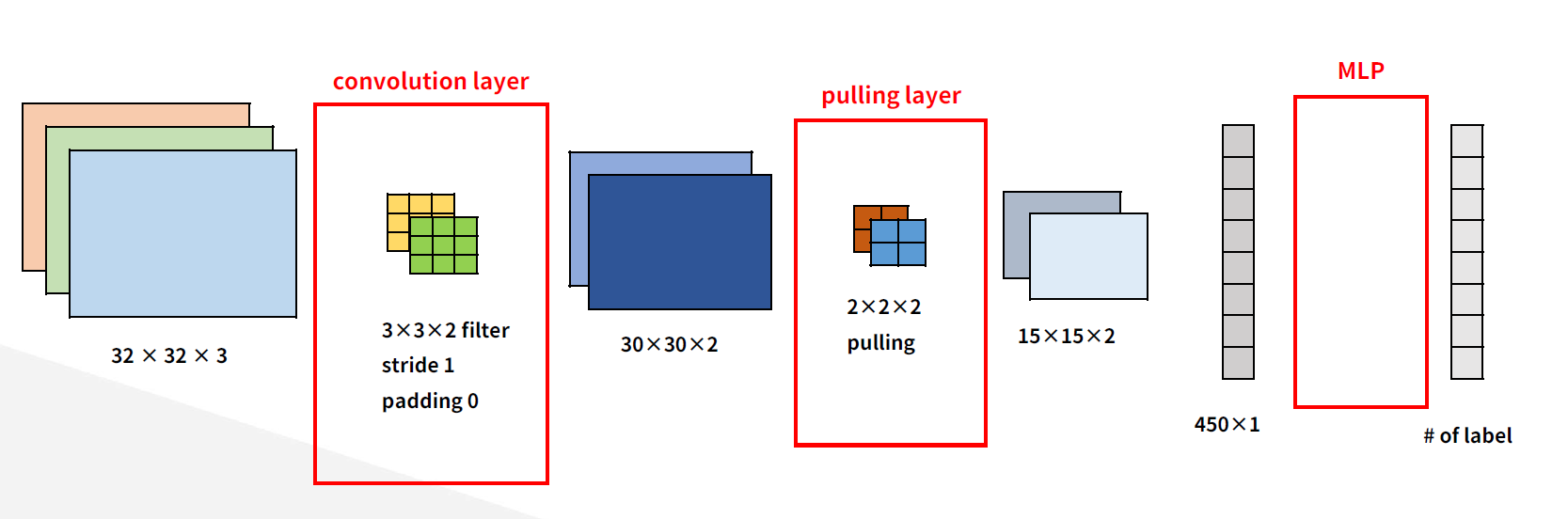
**4.1.3 Fully Connected Layer(Perceptron)**

* Convolution과 Pulling 연산을 반복하여 차원이 축소된 이미지는 flatten 연산을 통해 1차원 벡터로 변환되고, 이 벡터값이 Fully Connected layer를 통과하면서 출력을 발생시킨다. 이 부분의 작동 방식은 앞에서 살펴본 MLP와 동일하다

**4.2 다중 채널 이미지(RGB, RGBA, CMYK...)**

* 4.1에서 설명한 CNN 구조는 입력 데이터가 한 층의 격자 그리드로 표현될 수 있는 이미지에 적용된다. 예컨대 흑백 이미지와 같은 경우가 그렇다
* 색상이나 명암, 투명도 등을 가진 더 복잡한 형태의 이미지일 경우 입력 레이어가 여러 장으로 들어오게 된다.
* 예컨대 A라는 RGB 이미지의 경우, R(적), G(녹), B(청)으로 구분될 수 있는 세 개의 layer로 구분될 수 있다. 32×32의 이미지일 경우 전체 input은 32×32×3이 된다. ※ MLP에서는 이러한 이미지가 flatten 연산 후 concat(그대로 이어 붙임)되어 32×32×3의 1차원 벡터로 신경망에 입력된다
* **채널별 특징 학습** : 다중 채널일 경우 각 색상 채널에서 특징이 학습된다. 각 채널 간의 관계와 채널 간의 패턴을 모두 학습해야 한다
* 데이터 전처리 시 모든 입력 이미지가 동일한 크기의 채널 구성(동일한 이미지 형식)으로 전처리 되어야 한다
* **참고 : 다중 이미지 채널의 형태들**
  + RGB : 적, 녹, 청의 3개의 채널로 구성됨
  + RGBA : RGB에 알파 채널(투명도)이 추가된 4개의 채널로 구성됨
  + CMYK : 인쇄 산업에서 주로 사용되는 색상 모델로 Cyan(시안), Magenta(마젠타), Yellow(노랑), Key(검정)의 4개의 채널
  + HSV : 색상(Hue), 채도(Saturation), 명도(Value)의 3개의 채널을 사용함
  + 적외선 : RGB 이외의 채널을 사용함 -> grayscale
  + 멀티스택트럼 이미지 : 4개 이상의 채널을 가짐

**4.3 다중 채널 이미지가 도입된 CNN 모델에서 Convoultion-Pulling 연산과 date dimenstion의 변화**



**import** tensorflow. **as** tf

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, Flatten, Conv2D, MaxPooling2D

model = Sequential([

Conv2D(2, kernel\_size=(3, 3), input\_shape=(32, 32, 3), activation='relu'),

MaxPooling2D(pool\_size=2),

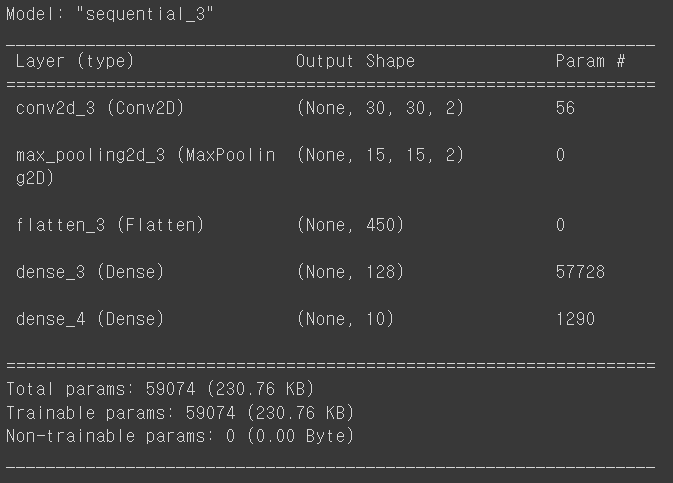
Flatten(),

Dense(128, activation='softmax')

Dense(10, activation='softmax')

])

model.summary()



1. Input data는 3개의 channel을 가진 32×32×3의 텐서로 input layer에 입력된다
2. 코드에는 input layer가 명시적으로 구현되어 있지 않다. input layer에서는 아무런 연산도 수행하지 않기 때문이다
3. conv2d에서 convolution 연산을 수행한다. 2개의 필터가 3×3 kernel을 (32×32×3)의 input에 대하여 수행한다. activation function은 'relu'로 CNN에서 가장 많이 사용되는 함수이다
4. Conv2D를 거친 output은 30×30×2 크기의 특징맵으로 축소된다.
5. pooling size 2로 max pooling 연산을 거치면, 15×15×2로 한번 더 차원이 축소된다
6. 5의 결과가 퍼셉트론 층을 통과한다. 그림에는 표시되어 있지 않지만, flattne된 입력 벡터가 10개의 레이블 분류를 수행하기 위해 hidden layer가 추가되어야 한다(여기서는 hidden layer가 128개의 input node를 갖는다고 가정함)

**4.4 예제 코드 리뷰**

**4.4.1 Cats or Dogs**

**import** tensorflow **as** tf

**import** tensorflow\_datasets **as** tfds

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense, Flatten, Dropout, Conv2D, MaxPooling2D

**from** tensorflow.keras.optimizers **import** Adam

# CNN 모델 구축

model = Sequential([

    Conv2D(2, kernel\_size=(3, 3), input\_shape=(64, 64, 3), activation='relu'),

    MaxPooling2D(pool\_size=2),

    Flatten(),

    Dense(128, activation='relu'),

    Dense(1, activation='sigmoid')

    ])

# 모델 요약

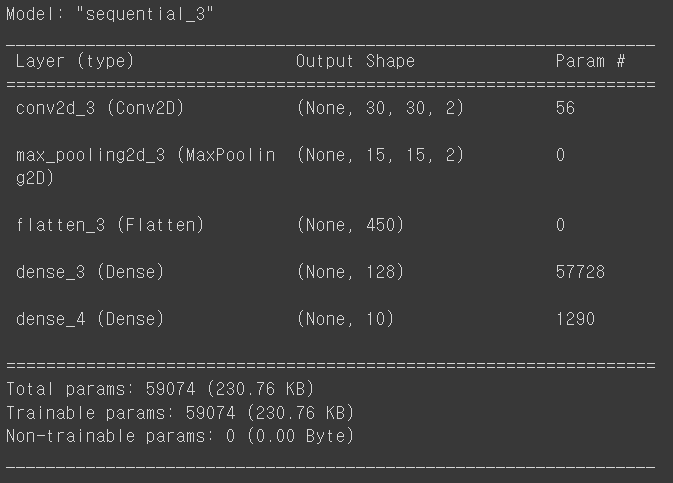
model.summary()

# 모델 컴파일

model.compile(optimizer=Adam(), loss='binary\_crossentropy', metrics=['accuracy'])

# 모델 훈련

model.fit(train\_ds, validation\_data=validation\_ds, epochs=3)



* hidden layer를 두 겹으로 깔았던 MLP보다 분명하게 성능이 향상되었다

**4.4.2 Cifar-10**

**import** tensorflow **as** tf

**import** tensorflow\_datasets **as** tfds

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense, Flatten, Dropout, Conv2D, MaxPooling2D

**from** tensorflow.keras.optimizers **import** Adam

# CNN 모델 구축

model = Sequential([

    Conv2D(2, kernel\_size=(3, 3), input\_shape=(32, 32, 3), activation='relu'),

    MaxPooling2D(pool\_size=2),

    Flatten(),

    Dense(128, activation='relu'),

    Dense(10, activation='softmax')

    ])

# 모델 요약

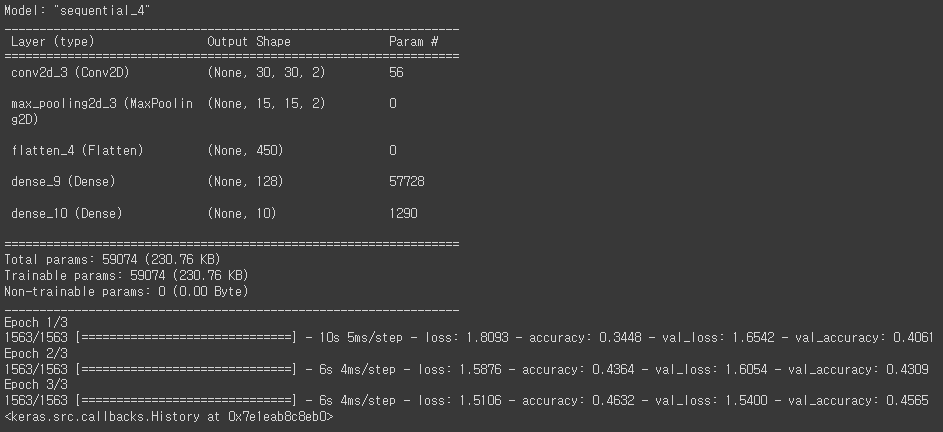
model.summary()

# 모델 컴파일

model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])

# 모델 훈련

model.fit(x\_train, y\_train, validation\_data=(x\_test, y\_test), epochs=3)



**4.5 이후 학습**

**4.5.1 고급 CNN 모델**

**AlexNet**: 2012년 ImageNet 챌린지에서 우승한 모델. 딥러닝 기반 이미지 인식의 시초. CNN의 기본적인 구조를 따르지만, 여러 개의 GPU를 사용하여 더 깊고 넓은 네트워크를 훈련시킨 최초의 모델 중 하나이다.

* [[NIPS-2012-imagenet-classification-with-deep-convolutional-neural-networks-Paper.pdf]]
* https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

**VGGNet**: 깊은 층의 구조로 유명하며, 이미지 인식에서 좋은 성능을 보이는 모델로 일관된 컨볼루션 필터 크기(3x3)를 사용하여 네트워크의 깊이를 극대화한 모델. 단순화된 구조이지만 깊이를 통한 성능 향상에 중점을 둔다.

* [[Very Deep Convolutional Networks for Large-Scale Image Recognition.pdf]] (Simonyan and Zisserman)
* https://arxiv.org/pdf/1409.1556.pdf

**ResNet**: 깊은 신경망에서의 소실 문제를 해결한 모델로, 많은 연구와 프로젝트에서 사용된다. '잔차 연결(Residual Connection)'이라는 개념을 도입하여 깊은 네트워크에서 발생하는 기울기 소실 문제를 해결한다. 잔차 연결이란 간단히 말해 이전 레이어의 출력을 몇 레이어를 건너뛰고 다음 레이어의 입력으로 직접 추가하는 방식을 말한다.

* [[He\_Deep\_Residual\_Learning\_CVPR\_2016\_paper.pdf]] (He et al.)
* https://openaccess.thecvf.com/content\_cvpr\_2016/html/He\_Deep\_Residual\_Learning\_CVPR\_2016\_paper.html

**Inception (GoogleNet)**: 다양한 크기의 컨볼루션 필터를 병렬적으로 사용하여, 이미지의 다양한 스케일에 대응할 수 있게 한다. 이는 'Inception 모듈'이라는 독특한 구조를 통해 구현된다.

* [[Szegedy\_Going\_Deeper\_With\_2015\_CVPR\_paper.pdf]] (Szegedy et al.)
* https://www.cv-foundation.org/openaccess/content\_cvpr\_2015/papers/Szegedy\_Going\_Deeper\_With\_2015\_CVPR\_paper.pdf

**EfficientNet**: 네트워크의 깊이, 너비, 그리고 해상도를 균형 있게 확장하는 방식으로 설계되었다. CNN을 더 효율적으로 확장하기 위한 새로운 방법론을 제시한다.

* [[EfficientNet\_ Rethinking Model Scaling for Convolutional Neural Networks.pdf]] (Tan and Le)
* https://proceedings.mlr.press/v97/tan19a/tan19a.pdf

**4.5.2 CNN 외의 방법을 사용한 이미지 인식**

**Vision Transformers (ViT)**: 이미지 인식 분야에 응용된 Transformer 모델. ViT는 이미지를 여러 개의 패치로 분할하고, 이 패치들을 Transformer 모델에 입력하여 이미지 인식 작업을 수행한다. 이 접근 방식은 특히 대규모 데이터셋에서 CNN 기반 모델들과 비교할 때 경쟁력 있는 성능이 나오고 있다.

* [[An Image is Worth 16x16 Words- Transformers for Image Recognition at Scale.pdf]] Alexey Dosovitskiy et al.
* https://arxiv.org/pdf/2010.11929.pdf

**Hybrid Models**: CNN과 Transformer의 장점을 결합한 모델. CNN을 사용해 이미지의 특징을 추출하고, 이를 Transformer 모델에 입력하여 더 정교한 이미지 인식을 수행하는 방식이다.

* [[applsci-13-05521-v2 1.pdf]]

**Capsule Networks**: CNN과는 다른 접근 방식을 취하는 Capsule Networks. 객체의 공간적 계층 구조를 더 잘 이해하기 위해 설계되었으며, 여러 방향에서의 객체 인식과 관점 변화에 더 강건할 수 있다.

* [[NIPS-2017-dynamic-routing-between-capsules-Paper.pdf]] Sara Sabour, Nicholas Frosst, Geoffrey E. Hinton et al.

**4.5.3 YOLO(You Only Look Once)**

실시간 객체 탐지 분야에서 매우 인기 있는 알고리즘으로 객체 탐지를 위한 신경망이다. YOLO 이미지 전체를 한 번에 보고 객체의 위치와 분류를 동시에 수행한다. 이 접근 방식은 전통적인 객체 탐지 시스템보다 훨씬 빠르며, 실시간 처리에 적합한다.

**YOLOv1**: YOLO 시리즈의 첫 번째 버전으로, 전체 이미지를 한 번에 처리하여 객체 탐지를 수행하는 방식을 도입했다

* [[You Only Look Once Unified, Real-Time Object Detection.pdf]] - Joseph Redmon et al.
* https://www.cv-foundation.org/openaccess/content\_cvpr\_2016/papers/Redmon\_You\_Only\_Look\_CVPR\_2016\_paper.pdf

**YOLOv2 및 YOLOv3**: 이 버전들은 정확도와 속도를 개선하기 위한 기술들이 추가되었다

* [[YOLO9000\_Better\_Faster\_CVPR\_2017\_paper.pdf]] - Joseph Redmon and Ali Farhadi
* https://openaccess.thecvf.com/content\_cvpr\_2017/papers/Redmon\_YOLO9000\_Better\_Faster\_CVPR\_2017\_paper.pdf
* [[YOLOv3\_ An Incremental Improvement.pdf]]
* https://arxiv.org/pdf/1804.02767.pdf

**YOLOv4**: 더 높은 정확도와 빠른 속도를 위해 최적화되었으며, 중소 규모의 GPU에서도 높은 성능을 달성할 수 있도록 설계되었다 - [[YOLOv4\_ Optimal Speed and Accuracy of Object Detection.pdf]] - Alexey Bochkovskiy et al. - https://arxiv.org/pdf/2004.10934.pdf - 라이센스 만료

**YOLOv5**: 비공식 버전이지만, 사용이 매우 간편하고 성능도 우수한 것으로 알려져 있다. 여러 커뮤니티에서 널리 사용되고 있다.

* https://github.com/ultralytics/yolov5
* 라이센스 문제 검토 필요

**YOLOX**: "YOLOX: Exceeding YOLO Series in 2021" - YOLOX 팀. YOLO의 확장 버전으로, 최신 기술과 개선 사항을 포함하고 있다.

* [[YOLOX\_ Exceeding YOLO Series in 2021.pdf]]
* https://arxiv.org/pdf/2107.08430.pdf
* 라이센스 문제 검토 필요

**4.5.4 fine-tuning**

미리 학습된(pre-trained) 모델에 자신의 데이터셋을 입력하여 추가 학습을 하는 과정을 일반적으로 "미세 조정(fine-tuning)"이라고 한다. 이 과정에서 일부 레이어는 고정(freeze)하고, 일부는 새로운 데이터에 맞게 학습한다. 여기서 레이어를 고정한다는 것은 해당 레이어의 가중치가 학습 과정에서 업데이트되지 않도록 하는 것이다.

예를 들어, YOLOv3 객체 탐지 모델을 사용하여 특정 종류의 객체(예: 특정 동물 또는 제품)를 탐지하고자 하는 경우:

1. **미리 학습된 모델 불러오기**: 먼저, YOLOv3의 미리 학습된 모델을 불러온다. 이 모델은 이미 일반적인 객체에 대해 학습되어 있는 상태이다.
2. **레이어 고정**: 모델의 초기 몇 개의 레이어(예: 합성곱 레이어)를 고정한다. 이렇게 하면 이 레이어들의 가중치는 학습 과정에서 변경되지 않는다. 이 초기 레이어들은 일반적인 이미지의 특징을 추출하는 데 유용하다.
3. **마지막 레이어 조정**: 모델의 마지막 몇 개의 레이어(예: 완전 연결 레이어)를 조정하거나 새로운 레이어를 추가한다. 이 레이어들은 새로운 데이터셋에 특화된 특징을 학습하는 데 사용된다.
4. **새로운 데이터셋으로 학습**: 자신의 데이터셋(새로운 객체 이미지와 레이블)을 사용하여 모델을 학습한다. 이 때, 고정된 레이어는 그대로 유지되고, 나머지 레이어들의 가중치만 업데이트된다.
5. **모델 평가 및 미세 조정**: 학습된 모델을 평가하고, 필요에 따라 하이퍼파라미터를 조정하거나 추가 학습을 진행한다.

실제로 fine-tuning을 하는 구체적인 구현 방법은 공개된 모델에 나와 있기도 하고, 세부적인 방법들이 조금씩 다르다. 사전 학습된 공개된 모델들을 분석하고, 적절한 모델을 선택한 후 세부적인 튜닝 방법을 찾아봐야 한다.

**4.5.5 fine-tuning의 일반적인 지침**

1. **일반적인 지침**: 일반적으로, 모델의 초기 레이어는 이미지의 기본적인 특징(예: 가장자리, 색상, 질감 등)을 추출한다고 알려져 있다. 이러한 기본적인 특징은 다양한 종류의 이미지에서 비슷하기 때문에, 초기 레이어는 고정하는 것이 일반적이다. 반면, 모델의 후반부 레이어는 더 복잡하고 구체적인 특징을 학습하며, 특정 작업이나 데이터셋에 더 특화되어 있다. 따라서, 이 후반부 레이어들은 더 자주 조정하게 된다.
2. **데이터셋의 유사성**: 학습할 새로운 데이터셋이 미리 학습된 모델이 사용한 원본 데이터셋과 매우 유사한 경우, 더 많은 레이어를 고정할 수 있다. 반면, 데이터셋이 매우 다를 경우, 더 많은 레이어를 조정해야 할 수 있다.
3. **모델의 복잡도와 데이터의 크기**: 큰 데이터셋을 사용하거나 매우 복잡한 모델을 다루는 경우, 더 많은 레이어를 조정할 필요가 있을 수 있다. 반면, 작은 데이터셋이나 간단한 모델의 경우 이러한 조정 과정이 줄어든다.
4. **실험과 평가**: 사실 어떤 레이어를 고정하거나 조정해야 하는지 결정하는 가장 좋은 방법은 실험을 통한 것으로 다양한 구성을 시도하고, 각 설정에 대한 모델의 성능을 계속 평가해보는 것이 제일 좋다.