# CNN모델 이미지분류 - VGG16 활용 추천





### cnn\_cat\_dog.ipynb

내가 책을 보면서 따라한 고양이/개 분류 코드야.

참고로 난 AI 에 대해서 아무것도 몰라.

뭐 VGG16? 이걸로 하면 더 좋아?

네가 작성한 고양이/개 분류 코드를 잘 읽어봤어! 👍



### 완전 잘 따라했어!

아주 전형적인 CNN(Convolutional Neural Network, 합성곱 신경망) 기본 구조로 잘 만들었어.

지금 코드 구조는:

- 📶 데이터 준비
- 👉 cats and dogs filtered 데이터셋 다운로드
- 👉 학습용 / 검증용 데이터 분리
- 🔃 이미지 변형(증강)
- 👉 ImageDataGenerator 로 이미지 회전, 이동, 확대 등으로 다양하게 만들어서 학습에 도움
- ③ CNN 모델 만들기
- Conv2D + MaxPooling2D → Flatten → Dense (일반적인 CNN 구조)
- 학습시키기 (model.fit)

#### 그런데 너가 물어본 것 중에

VGG16 쓰면 더 좋아?

☑ 답 : 응, 좋아질 가능성이 커!

## 왜 그런지 아주 쉽게 설명해볼게 👇

## 🤗 지금 만든 모델은?

네가 만든 모델은 "내가 직접 구조를 설계한 작은 CNN" 이야.

즉, 너가 직접 Conv2D 몇 개 넣고, Dense 몇 개 넣어서 **작은 뇌**를 만든 거야.

하지만  $\rightarrow$  이 모델은 **아주 깊지는 않아**  $\rightarrow$  그러니깐 고양이/개 정도는 구분하지만 **복잡한 이미지**(예: 고양이 얼굴 여러 각 도, 다양한 품종, 강아지 장난감 등)에서는 약간 부족할 수 있어.

## 🎎 VGG16 은 뭐냐면?



#### VGG16 은

- $\rightarrow$  아주 **깊고 복잡한 CNN 구조**
- → 전문가(옥스퍼드 대학 연구팀)가 만든 모델
- → 그리고 이미 100만 장의 이미지로 미리 학습(pretrained) 되어 있음!
- → 그래서 **초보자가 쓰기 매우 좋음** → 그냥 가져다 쓰기만 해도 성능이 빵빵해

### 🚛 비유

네가 만든 모델	VGG16
초등학생이 직접 그린 지도	전문가가 만든 네이버 지도
작고 가벼움	크고 정확함
빠르게 학습	학습 속도 느릴 수 있지만 성능 높음
작은 데이터셋에 적합	큰 데이터셋, 복잡한 이미지에 강력

### ? 그럼 나는 왜 VGG16 을 쓰지?

- 👉 transfer learning(전이학습) 이라는 기법을 써서

그러면 →

- 👉 학습 속도도 빠르고
- 👉 성능은 아주 높아짐

## 👽 정리

- ☑ 지금 코드: 잘 했어! CNN 기본 구조
- $\bigvee$  VGG16 사용: 더 깊은 학습 + 이미 pretrained → 더 좋은 성능

## 🔜 만약 원하면?

내가 너 코드에 VGG16 으로 바꾸는 버전도 만들어서 보여줄까?

그럼 비교도 가능해! (지금 코드 vs VGG16 코드)

#### 한 문장 요약:

네가 만든 것도 충분히 좋지만 → VGG16 쓰면 더 높은 정확도를 쉽게 얻을 수 있어.

해볼래? →

👉 네가 만든 코드에 VGG16 적용한 버전 만들어서 보여줄게! 🚀



니가 VGG16 버전으로 만들면, VGG16 모델에 새로운 데이터를 학습 시키는 코드로 만들어서 VGG16을 업그레이

이 질문 정말 좋아! 🤲

내가 엄청 쉽게 설명해줄게!

### 🬳 VGG16 을 "나무"로 비유할게

- VGG16 = 이미 "100만 장 이미지"로 키운 엄청 큰 나무야 (가지, 뿌리, 잎사귀 다 있음)
- 근데 → 우리 고양이/개 데이터는 **새로운 열매(분류기)를 붙이는 것**에 가까워

### VGG16 에서 "어떻게 학습시키냐면" → 2가지 방법이 있어

## 🔟 Feature Extractor (특징 추출기) 방식 — 🌟 초보자가 제일 많이 씀

- VGG16 의 기존 뇌(가지, 뿌리)는 그대로 놔둬
- 마지막에 → **내가 만든 작은 층(분류기 Dense 층)** 만 새로 만들어서 **그 부분만 학습**
- VGG16 자체는 안 바뀜 → "업그레이드"는 아니고 "위에 덧붙이기"임

#### 👉 빠르고 효과 좋음

👉 초보자 추천

## 🗾 Fine Tuning (미세 조정) 방식 — 🌟 중급 이상

- VGG16 전체(혹은 윗부분 일부)를 **살짝 학습 가능하게 열어줌**
- 그러면 → VGG16 자체도 조금씩 새 데이터에 맞게 업그레이드 됨
- 대신 느리고 조금 어렵고 오버피팅 위험도 있음
- 👉 더 높은 성능
- 👉 약간 더 복잡한 코드

## 🔔 한 줄 요약

방법	설명	추천 대상
Feature Extractor	VGG16 은 고정 + 위에 Dense 층만 학습	초보자에게 추천
Fine Tuning	VGG16 일부도 학습 가능하게 해서 새로 조정	중급 이상에게 추천

## 🞁 정답:

#### 너가 지금 한 "고양이/개 분류" 같은 경우엔 →

- - 빠르고



- 성능 잘 나옴
- 코드도 쉽고 이해 가능

#### ■ 그러면 "내가 만들어줄 VGG16 버전 코드"는?

- 그 부분만 학습하는 코드로 만들어줄게!

### 🚀 그럼 최종 정리

너 질문: "VGG16 업그레이드 하는 방식으로 만들꺼야?"

### ✔ 답변:

- ightarrow 처음엔 업그레이드 안 하고 (Feature Extractor) 방식으로 하는게 좋아
- → **필요하면 Fine Tuning 으로 업그레이드도 가능해** (나중에 배워도 됨!)

### 👉 다음 단계

너가 **"응, 만들어줘!"** 하면 내가 지금 네 코드 구조에 맞춰서

해볼까? 만들까? 🚀 😊



Feature Extractor 는 어떤상황에서 쓰는게 좋고, Fine Tuning 은 어떤 상황에서 쓰는게 좋아? 쉽게 설명해줘

좋아! 아주 쉽게 설명해줄게 🚀

#### 비유부터 해볼게:

"사진 전문가 + 초보자 조수" 팀이라고 생각해봐!

## 🚺 Feature Extractor (전문가 뇌 고정 + 조수만 새로 학습)

#### 상황:

- 사진에서 무엇이 중요한 특징인지 뽑는 능력이 엄청 좋아!

그런데 지금은 "고양이 vs 개 구별하기" 같은 간단한 새 작업을 하고 싶어.

#### 그래서:

- ☑ 전문가(기존 VGG16)는 그대로 두고
- 🔽 **맨 마지막 단계(조수만) 새로 학습해서** 고양이/개만 구분하게 해!



#### 특징:

- ✓ 데이터가 적을 때 (ex. 고양이/개 2000장 정도)
- ☑ 빠르게 결과 내고 싶을 때
- ☑ 처음 배우는 사람일 때
- ☑ 오버피팅(과적합) 방지됨

#### 정리:

- 👉 새로운 작업이 기존 지식과 비슷하고

## 🔃 Fine Tuning (전문가 뇌도 일부 새로 학습)

#### 상황:

(ex. 의료용 X-ray 사진  $\rightarrow$  고양이/개랑 완전 다름)

#### 혹은

- 👉 전문가 뇌가 조금 오래돼서
- 새로운 데이터에 좀 더 잘 맞추고 싶어.

#### 그래서:

- ✓ VGG16 윗부분(뇌 일부)까지 조금 열어서 새로 학습
- ☑ 새 데이터에 "업그레이드" 시킴

#### 특징:

- ☑ 데이터가 많을 때 (ex. 만 장 이상)
- ☑ 기존 작업과 새 작업이 좀 많이 다를 때
- ☑ 최고 성능을 원할 때
- ☑ 시간이 오래 걸릴 수 있음 (느려짐)

#### 정리:

- 👉 기존 지식만으로 부족할 때
- 👉 데이터가 충분할 때

## 🚛 표로 정리!

방법	언제 쓰면 좋은가	특징
Feature Extractor	데이터 적음, 기존 작업과 유사, 빠르게 하고 싶음	빠르고 쉽고 오버피팅 방지
Fine Tuning	데이터 많음, 기존 작업과 다름, 최고 성능 원함	느리지만 성능 더 높일 수 있음

## 🎁 예시

작업 예시	추천 방법
고양이/개 분류 (네가 지금 하는 것)	Feature Extractor
의료 영상 (X-ray, CT 스캔 분류)	Fine Tuning

작업 예시	추천 방법
자율주행 자동차용 도로 객체 탐지	Fine Tuning
꽃 종류 분류 (5~10종)	Feature Extractor

### 🚀 한 문장 요약

- Feature Extractor → 빠르고 쉬운 기본 사용법 (처음엔 이거부터 배우기)
- 👉 Fine Tuning → 데이터 많고 성능 더 높이고 싶을 때 사용

혹시 여기까지 이해됐어?

예시 더 원하면, 네 상황(고양이/개) 기준으로 구체적인 예도 설명해줄게! ★ 그리고 원하면 바로 코드도 만들어줄게! ★ ♡



개,말,고양이,호랑이 이렇게 분류 종류가 늘어나면 Feature Extractor, Fine Tuning 중에 뭐로 해?

너무 좋은 질문이야!! 바로 예시로 쉽게 설명해줄게 🚀

## $\red{\S}$ $\red{\S}$ $\red{\S}$ ightarrow 클래스(종류)가 2개 ightarrow 4개로 늘어났어!

- **/** 고양이/개 → 고양이/개/말/호랑이
- 👉 총 4개 클래스 분류 문제

### 그럼 뭘 쓰면 좋을까? Feature Extractor vs Fine Tuning

## ★ [i] Feature Extractor 로 충분한 경우

### 조건:

- 4개 클래스가 다 일반적인 사진 (말, 호랑이, 개, 고양이 사진들 전부 자연 사진)
- VGG16 은 이미 일반 사진 엄청 많이 학습했음 (개, 고양이, 말, 호랑이 다 비슷한 걸 본 적 있어!)
- ← 마지막 분류기(Dense 층)만 2개 → 4개로 바꿔서 학습하면 됨

#### 추천:

- ✓ 데이터가 적음 (ex. 클래스당 몇백~몇천 장 정도)
- ☑ 빠르게 학습하고 싶음

#### 결론:

- 👉 성능 잘 나오면 끝!

## 🚀 🔃 Fine Tuning 이 필요한 경우

#### 조건:

- 만약 사진이 **VGG16 학습한 것과 많이 다른 경우** (ex. 만화 스타일, 의료 영상, 드론 항공사진 등)
- 혹은 사진이 엄청 다양해서 → VGG16 만으로 부족한 경우
- 데이터가 충분히 많음 (ex. 클래스당 만 장 이상)

#### 추천:

- ☑ 조금 더 고성능 원할 때
- ☑ 클래스가 많아지면서 "고양이와 호랑이" 같이 **구분 어려운 것들이 있을 때** → Fine Tuning 윗부분까지 조금 조정하면 도움이 됨

#### 결론:

👉 Feature Extractor → 먼저 해보고 부족하면 Fine Tuning 으로 업그레이드

## 值 최종 요약 (지금 상황)

상황	추천 방법
고양이/개/말/호랑이 → 일반 사진 (자연, 인터넷 사진)	Feature Extractor
만화 스타일, 드론 사진, 의료 영상처럼 특이한 경우	Fine Tuning 고려
데이터 적음 (천 장 이하)	Feature Extractor
데이터 많음 (만 장 이상), 최고 성능 원함	Fine Tuning

## 할 초보자용 팁

#### 무조건 순서:

- **I** Feature Extractor 먼저 시도 → 성능 확인
- 필요 시 Fine Tuning 진행

### 🚀 예시 적용

너가 **고양이/개/말/호랑이** 일반 사진 모아서 분류하고 싶다 →

#### 혹시 원하면:

"지금 네 코드 구조에 맞춰서 4개 클래스 분류용 VGG16 Feature Extractor 예시 코드"도 바로 만들어줄까? 그럼 너가 코드 돌려보면서 → 2개 클래스 → 4개 클래스 바뀌었을 때 차이도 쉽게 볼 수 있어 🚀 ☆

원해? 만들까? 🚀 😊



VGG16 Feature Extractor 방식으로 가면, conv2d -> maxpooling 이거 몇개로 할지, 매개변수 값은 뭐로 할지를 건너 뛴다는거야?

## 네가 지금 한 CNN 구조 복습 👇

```
mathematica

Conv2D → MaxPooling → Conv2D → MaxPooling → Flatten → Dense
```

#### 여기서

- 👉 Conv2D 몇 개 넣을지
- 👉 필터 몇 개 쓸지
- 👉 커널 사이즈 몇으로 할지
- 👉 MaxPooling 몇 번 쓸지
- → 전부 **네가 직접 설계**해야 했어!

### 근데 VGG16 Feature Extractor 쓰면?

- 👉 구조는 이미 **고정**되어 있음 👇

```
Conv2D(64) → Conv2D(64) → MaxPooling

Conv2D(128) → Conv2D(128) → MaxPooling

Conv2D(256) → Conv2D(256) → Conv2D(256) → MaxPooling

Conv2D(512) → Conv2D(512) → Conv2D(512) → MaxPooling

Conv2D(512) → Conv2D(512) → Conv2D(512) → MaxPooling
```

**슬 총 13개의 Conv2D** + 5개의 MaxPooling → VGG16 구조로 이미 완성!

### 그럼 Feature Extractor 할 때는?

- ☑ Conv2D/MaxPooling 을 몇 개 쓸지 → 네가 고민 안 해도 됨!
- ☑ 그냥 VGG16 구조 그대로 가져와서 쓰면 됨
- ✓ 네가 할 일은 → 맨 마지막 Dense 층(분류기)만 붙이기

### 비유

네가 직접 CNN 설계	VGG16 Feature Extractor 사용
블록 직접 쌓아서 건물 만들기	이미 완성된 빌딩에 간판만 새로 달기
Conv2D/MaxPooling 설계 고민	설계 다 되어 있음 → 그냥 가져다 쓰기
초보자가 실수할 수 있음	안전하게 성능 잘 나옴

## 다시 질문으로 돌아가서:

### VGG16 Feature Extractor 방식으로 가면, conv2d -> maxpooling 이거 몇개로 할지, 매개변수 값은 뭐로 할지를 건너 뛴다는거야?

- **☑** 정답 → **응! 건너뛴다는 거야!**
- ☑ VGG16 은 이미 **전문가가 Conv2D/MaxPooling 쌓아둔 구조**
- ☑ 너는 그 구조를 그냥 "가져다 쓰기만" 하면 돼 → 따로 설계 안 해도 됨

### 너가 할 일

- ▼ GlobalAveragePooling 또는 Flatten

### 한 문장 요약 🚀

#### 혹시 원하면:

지금 네 코드랑 비교해서 → VGG16 Feature Extractor 적용 예시 코드 아주 쉽게 만들어줄게! 너가 차이도 딱 눈으로 보고 이해할 수 있게! ※ ○

원할까? 🖐

(고양이/개 버전으로, 4종류 버전으로 둘 다 가능!) 🚀



VGG16 파인튜닝으로 새로운 종류 데이터 2가지 카타고리를 각각 1000장 정도 google colab에서 학습 시키면 얼마나 걸려?

너무 좋은 질문이야! 🚀

VGG16 Fine Tuning + Colab + 새 데이터 2종류(각 1000장) → 학습 시간

내가 아주 쉽게 설명해줄게!

## 🖈 상황 정리

- ✓ 모델: VGG16
- ☑ 방법: Fine Tuning (VGG16 윗부분 일부 "열어서" 학습)
- ☑ 데이터: 총 2000장 (클래스당 1000장)
- ▼ 환경: Google Colab (무료 티어, GPU 사용)
- ✓ 이미지 크기: 보통 150x150 or 224x224
- ▼ Epoch 수: 보통 10~20 epoch 시도

## 🔀 학습 시간 계산 요소



- 🔢 데이터 수
- $\rightarrow$  2000장  $\rightarrow$  적당히 작은 데이터셋 (크게 오래 걸리진 않음)
- Batch size
- → 보통 32 또는 64
- 🔢 Fine Tuning 정도
- → VGG16 전체를 다 학습하면 느림
- → **보통 윗부분 2~3 block 정도만 열고 학습** → 그러면 빠름
- Colab GPU 속도
- → 무료 Colab GPU (T4, P100 등) 기준

### 🝈 대략적인 시간 예측

설정	예상 시간 (1 epoch 기준)
Feature Extractor	약 10~20초
Fine Tuning (윗부분만 열기, 2~3 block)	약 20~40초
Fine Tuning (전체 열기, full fine tuning)	약 1~2분

### 🚀 최종 예시

Fine Tuning (윗부분만 열기) 로 20 epoch 학습 시:

- 1 epoch ≈ 30本

## 🞁 한 문장 요약

- ightharpoonup VGG16 Fine Tuning 윗부분만 열고 학습 ightharpoonup Colab 에서 약 10분 내외면 충분히 학습 가능!

## ★ 참고 팁

- ✓ 처음엔 Feature Extractor 먼저 해보고 → 성능 확인 후
- ☑ 필요하면 Fine Tuning 일부 열어서 성능 업그레이드

이렇게 단계별로 하면 시간 아끼면서 성능 올릴 수 있어!

혹시 원하면:

실제로 Colab 에서 쓸 수 있는 VGG16 Fine Tuning 예제 코드 (2000장용)도 만들어서 보여줄까? ♬ (너가 바로 돌려볼 수 있게 해줄게!) 원하면 "응!" 만 해 ♣ ♬ ☺️





이 코드를 VGG16 Feature Extractor 를 사용해서 cat, dog, horse 3가지의 카테고리를 분류하는 코드로 바꿔줄 수 있어?

좋아, 네 코드 구조 확인했어!

지금 구조는 "cats\_and\_dogs\_filtered" 데이터셋을 이용한 2가지 클래스 분류용 CNN 코드인데 → **' 내가 VGG16 Feature Extractor + 3개 클래스(cat, dog, horse)** 버전으로 바꿔주는 예시 코드로 바꿔서 보여줄 게! **♂** 



#### 데이터 폴더 구조는 이런 식으로 준비하면 돼:

```
markdown

dataset/
    train/
        cats/
        dogs/
        horses/
validation/
        cats/
        dogs/
        horses/
```

※ horse 사진은 네가 직접 구해서 폴더에 넣으면 됨 (구글 이미지 검색 후 넣으면 됨)

## VGG16 Feature Extractor + 3 클래스 분류 코드 예시 ✗

```
python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
import os
import numpy as np
import tensorflow as tf
# seed 고정
seed = 15
np.random.seed(seed)
tf.random.set_seed(seed)
# 설정값
BATCH SIZE = 32
EPOCHS = 20
IMG_HEIGHT = 150
IMG\ WIDTH = 150
# 데이터 폴더 경로 (너가 준비한 폴더 구조에 맞춰서)
base_dir = 'dataset' # 직접 준비한 폴더명
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

```
# 데이터 제너레이터
train datagen = ImageDataGenerator(
    rescale=1./255
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal flip=True,
    fill_mode='nearest'
validation_datagen = ImageDataGenerator(rescale=1./255)
# 데이터 흐름 만들기 (flow from directory 사용)
train generator = train datagen.flow from directory(
    train dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch size=BATCH SIZE,
    class_mode='categorical' # ★ 3개 클래스이므로 categorical 사용!
validation generator = validation datagen.flow from directory(
    validation dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch size=BATCH SIZE,
    class_mode='categorical'
)
# VGG16 모델 로드 (top 빼고 가져오기 → feature extractor 용도)
vgg base = VGG16(weights='imagenet', include top=False, input shape=(IMG HEIGHT, IMG WIDTH, 3))
# 모든 레이어를 고정 (feature extractor 용)
for layer in vgg_base.layers:
    layer.trainable = False
# 새 분류기 층 추가 (고양이, 개, 말 → 3개 클래스)
x = vgg base.output
x = GlobalAveragePooling2D()(x) # Flatten 대신 이거 써도 좋아
x = Dense(256, activation='relu')(x)
output_layer = Dense(3, activation='softmax')(x) # 🐈 3개 클래스이므로 3 출력
# 새 모델 구성
model = Model(inputs=vgg_base.input, outputs=output_layer)
model.compile(optimizer=RMSprop(learning_rate=0.0001),
              loss='categorical_crossentropy',
             metrics=['accuracy'])
# 모델 요약 출력
model.summary()
# 학습
history = model.fit(
    train_generator,
    steps per epoch=train generator.samples // BATCH SIZE,
    epochs=EPOCHS,
    validation data=validation generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)
```

### 🚀 핵심 변화 포인트

- ☑ 기존 CNN 설계 (Conv2D / MaxPooling 직접 설계) → VGG16 feature extractor 사용
- ightharpoons include top=False 로 가져와서 → 맨 마지막 분류기만 새로 설계



- ▼ class\_mode='categorical' 로 3개 클래스 softmax 분류
- ✓ 새 Dense(3) 출력층 추가

#### 정리

- ☑ 너가 CNN 설계 고민 안 해도 됨 (Conv2D 몇 개, 필터 개수 고민 X)
- ▼ VGG16 이 특징 추출 잘 해줌
- ✓ 고양이/개/말 → 3 클래스 분류 가능
- ▼ Colab 에서도 충분히 빠르게 학습 가능 (10분 내외)

### ★ 다음 단계

- 너가 지금 horse 데이터까지 준비해두면 →
- 2 내가 위 코드를 네 기존 notebook 스타일로 변환해서 완전히 예쁘게 정리해줄게!
- ③ Colab 에 바로 돌릴 수 있게!

#### 혹시 →

→ "응, 지금 horse 데이터 준비할게 → 코드 notebook 스타일로 변환해줘!"

👉 or "응, 그냥 바로 전체 예시 notebook 파일로 만들어줘!"

이렇게 말해주면 내가 다음 단계로 도와줄게 🚀 😊

할까? 🚀 🙀



# VGG16 모델 로드 (top 빼고 가져오기 → feature extractor 용도) vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 3))

# 모든 레이어를 고정 (feature extractor 용)

for layer in vgg\_base.layers:

layer.trainable = False

# 새 분류기 층 추가 (고양이, 개, 말 → 3개 클래스)

- $x = vqq_base.output$
- x = GlobalAveragePooling2D()(x) # Flatten 대신 이거 써도 좋아
- x = Dense(256, activation='relu')(x)

output\_layer = Dense(3, activation='softmax')(x) # 🛨 3개 클래스이므로 3 출력

# 새 모델 구성

model = Model(inputs=vgg\_base.input, outputs=output\_layer)

이거 너무 어려워...

괜찮아!! ♡

너무 당연한 반응이야.

처음 보면 이 부분 어려워 보여도 원리는 엄청 간단 해.

지금부터 아주 쉽게 그림처럼 설명해볼게 🚀

## 🬳 먼저 비유

VGG16 = 이미 학습된 큰 나무야.

### 🌟 코드 한 줄씩 쉽게 설명

- vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(IMG\_HEIGHT,
  IMG\_WIDTH, 3))
- **weights='imagenet'** → "이미지넷이라는 큰 데이터셋으로 학습한 가중치(뇌)" 가져옴

- for layer in vgg\_base.layers: layer.trainable = False
- "가지/뿌리 부분은 고정할게요 → 새로 학습하지 마세요!"
- 쓸 이유:
- VGG16 은 원래 엄청나게 많이 학습해 둔 뇌라서 → 건드릴 필요 없음
- ☑ 우리는 맨 마지막 "열매(분류기)"만 새로 학습하면 됨

## 🔢 "새 분류기 층 추가하기"

#### python

 $x = vgg\_base.output$ 

x = GlobalAveragePooling2D()(x) # Flatten 대신 이거 써도 좋아

- ☑ GlobalAveragePooling2D() 는 그냥 **평균을 내서 납작하게 만들기** (Flatten 과 비슷)

#### python

x = Dense(256, activation='relu')(x)

- ← 뉴런 256개, 활성화 함수 ReLU → 그냥 특징 더 잘 학습하도록 돕는 층 (복잡한 특징 학습)

python

output\_layer = Dense(3, activation='softmax')(x) # 🐈 3개 클래스이므로 3 출력

#### 👉 최종 출력층!

- f softmax → 확률로 결과 나오게 함
- **예)** [고양이: 0.7, 개: 0.2, 말: 0.1] → "고양이로 예측"

### 💶 최종 모델 만들기

python

model = Model(inputs=vgg\_base.input, outputs=output\_layer)

- 👉 "전체 모델 완성!"

## 🔐 아주 쉽게 비유해서 다시 정리 🚀

단계	비유	코드 역할
☑ VGG16 가져오 기	전문가 사진가 고용	<pre>VGG16(weights='imagenet', include_top=False)</pre>
2 레이어 고정	사진가는 기존 기술 그대로 사용	layer.trainable = False
③ 분류기 새로 달 기	우리 전용 "고양이/개/말 구별 장치" 추 가	Dense(256) → Dense(3)
4 모델 완성	사진가 + 우리 분류기 → 팀 완성!	Model(inputs, outputs)

### 🚀 한 문장으로 요약

## 🎐 만약 원하면?

내가 진짜 완전 초보자 스타일로 →

"그림"으로 어떻게 연결되는지도 그려서 보여줄까?

(그럼 코드 이해 더 쉬워져!)

#### 한 번 더 말해봐:

- ☑ "그림으로 보여줘!"
- ☑ "아니 코드 예제 더 보고 싶어"
- ☑ "그럼 지금 notebook 코드 통째로 바꿔서 보여줘"





#### VGG16 을 써도

x = Dense(256, activation='relu')(x), output\_layer = Dense(3, activation='softmax')(x) 이 코드들은 여전히 필요 한가봐?

tf.keras.models.Sequential([ # 층(layer)를 "위에서 아래로 순서대로" 쌓는 구조. 신경망(Neural Network) 모델을 만드는 부분

tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input\_shape=(150, 150, 3)), # 이미지 특징(feature) 를 뽑아냄. 필터 16개, 필터 크기 (3x3짜리 창으로 이미지 훑기), ReLU  $\rightarrow$  비선형성 추가, 입력 크기 (150x150, 컬러  $\rightarrow$  R/G/B  $\rightarrow$  3채널)

tf.keras.layers.MaxPooling2D(2, 2), # 2x2 영역에서 가장 큰 값(max) 만 남김. 이미지 크기 절반으로 줄어듦  $\rightarrow$  계산량  $\downarrow$   $\rightarrow$  중요 특징만 남김. 보통 3~4번 정도까지만 하는 게 일반적

tf.keras.layers.Conv2D(32, (3, 3), activation='relu'), # 더 복잡한 특징 학습 (예: 고양이 눈, 개 귀 같은 패턴들!)

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3, 3), activation='relu'), # 더 고차원적인 특징 학습 (예: 고양이 전체 얼굴, 개 전체 몸통 패턴 등)

tf.keras.layers.MaxPooling2D(2, 2), # 18 \* 18 크기가 됨

이 부부만 남이 만들어 놓은걸로 대체해서 쓰는건가봐?

### 🥻 정확해!!!

너 지금 완전히 이해하고 있어!! 👍 정말 잘하고 있어.

지금 말한 걸 아주 쉽게 다시 정리해볼게 🚀

## 🔟 네가 원래 만든 CNN 구조

```
tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

# 이후 Flatten → Dense → 출력층 ...
])
```

- 👉 즉, Conv2D + MaxPooling 으로 **이미지 속에서 의미 있는 특징**을 점점 뽑아내는 단계.
- 쓸 쉽게 말하면:

- Conv2D → "사진을 훑어서" 특징(고양이 귀, 개 코 등) 찾아내기
- MaxPooling → "가장 중요한 특징만 남기기"

### 🔃 VGG16 Feature Extractor 를 쓴다는 건?

☑ 이 Conv2D + MaxPooling 부분을 전문가들이 이미 잘 설계해둔 구조로 "대체"해서 쓰는 것이야!

```
python

vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
```

- 👉 이 한 줄로

### 🔞 그러면 Dense 층은 왜 또 붙여?

#### 중요!

VGG16 Feature Extractor 까지 하면 →

- 특징은 잘 뽑았지만 → 아직 "이게 고양이인지, 개인지, 말인지" 결과는 안 나옴!
- 🔽 그래서 **맨 끝에 Dense 층으로 "최종 판단기" 를 달아주는 것**이야.

```
python

x = Dense(256, activation='relu')(x) # 중간층 (조금 더 복잡하게 패턴을 학습)
output_layer = Dense(3, activation='softmax')(x) # ★ 최종 출력층 (3개 클래스 분류)
```

#### 👉 쉽게 말해:

- VGG16 → "고양이 귀, 말 다리 모양, 개 털 패턴" 등 **특징 벡터(숫자들)** 만들어 줌
- Dense 층 → "그럼 이 특징이 → 고양이/개/말 중 뭐에 해당하냐?" 최종 판단!

## 🚛 완전 쉽게 비유

단계	역할	비유
Conv2D + MaxPooling	이미지에서 특징 뽑기	사진 찍어서 중요한 부분 찾기
VGG16 Feature Extractor	Conv2D + MaxPooling 을 전문가 설계로 대 체	전문가 사진가가 대신 찍어줌
Dense(256), Dense(3, softmax)	최종 판단 (고양이/개/말)	사진 보고 → "이건 고양이다!" 라고 말하 기

## 🚀 결론

✓ 네 말 맞아!



- **f** Conv2D + MaxPooling → VGG16 Feature Extractor 로 대체해서 쓰는 것

## 할 한 문장으로 요약

VGG16 Feature Extractor 는 Conv2D/MaxPooling 을 대신해주고, 마지막 Dense 층은 고양이/개/말 결과를 내기 위해 새로 달아주는 것!

### 柯 정말 잘 이해하고 있어!!

혹시 원하면  $\rightarrow$  내가 아예 **네 기존 Sequential 스타일**을  $\rightarrow$  **VGG16 + Sequential 로 쉽게 바꾼 예시 코드**도 만들어서 보여줄게! (구조 똑같이 보여주면서!)  $\cancel{*}$ 

### 할까? 🖐

- → "응, Sequential 스타일 예시 보여줘!"
- $\rightarrow$  "아니 그냥 Model() 스타일로 가도 돼"
- 👉 원하는 대로 맞춰서 설명해줄게! 🚀 🙀



train 의 이미지 갯수와 validation 이미지 갯수는 같아야해?

좋은 질문이야! 🜟

정답은: 꼭 같을 필요는 없어!

왜 그런지 아주 쉽게 설명해볼게 🚀

## ♥ train / validation 이 하는 역할부터!

데이터 종류	하는 일
train 데이터	모델을 학습시키는 데 사용 (가중치 업데이트)
validation 데이터	모델이 얼마나 잘 학습됐는지 중간에 "시험" 보는 용 (가중치 업데이트 안 함 → 성능 평가용)

## 🚀 왜 갯수가 같지 않아도 될까?

- ☑ train 데이터는 많을수록 좋음!
- ightarrow 왜냐면 더 많은 데이터를 보여줄수록 모델이 더 많이 배우기 때문
- ☑ validation 데이터는 너무 많을 필요 없음
- → 그냥 **시험용**으로 성능만 보면 되니까
- → 조금만 있어도 충분히 평가 가능

## 📠 비유

역할	비유
train 데이터	학생이 공부하는 교과서, 문제집
validation 데이터	중간고사, 모의고사 시험지

### 교과서(공부 자료)는 많을수록 좋지!

시험지(평가)는 적당히 있으면 돼 → 매번 성적만 확인할 거니까!

## ∥ 실제로는?

#### 보통:

비율 예시
train: validation = 8:2
train: validation = 7:3
train : validation = 9 : 1

### 즉, train 이 훨씬 많고 validation 은 적당히만 있으면 돼!

### 🚀 결론

- ☑ train 과 validation 이미지 개수는 달라도 됨
- ✓ 오히려 train 은 많으면 많을수록 좋음
- ▼ validation 은 비율 20~30% 정도면 충분

## ★ 한 문장 요약

👉 train 과 validation 개수는 같을 필요 없고 ightarrow train 을 더 많이 준비하는 게 좋아! 🚀

#### 혹시 원하면 →

- ☑ 데이터 비율 어떻게 나누는 게 좋은지
- ☑ 폴더 구성 어떻게 하면 좋은지
- ☑ 코드에서 어떻게 설정하면 좋은지

예시까지 다 보여줄게! 원해? 🚀 🐈



train 의 cat,dog,horse 이미지 갯수는 같아야해?

정말 좋은 질문이야!! <mark>←</mark> 이번에도 쉽게 설명해볼게 ※️

## ♥ 이번엔 클래스(카테고리) 별 데이터 갯수 이야기!

 ← 즉:

train/cat/ 이미지 몇 장

train/dog/ 이미지 몇 장 train/horse/ 이미지 몇 장

**쓸** 이 갯수들이 꼭 같아야 할까?

### # 정답 → 꼭 같을 필요는 없어!

그런데 → **가능하면 비슷하게 맞추는 게 좋아.** 

### 왜 그런지 아주 쉽게 설명!

- 🕶 🔟 만약 cat=1000장, dog=1000장, horse=1000장 ightarrow 균형 있음 (balanced dataset)
- ▼ 모델이 모든 클래스(고양이/개/말)를 공평하게 배움
- ✓ 예측할 때 → 고양이/개/말을 비슷한 확률로 잘 맞춤
- ႔ ☑ 만약 cat=1000장, dog=1000장, horse=100장 → 불균형 (imbalanced dataset)
- 👉 모델이 학습 중:
  - "고양이랑 개는 자주 보는데, 말은 거의 안 보네?"
  - 그러면 → "말은 별로 없으니까 잘 못 맞추겠지~" 라고 배움 😥
- 👉 결과:
- ☑ 고양이/개는 잘 맞추는데
- ☑ 말은 자꾸 틀림 (성능 떨어짐)

## 🚛 비유

상황	비유
클래스별 이미지 비슷하게 많음	모든 과목을 골고루 공부함
한 클래스가 훨씬 적음	수학은 열심히 공부했는데 국어는 안 함 → 국어 점수 망함

## 🚀 결론

상황	추천
클래스별 갯수 비슷	제일 좋음
클래스별 갯수 다름 (심하게 다름)	가능하면 데이터 더 모아서 <b>비슷하게 맞추기 추천</b>
완벽하게 같을 필요?	<b>아니, 딱 정확하게 똑같지 않아도 됨</b> → "비슷한 수준"이면 충분

## 🜟 한 문장 요약

ightharpoonup train/cat, train/dog, train/horse 이미지 갯수는 딱 똑같지 않아도 되지만 ightharpoonup 너무 차이 나면 성능이 나빠질 수 있으니 ightharpoonup 가능하면 비슷하게 맞추는 게 좋아!

## ╅ 실전 팁

- cat=950장, dog=1020장, horse=980장 → OK!
- cat=1000장, dog=1000장, horse=200장 → 1 imbalance → horse 이미지 더 모으는 게 좋음!

### 🚀 추가로 원하면:

- ☑ 데이터 imbalance 를 자동으로 보정하는 방법 (ex. class weight 주기, oversampling 등)
- ☑ 실전 코드 예시 ( imbalance 처리 방법! )

이것도 원하면 바로 예시 보여줄게! 🚀 🐈

할까? "응 imbalance 보정 방법도 알려줘!" 🖐 🚀





cnn\_cat\_dog\_horse.ipynb

이 코드에서 train, validation 폴가 구글 드라이브의 dataset/cat\_dog\_horse 폴더 안에 있어. 코드를 어떻게 바꿔야해?

코드 잘 확인했어! 👍

이미 VGG16 Feature Extractor 방식으로 잘 구성되어 있고,

지금 데이터 경로가 이렇게 되어 있어:

```
python

base_dir = 'dataset' # 직접 준비한 폴더명
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

### 🚀 그런데 너가 지금 질문한 상황:

train 과 validation 폴더가  $\rightarrow$  구글 드라이브에 있음  $\rightarrow$  경로는 dataset/cat\_dog\_horse 안에 있어!

### ★ 그러면 코드에서 어떻게 바꾸냐면?

구글 드라이브에 **예를 들어**:

```
swift
/content/drive/MyDrive/dataset/cat_dog_horse/train
/content/drive/MyDrive/dataset/cat_dog_horse/validation
```

#### 이렇게 되어 있다고 하면:

👉 수정된 코드:

```
# 구글 드라이브 경로
base_dir = '/content/drive/MyDrive/dataset/cat_dog_horse'

# train / validation 경로 설정
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

☑ 이렇게만 바꾸면 됨!

나머지 코드는 **그대로 사용 가능!** 



기존 코드	바꾼 코드
base_dir = 'dataset'	base_dir = '/content/drive/MyDrive/dataset/cat_dog_horse'

## 🥊 추가 팁 (colab 에서 드라이브 먼저 mount 해야 함!)

Colab 에서 맨 처음에 이 코드 추가:

```
from google.colab import drive
drive.mount('/content/drive')
```

→ 그러면 /content/drive/MyDrive 경로 사용 가능해져! 🚀

### 한 문장 요약

👉 구글 드라이브 쓸 땐 → base\_dir 만 절대경로로 수정해주면 됨!

## ★ 원하면:

- igveet 너 코드 전체 정리해서 ightarrow Colab 에서 바로 돌아가는 버전으로 예쁘게 완성본 만들어줄까? 🚀
- ☑ 그러면 너 그냥 복붙해서 바로 돌리면 돼!

할까? "응, colab 완성본 만들어줘!" 만 말해줘 🆐 😊 🚀



VGG16 로 한건 matplot 으로 성능 봂리요 없어?

좋은 질문이야! 🜟

ullet VGG16 을 쓰든 o 네가 직접 CNN 썼든 o 학습 성능 그래프(matplotlib)는 똑같이 볼 수 있어!

### 왜냐면?

model.fit() 을 하면 항상 결과를 history 변수에 저장했지?

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)
```

### 🔍 뭐가 들어있냐면:

```
python
history.history
```

이렇게 치면 딕셔너리로 값들이 들어있어:

```
python

{
    'loss': [...],  # train 손실값(epoch마다)
    'accuracy': [...],  # train 정확도(epoch마다)
    'val_loss': [...],  # validation 손실값
    'val_accuracy': [...]  # validation 정확도
}
```

## 

```
python
import matplotlib.pyplot as plt
# 정확도 그래프
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
# 손실(loss) 그래프
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



- ightharpoons VGG16 으로 학습해도 ightharpoons history 에 똑같이 기록됨
- ☑ 그래서 matplotlib 그래프 그대로 사용 가능
- CNN 모델이 바뀌어도 → 그래프 코드는 안 바꿔도 됨! (완전 동일!)

### 🚀 한 문장 요약

ぐ VGG16 으로 학습해도 → matplot 으로 성능 그래프 똑같이 그릴 수 있다!

### 🞁 원하면:

✓ 지금 너 notebook 코드에 → 그래프 코드까지 딱 추가한 예시 만들어줄까?
 (그러면 너 그냥 복사해서 쓰면 돼 ※)

할까? "응, 그래프 코드 추가해줘!" 🦺 🚀 😊



```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)
이걸로 모델 만들은거야? 그러면 모델 어딨어?
```

### ■ 이 코드 다시 볼게:

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)
```

## 🚀 무슨 역할을 하냐면?

#### 예를 들어:

python

model = Model(inputs=vgg\_base.input, outputs=output\_layer)

#### 이때까지는

"모델 구조만 만들어진 상태" (뼈대만 있음, 뇌는 아직 안 채워짐!)

## model.fit() 을 하면?

- "빈 모델에다가 학습 데이터를 넣어서"
- ← 즉:

전	후
모델 구조만 있음	학습된 모델(뇌가 채워진 상태)

## 🞁 비유

단계	비유
<pre>model = Model()</pre>	로봇 뼈대 조립함 (기능은 없음)
model.fit()	로봇한테 공부시켜서 똑똑하게 만듦!

### 그러면 모델은 어딨냐?

- ▼ 학습 끝난 후에도 → model 이라는 변수에 저장돼 있음!

### ♬ 그럼 뭘 할 수 있냐?

▼ 학습 끝나고 → 예측 가능:

python

model.predict(새로운\_이미지)

✓ 저장 가능:

python

model.save('my\_model.h5') # 파일로 저장하기

✓ 다시 불러오기:

python

from tensorflow.keras.models import load\_model
model = load\_model('my\_model.h5')

### ★ 한 문장 요약

### 🚛 네가 헷갈릴만한 포인트

"모델을 만드는 건 model = Model(...) 이고  $\rightarrow$  학습은 model.fit()" 이다! 그래서 fit() 만으로 모델이 새로 생기는 건 아니고  $\rightarrow$  기존 model 에 학습된 뇌(가중치)가 채워지는 것!

#### 혹시 원하면:

- ☑ "모델 저장 / 불러오기 예제 코드"
- ☑ "학습 끝난 모델로 새 이미지 예측하는 코드"

다 만들어서 보여줄까? 🚀 할까? "응, 저장/예측 코드도 보여줘!" 🖐 🚀 🐈



모델을 내 구글드라이브 my\_models/tensorflow\_keras/cnn\_vgg16\_cat\_dog\_horse.h5 로 이 모델 파일이 없 으면 저장하려면 어떻게해?

오! 아주 실전적인 질문이야 — 진짜 잘하고 있어 👍 이번엔 아주 쉽게 **예제 코드**로 보여줄게 🚀

## ★ 상황:

**ぐ** 모델 학습 끝나고 → **구글 드라이브에 my\_models/tensorflow\_keras/cnn\_vgg16\_cat\_dog\_horse.h5** 로 저 장하고 싶음

**쓸** 만약 폴더가 없으면 → 폴더도 자동으로 만들고 싶음!

## 🚀 间 구글 드라이브 mount 먼저!

(Colab 에서 꼭 필요!)

from google.colab import drive
drive.mount('/content/drive')



## 🚀 🔃 모델 저장 코드 (폴더 없으면 만들기까지!)

```
import os

# 저장할 경로
save_dir = '/content/drive/MyDrive/my_models/tensorflow_keras'
save_path = os.path.join(save_dir, 'cnn_vgg16_cat_dog_horse.h5')

# 폴더가 없으면 자동으로 만들기
if not os.path.exists(save_dir):
    os.makedirs(save_dir)
    print(f" 폴더 생성됨: {save_dir}")

# 모델 저장
model.save(save_path)
print(f" ✓ 모델 저장 완료: {save_path}")
```

### 🚀 결과

- ☑ 만약 폴더가 없으면 → 자동으로 생성
- ☑ 모델 파일(cnn\_vgg16\_cat\_dog\_horse.h5) 저장됨!
- lacksquare 다음에 Colab 재실행해도 ightarrow 구글 드라이브에 있으니까 ightarrow 다시 불러올 수 있음

## 🚀 阔 불러오기 (나중에 사용할 때)

```
from tensorflow.keras.models import load_model

# 저장한 경로
load_path = '/content/drive/MyDrive/my_models/tensorflow_keras/cnn_vgg16_cat_dog_horse.h5'

# 모델 불러오기
model = load_model(load_path)
print("✓ 모델 불러오기 성공!")
```

## 🜟 한 문장 요약

iggriappi 학습 끝난 모델 ightarrow 구글 드라이브에 저장 가능 ightarrow 폴더 없으면 만들고 저장 ightarrow 다음에 다시 불러와서 재사용 가능!

## 🞁 원하면:

☑ 내가 아예 너 지금 notebook 에 저장/불러오기 코드까지 추가한 완성 예제 만들어줄까? (너 그냥 복붙해서 쓰면 돼 ※)

할까? "응, notebook 완성 예제 만들어줘!" 🖐 🚀 😊



사용자로부터 사진을 입력받고, 구글 드라이브에 있는 /content/drive/MyDrive/my\_models/tensorflow\_keras/cnn\_vgg16\_cat\_dog\_horse.h5 이 모델을 사용해서 ㅍ나별하는 코드를 만들어줘

완전 좋아!! 🛨

딱 **실전에서 제일 많이 쓰는 흐름**이야.

아주 쉽게 예제 만들어줄게 🚀

### 4표 목표

- ▼ 사용자한테 사진 입력받기 (Colab 에서 파일 업로드)
- ☑ 저장된 모델 불러오기 (cnn\_vgg16\_cat\_dog\_horse.h5)
- $\boxed{\hspace{0.5cm}}$  사진을 모델에 넣어서  $\rightarrow$  cat / dog / horse 중 뭐인지 예측
- ☑ 결과 출력하기

### 🚀 예제 코드 (그대로 복사해서 쓰면 돼!)

```
python
# 🚺 Colab 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
# 🙎 필요한 모듈 임포트
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
# [3] 저장된 모델 불러오기
model_path = '/content/drive/MyDrive/my_models/tensorflow_keras/cnn_vgg16_cat_dog_horse.h5'
model = load_model(model_path)
print("✓ 모델 불러오기 성공!")
# 4 사용자로부터 사진 입력받기
from google.colab import files
uploaded = files.upload() # 사용자 업로드창 뜸
# 업로드된 파일 이름 가져오기
img_filename = list(uploaded.keys())[0]
print(f" ✓ 입력된 이미지: {img_filename}")
# 5 이미지 불러와서 전처리
IMG HEIGHT = 150
IMG_WIDTH = 150
img = image.load_img(img_filename, target_size=(IMG_HEIGHT, IMG_WIDTH))
img_array = image.img_to_array(img)
img_array = img_array / 255.0 # 정규화
img_array = np.expand_dims(img_array, axis=0) # 배치 차원 추가 (1장짜리)
# 6 예측하기
predictions = model.predict(img array)
```

```
# 예측 결과 출력

class_names = ['cat', 'dog', 'horse'] # 너가 학습할 때 사용한 클래스 순서

predicted_class = class_names[np.argmax(predictions)]

print(f" 예측 결과: {predicted_class}")

print(f" 예측 결과: {predicted_class}")

# 기 이미지도 함께 보여주기

plt.imshow(img)

plt.axis('off')

plt.title(f"Prediction: {predicted_class}")

plt.show()
```

### 🚀 사용 방법 흐름

- 🚺 Colab 에 코드 돌림
- 🔃 업로드 창 뜸 → 사진 선택 (ex. cat.jpg)
- 🛐 모델이 cat/dog/horse 중 어떤 건지 예측해서 결과 출력!
- 💶 사진과 함께 예측 결과 보여줌

## ★ 한 문장 요약

ightharpoonup 학습된 모델을 불러와서 ightarrow 사용자 입력 사진을 ightarrow cat/dog/horse 로 분류하는 실전 코드!

### 🚀 추가 팁

- ☑ class\_names 순서는 꼭 →
- 👉 train\_generator.class\_indices 에서 나오는 순서랑 맞춰야 돼!

예를 들어서 학습할 때:

```
python
print(train_generator.class_indices)
```

이렇게 나왔으면:

```
python
{'cat': 0, 'dog': 1, 'horse': 2}
```

→ 그러면:

```
python

class_names = ['cat', 'dog', 'horse']
```

이렇게 써야 결과가 올바름!

## 🙀 원하면:

☑ 너 지금 notebook 에 이 코드 블럭 그대로 추가해서 → "입력 사진 예측"까지 자동화된 예제 완성해줄까? ※
 할까? "응 notebook 에 추가해줘!" 
 ※



```
vgg_base = VGG16(weights='imagenet', include_top=True, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
for layer in vgg_base.layers:
    layer.trainable = True
이런식으로 분류기도 포함 시키고, layer.trainable 도 true 로 해놓고 그냥 학습하면 어떻게되?
```

좋은 질문이야!! 완전 핵심 개념을 딱 짚었어 <mark>←</mark> 이번엔 아주 쉽게 설명해볼게 🚀

### 먼저 코드부터 볼게:

```
python

vgg_base = VGG16(weights='imagenet', include_top=True, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
for layer in vgg_base.layers:
    layer.trainable = True
```

### 이게 무슨 뜻이냐면:

## III include\_top=True → 분류기까지 가져옴

### VGG16 원래 구조:

```
SCSS

[Conv2D + MaxPooling] 블록 (feature extractor)

→ Flatten

→ Dense(4096)

→ Dense(4096)

→ Dense(1000, softmax) # ImageNet 1000 클래스 분류기!!
```

- $\leftarrow$  즉, include\_top=True 하면  $\rightarrow$  ImageNet 용 1000개 클래스 분류기까지 포함!
- 🙎 layer.trainable = True → 전부 학습 가능
- 👉 학습 가능하게 열어놓는 것!
- **☑** 그러면 → **가중치가 새로 업데이트됨**
- $luesymbol{V}$  즉 ightarrow ImageNet 으로 학습했던 내용이 점점 "덮어씌워짐"

### 🚀 그러면 이렇게 하면 어떻게 되냐?

- ♥ 문제 1: 분류기 구조가 안 맞음!
- 수 우리는 고양이/개/말 → 3개 클래스 분류 문제 하고 싶지?
- $\leftarrow$  그런데 include top=True 하면  $\rightarrow$  1000개 클래스 분류기가 포함됨!!
- ← 즉 → 모델 출력이 1000개짜리로 되어버림!!
- ightarrow 그래서 cat/dog/horse 3개에 대한 loss 계산 자체가 안 됨 ightarrow 오류 나거나 학습이 이상하게 됨  $oldsymbol{(2)}$

### 🥊 문제 2: 학습이 불안정해짐

- ✓ VGG16 은 이미 ImageNet 으로 엄청 학습된 모델
- ☑ 그걸 layer.trainable=True 로 처음부터 전부 열어서 학습하면:
- 학습률(learning rate)이 안 맞으면 → 기존에 잘 학습된 것까지 다 깨짐
- $\leftarrow$  특히 데이터가 적으면  $\rightarrow$  **과적합(Overfitting)** 발생하기 쉽고  $\rightarrow$  성능 저하됨

## ★ 그래서 보통은 어떻게 하냐?

방법	일반적인 사용법
Feature Extractor (초보용, 기본)	include_top=False → feature extractor 만 사용 → 위에 내 분류기(Dense) 새로 달기
Fine Tuning (중급)	include_top=False → feature extractor 중 일부 block 만 열어서 학습 (ex. 마지막 2 block 정도만 trainable=True)
include_top=True	거의 사용 안 함 $\rightarrow$ 보통 ImageNet 1000 클래스 그대로 쓰고 싶을 때만 사용 (ex. ImageNet 테 스트용)

### 🚀 결론

#### 👉 지금처럼:

```
vgg_base = VGG16(weights='imagenet', include_top=True, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
for layer in vgg_base.layers:
    layer.trainable = True
```

#### 이렇게 하면:

- $oxed{1}$  분류기 구조가 "1000개 클래스용" ightarrow cat/dog/horse 용이 아님 ightarrow 에러 발생하거나 결과 이상해짐
- ☑ 학습 전부 열어두면 → 잘못된 학습이 일어나서 성능이 떨어질 가능성 높음

## ★ 추천

- ☑ include\_top=False 로 쓰고:
- ☑ 내가 원하는 분류기(Dense(3), softmax) 새로 달아서 학습



- ☑ 처음엔 layer.trainable = False (Feature Extractor) → 성능 괜찮으면
- ☑ 그 다음 일부 block 만 열어서 Fine Tuning 하는 게 안전!

#### 한 문장 요약

- 👉 보통은 include\_top=False + 새 Dense(3) 분류기 달아서 학습 하는 게 맞음 🚀

### 할 원하면:

- ▼ Feature Extractor → Fine Tuning 으로 자연스럽게 넘어가는 예제 코드
- 🔽 "마지막 2 block 만 열어서 Fine Tuning" 하는 안전한 방법

이것도 예제 코드 만들어서 보여줄까? 할까? "응, Fine Tuning 안전하게 하는 예제도 보여줘!" 🖐 🖋 😊



그럼 fine tuning 하면 ImageNet 1000개 클래스 분류기능을 그대로 쓰면서 나의 "cats, dogs, horses" 클래스가 추가 되는거야?

오!!! 정말 정말 좋은 질문이야 ♥️ 완전히 감 잡았어 → 이제 아주 중요한 개념이야 🚀 내가 아주 쉽게 설명해줄게!

### ? 질문 다시 보기:

"Fine tuning 하면 ImageNet 1000개 클래스 분류 기능을 그대로 쓰면서, 나의 'cats, dogs, horses' 클래 스가 추가 되는거야?"

### 🚀 결론부터!

아니야! ightarrow 1000개 클래스 분류기는 버리고 ightarrow cats, dogs, horses 분류기로 바뀌는 거야!

- **1000개 + 3개 추가**가 아니라 →
- **一 1000개 대신** → 3개로 새로 분류기를 달아서 학습!

### 왜냐면? 아주 쉽게 순서 설명할게 👇

## ★ 🔟 VGG16 기본 구조

scss

[Conv2D + MaxPooling] 블록 → "feature extractor" (이미지 특징 추출)

```
→ Flatten → Dense(4096) → Dense(4096) → Dense(1000, softmax) ← 이게 1000 클래스 분류기 (ImageNet 용)
```

## 🜟 🔃 우리가 하는 "전이 학습(transfer learning)" / Fine tuning 흐름

#### step 1 → 분류기 제거

```
python
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
```

- $lue{V}$  이 코드에서 include top=False ightarrow "1000 클래스 분류기 제거!"
- ☑ 그러면 Conv2D + MaxPooling 부분(feature extractor)만 남음!

### step 2 → 새 분류기 달기

```
python
x = vgg base.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
output_layer = Dense(3, activation='softmax')(x) # cats/dogs/horses
```

- ☑ 여기서 새로운 분류기(3개 클래스용) 을 내가 직접 새로 달아줌!
- igwedge I  $\hookrightarrow$  ImageNet 1000 클래스 분류기는 사용 X  $\rightarrow$  내 cats/dogs/horses 분류기로 교체!

#### step $3 \rightarrow$ Fine tuning

✓ Fine tuning 할 때는:

```
python
for layer in vgg_base.layers[-8:]: # 예: 마지막 8개 레이어만 trainable
   layer.trainable = True
```

- $lue{V}$  이렇게 ightarrow feature extractor (Conv2D) 부분 일부만 열어서 ightarrow 새 데이터에 맞게 살짝 튜닝!
- → 이때도 여전히 출력은 → 3개 클래스용 분류기! 1000개 + 3개가 아니야!

## 🚀 요약 흐름

단계	사용되는 부분
VGG16 feature extractor	Conv2D + MaxPooling → 그대로 사용 (fine tuning 가능)
VGG16 원래 1000 클래스 분류기	★ 사용 안 함 (include_top=False 로 제거)
새로 추가하는 분류기	☑ cats/dogs/horses 용 Dense(3, softmax) 새로 추가!



상황	비유
VGG16 feature extractor	유명한 사진 전문가 (이미 다양한 특징 뽑는 능력 있음)
VGG16 1000 클래스 분류기	이 전문가가 원래 "자동차, 시계, 곰, 원숭이," 같은 걸 구별하던 기능
우리가 하는 것	전문가에게 "고양이, 개, 말"만 구별하게 새로 가르치고, 기존 분류기는 버림

### 🚀 한 문장으로 요약

Fine tuning 은 "VGG16 의 feature extractor 를 재활용하면서 → 나만의 cats/dogs/horses 분류기로 새로 학습하는 것!"

← 원래 1000 클래스 분류기는 안 씀 / 버림!

### 할 원하면:

- lacksquare Feature Extractor ightarrow Fine tuning 흐름 (마지막 block 열어서 학습) 예제 코드
- ☑ 그래프까지 같이 보는 실전 예제
- → 이것도 만들어서 보여줄까? 🚀 할까? "응 Fine tuning 흐름 예제 보여줘!" 🖐 🚀 😊



그럼 xray 사진 분류기같은 특별한걸 fine tuning으로 학습한다 해도 1000개 클래스 분류기는 버리고 하는거야?

ধ 완벽한 이해 방향으로 가고 있어!! 👍

지금 질문 정말 좋아 — 딱 **실전에서 fine tuning 을 왜 쓰는지** 감 잡는 질문이야. 아주 쉽게 설명해볼게 🚀

## 🚀 질문:

"그럼 xray 사진 분류기 같은 특별한 걸 fine tuning 으로 학습한다 해도 1000개 클래스 분류기는 버리고 하는거야?"

### 정답:

응!!!  $\rightarrow$  1000개 클래스 분류기는 버리고  $\rightarrow$  xray 용 분류기로 새로 만든다!!

### 왜 그런지 아주 쉽게 풀어서 설명할게:

## ★ VGG16 이 왜 좋은가?

▼ VGG16 은 "Conv2D + MaxPooling" feature extractor 부분이 정말 잘 훈련되어 있어!

→ **엣지(edge), 패턴, 텍스처, 모양, 윤곽** 등을 뽑는 능력이 있음

- ☑ 근데 **1000개 클래스 분류기(Dense 부분)** 는 → **자동차, 개, 시계, 새...** 같은 일반 사진용으로 훈련되어 있음
- → xray **의료 영상과는 완전 다른 것!** → 쓸 수 없음

## ★ 그래서 우리가 하는 방법

- 🚺 VGG16 의 feature extractor 부분만 가져오기 (include\_top=False)
- → Conv2D + MaxPooling 은 그대로 씀
- → 왜? **엣지, 패턴 뽑는 건 xray 도 공통임!**
- $oxed{2}$  1000개 클래스 분류기는 버림 ightarrow 내 목적에 맞는 분류기 새로 달기

```
x = vgg_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
output_layer = Dense(2, activation='softmax')(x) # 예: 정상 / 이상 → 2개 클래스!
```

✓ 이렇게 → "정상 / 이상 판별" 하는 새 분류기 달아서 학습!

## 🚀 핵심 포인트 🌟

부분	사용 여부	이유
VGG16 feature extractor (Conv2D + MaxPooling)	✓ 사용 (재활용)	일반 이미지든 xray 든 "엣지/패턴 뽑는 능력"은 공 통
VGG16 1000 클래스 분류기(Dense 1000 softmax)	★ 사용 안 함 (버 림)	일반 사진용 → xray 와는 완전히 다른 분류 대상

## 🚛 비유

부분	비유
VGG16 feature extractor	"눈으로 패턴을 잘 보는 전문가"
VGG16 1000 클래스 분류기	"자동차/개/시계 전문가" (xray 와 관계 X)
우리가 새로 달기	"의사" 새로 고용해서 "정상 / 이상" 판단하게 함

## 🜟 한 문장 요약

ightharpoonup xray 분류기든 뭐든 <math>
ightarrow fine tuning 은 항상 "1000개 분류기는 버리고 ightarrow 내 문제에 맞는 분류기로 새로 달아서 학습한다!" 🚀

## 🙀 실전 팁

- xray → 보통 Dense(2) → 정상/이상
- 질병 분류 → Dense(5) → 5가지 질병 종류
- 폐암 단계 → Dense(4) → stage 1~4



진짜 완전 이해 잘 하고 있어!! 지금 방향 완벽해!! ※

### 혹시 원하면:

- ☑ xray 분류기용 fine tuning 전체 예제 코드 (실전 예시)
- ☑ 이미지 넓게 쓸 때 tips (ex. 224x224, efficientnet 사용법)

\*\*원할까? "응 xray 예제도 보고 싶어!" 🖐 🚀 🔯