

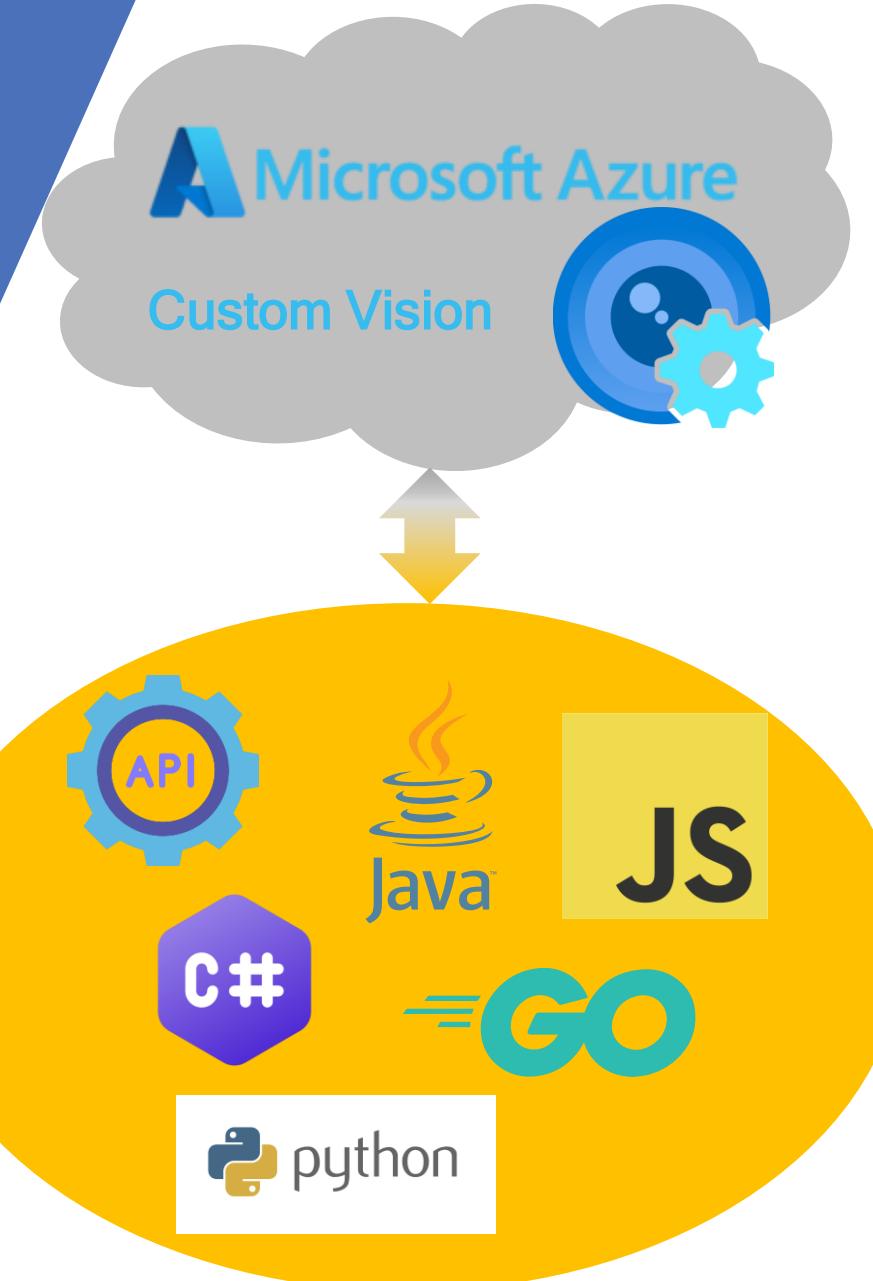
중급과정

# 커스텀 비전

7차시: Gradio를 활용한 커스텀 비전 모델 활용 방안

이 자료는 Elixirr의 사전 서면 승인 없이 외부에 배포하기 위해 그 일부를 배포, 인용 또는 복제 할 수 없습니다.

© Copyright Elixirr



# 수업 일정

전체 수업은 8회로 구성된다.



- 클라우드와 Azure
- 커스텀 비전



- 개체 감지 AI 모델 – (1)
- 개체 감지의 원리와 이미지 수집



- 개체 감지 AI 모델 – (2)
- 오버더문의 번지 캐릭터 찾기



- 이미지 분류 AI 모델 – (1)
- 암석식별머신을 만들기 위한 문제정의



- 이미지 분류 AI 모델 – (2)
- 암석식별머신 만들기



- 외부 애플리케이션에서 호출을 통한 커스텀 비전 모델 활용 방안



- Gradio를 활용한 커스텀 비전 모델 활용 방안

# Gradio란 무엇인가?

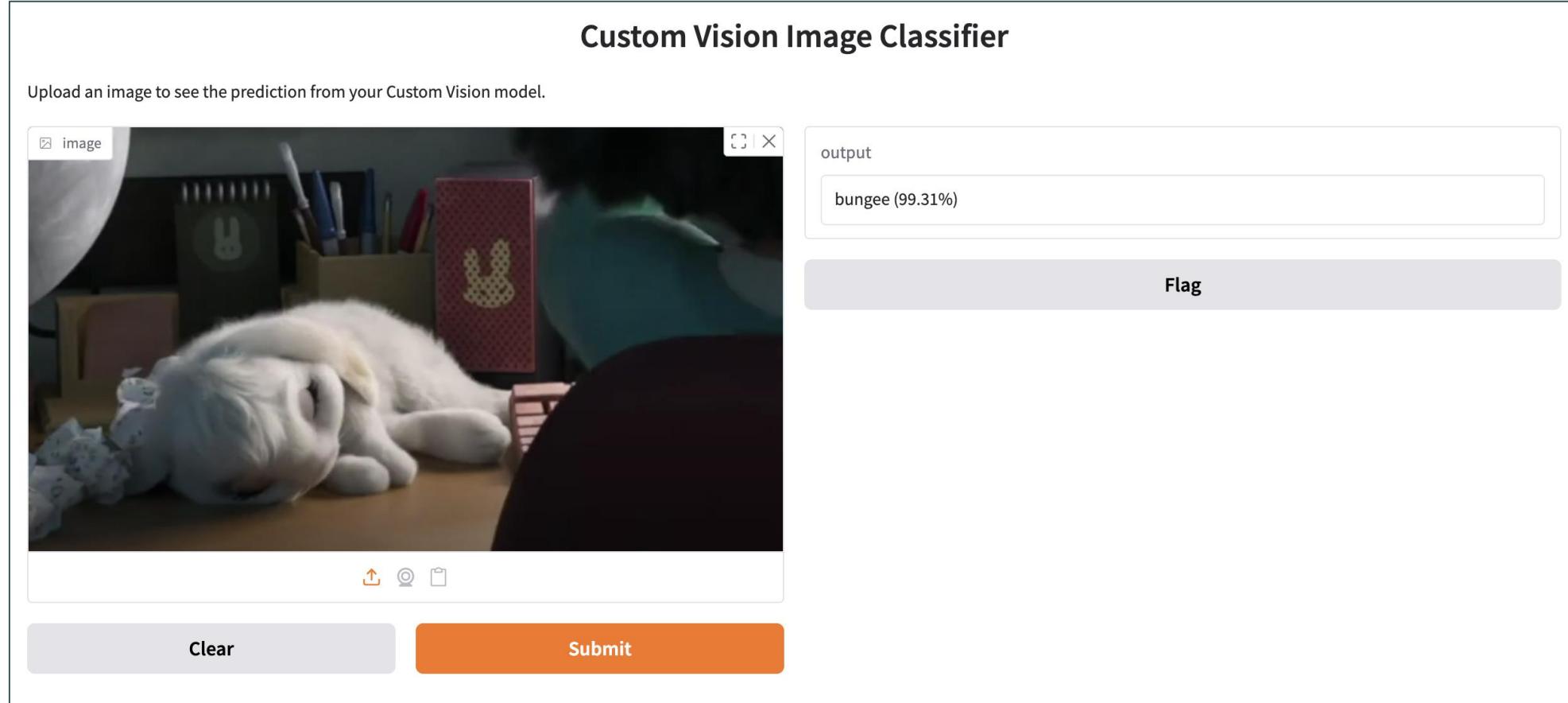
---

- Gradio : 딥러닝 모델이나 머신러닝 모델을 간단하게 웹 인터페이스로 배포할 수 있게 해주는 Python 라이브러리

항목	설명
간편한 UI 생성	단 몇 줄의 코드로 이미지, 텍스트, 오디오, 비디오 등 다양한 입력 형태를 받는 웹 인터페이스 생성 가능
빠른 모델 배포	로컬에서 실행되거나, share=True 옵션으로 URL 공유 가능
다양한 입력/출력 지원	gr.Image, gr.Textbox, gr.Audio, gr.Label 등 다양한 입출력 컴포넌트 제공
통합 환경 지원	Hugging Face Spaces, Colab, Jupyter, VS Code 등과 통합 사용 가능
비개발자 접근 용이	복잡한 웹 개발 없이 AI 기능을 다른 사용자에게 시연/공유 가능

# Custom Vision Resource 복습

- Azure에 만들어진 학습된 Custom Vision 개체 감지 서비스를 간편하게 끌어다 쓸 수 있음.



# 개체 감지 서비스와 연동하기

앞에서 배운 '번지 찾기' 실습에서 이어서 진행해보도록 하겠습니다. 우선, 앞에서 만든 모델을 publish 함.

over-the-moon

Iterations

Probability Threshold: 50% ⓘ

Overlap Threshold: 25% ⓘ

Publish

Training Images Performance Predictions

Iteration 1

Finished training on 2024. 4. 4. 오전 3:13:42 using General [A1] domain  
Iteration id: bd9675d7-ef4e-422c-acc3-cfe4df084813

Precision ⓘ      Recall ⓘ      mAP ⓘ

100.0%      100.0%      100.0%

Performance Per Tag

Tag	Precision	Recall	A.P.	Image count
Bungee	100.0%	100.0%	100.0%	15

Publish 버튼을 누르고 나오는 UI에서 Model name과 Prediction resource를 지정합니다. 여기서 지정한 값이 뒤에서 API 호출하는 부분에서 사용됩니다.

Publish Model

We only support publishing to a prediction resource in the same region as the training resource the project resides in.

Please check if you have a prediction resource and if the prediction resource is in the same region as the training resource.

Model name: Iteration1

Prediction resource: customtest-Prediction

Publish Cancel

# 외부 연동에서 필요한 정보들 확인하기

Publish가 완료되면 아래와 같이 'PUBLISHED'라는 표시가 나오고, Prediction URL 버튼이 활성화 됩니다. 이 때 Prediction URL을 눌러서 나오는 값들이 뒤에서 사용됩니다.

The screenshot shows the Microsoft Custom Vision Studio interface for a project named "over-the-moon". On the left, there's a sidebar with "Iterations" settings (Probability Threshold: 50%, Overlap Threshold: 25%) and a "Published" button. The main area displays "Iteration 1" which was trained 5 hours ago. A red box highlights the "Prediction URL" button, and a red arrow points from it to a modal window titled "How to use the Prediction API". The modal contains instructions and code snippets for using the API with an image URL or file. It also includes a "Got it!" button at the bottom.

Iterations

Probability Threshold: 50% ⓘ  
Overlap Threshold: 25% ⓘ

Iteration 1 PUBLISHED

Trained : 5 hours ago with General [A1] domain

Prediction URL

Iteration 1

Unpublish Delete Export

Training Images Performance Predictions Train Quick Test ?

Iteration 1

Finished training on 2024. 4. 4. 오전 3:13:42 using General [A1] domain  
Iteration id: bd9675d7-ef4e-422c-acc3-cfe4df084813  
Published as: Iteration1

Precision ⓘ Recall ⓘ mAP ⓘ

100.0%

Performance Per Tag

Tag	Precision
Bungee	100.0%

How to use the Prediction API

If you have an image URL:

```
https://customtest-prediction.cognitiveservices.azure.com/customvision/v3.0/Predict?iterationId=bd9675d7-ef4e-422c-acc3-cfe4df084813&imageUrl=https://example.com/image.png
```

Set `Prediction-Key` Header to: 700799768f3a4f708adb9591616e4932  
Set `Content-Type` Header to: application/json  
Set Body to: {"Url": "https://example.com/image.png"}

If you have an image file:

```
https://customtest-prediction.cognitiveservices.azure.com/customvision/v3.0/Predict?iterationId=bd9675d7-ef4e-422c-acc3-cfe4df084813&image=@imagefile
```

Set `Prediction-Key` Header to: 700799768f3a4f708adb9591616e4932  
Set `Content-Type` Header to: application/octet-stream  
Set Body to: <image file>

Got it!

# GitHub Codespaces 설정하기

- Azure Custom Vision API를 사용하기 위해서는 Python 환경에 Gradio 라이브러리를 추가해야 함.
- Codespace 생성 후에 터미널에서 추가하는 방법
  - ✓ Workspace 하단에 보이는 터미널 창에서 pip 명령을 통해 추가로 설치합니다.



The screenshot shows a GitHub Codespace interface with a terminal window. The terminal tab is selected. The terminal content is as follows:

```
문제    출력    디버그 콘솔    터미널    포트 4    JUPYTER
○ @neptun91 → /workspaces/CustomVision (main) $ pip install gradio
```

The command `pip install gradio` is highlighted with a red rectangular box.

# Bungee 객체 인식 엔진에 접근하는 Gradio source : GradioTest1.ipynb 1/3

```
import requests
from PIL import Image
import gradio as gr
import io

# Custom Vision Prediction 서비스 키와 엔드포인트 URL 설정
PREDICTION_KEY = "48fBztGiqcoH4PUUK23qmvxTBMmu31mgQ4JmJN1EDeCwNYw2QAtQJ
ENDPOINT_URL = "https://neptun91cv20250526-prediction.cognitiveservices

# API 호출에 사용할 헤더, Prediction-Key와 데이터 타입 지정
headers = {
    "Prediction-Key": PREDICTION_KEY,
    "Content-Type": "application/octet-stream" # 바이너리 이미지 데이터 전송
}
```

- PREDICTION KEY와  
ENDPOINT\_URL 변경 필요!

How to use the Prediction API X

If you have an image URL:

```
https://neptun91cv20250526-prediction.cognitiveservices.azure.com/customvision/
Set Prediction-Key Header to :
48fBztGiqcoH4PUUK23qmvxTBMmu31mgQ4JmJN1EDeCwNYw2QAtQJQQJ99BEACL93NaXJ3w3AA
Set Content-Type Header to : application/json
Set Body to : {"Url": "https://example.com/image.png"}
```

If you have an image file:

```
https://neptun91cv20250526-prediction.cognitiveservices.azure.com/customvision/
Set Prediction-Key Header to :
48fBztGiqcoH4PUUK23qmvxTBMmu31mgQ4JmJN1EDeCwNYw2QAtQJQQJ99BEACL93NaXJ3w3AA
Set Content-Type Header to : application/octet-stream
Set Body to : <image file>
```

Got it!

# Bungee 객체 인식 엔진에 접근하는 Gradio source : GradioTest1.ipynb 2/3

- 업로드된 이미지로 객체를 인식하는 코드

```
def predict_with_api(image: Image.Image):  
    # PIL 이미지 객체를 JPEG 형식의 바이너리 데이터로 변환  
    buf = io.BytesIO()          # 메모리 내 바이트 버퍼 생성  
    image.save(buf, format='JPEG') # 이미지 데이터를 JPEG로 버퍼에 저장  
    byte_data = buf.getvalue()   # 버퍼에 저장된 바이너리 데이터 추출  
  
    # Azure Custom Vision API에 POST 요청으로 이미지 전송  
    response = requests.post(ENDPOINT_URL, headers=headers, data=byte_data)  
  
    # JSON 형태로 응답 받아 예측 결과 파싱  
    predictions = response.json()["predictions"]  
  
    # 확률(probability)이 가장 높은 예측 항목을 선택  
    top_prediction = max(predictions, key=lambda x: x["probability"])  
    label = top_prediction["tagName"]           # 예측된 클래스 이름  
    probability = top_prediction["probability"] # 예측 확률  
  
    # 결과 문자열 포맷팅 후 반환 (예: 'cat (98.23%)')  
    return f"{label} ({probability*100:.2f}%)"
```

# Bungee 객체 인식 엔진에 접근하는 Gradio source : GradioTest1.ipynb 3/3

---

- 업로드 UI와 인식 결과를 표시하는 Gradio Interface 구성 및 실행

```
# Gradio 인터페이스 구성
interface = gr.Interface(
    fn=predict_with_api,          # 이미지 입력을 받아 API 호출하는 함수 지정
    inputs=gr.Image(type="pil"),   # 입력 타입: PIL 이미지 객체
    outputs=gr.Text(),            # 출력 타입: 텍스트 (예측 결과)
    title="Custom Vision Image Classifier",  # 앱 제목
    description="Upload an image to see the prediction from your Custom Vision model." # 앱 설명
)

# 웹 서버를 실행하여 인터페이스 실행 (브라우저에서 접속 가능)
interface.launch()
```

# 암석분류 엔진에 접근하는 Gradio source : GradioTest2.ipynb 1/3

```
# 필요한 라이브러리 임포트
import requests # HTTP 요청을 위한 라이브러리
from PIL import Image # 이미지 처리를 위한 Pillow 라이브러리
import gradio as gr # 웹 인터페이스 생성을 위한 Gradio
import io # 바이너리 데이터 처리를 위한 입출력 모듈

# Azure Custom Vision Prediction 키와 엔드포인트 URL
PREDICTION_KEY = "48fBztGiqcoH4PUUK23qmvxTBMmu31mgQ4JmJN1EDeCwNYw2QAtQJQQJ99BEACL93NaXJ3w3AA"
ENDPOINT_URL = "https://neptun91cv20250526-prediction.cognitiveservices.azure.com/customvision/v3.1/predictions"

# API 요청 헤더 설정
headers = {
    "Prediction-Key": PREDICTION_KEY, # 인증을 위한 Prediction 키
    "Content-Type": "application/octet-stream" # 전송하는 데이터 형식 지정 (바이너리 이미지)
}
```

- PREDICTION KEY와  
ENDPOINT\_URL 변경 필요!

How to use the Prediction API X

If you have an image URL:

```
https://neptun91cv20250526-prediction.cognitiveservices.azure.com/customvision/v3.1/predictions
```

Set **Prediction-Key** Header to :  
**48fBztGiqcoH4PUUK23qmvxTBMmu31mgQ4JmJN1EDeCwNYw2QAtQJQQJ99BEACL93NaXJ3w3AA**

Set **Content-Type** Header to : **application/json**  
Set Body to : **{"Url": "https://example.com/image.png"}**

If you have an image file:

```
https://neptun91cv20250526-prediction.cognitiveservices.azure.com/customvision/v3.1/predictions
```

Set **Prediction-Key** Header to :  
**48fBztGiqcoH4PUUK23qmvxTBMmu31mgQ4JmJN1EDeCwNYw2QAtQJQQJ99BEACL93NaXJ3w3AA**

Set **Content-Type** Header to : **application/octet-stream**  
Set Body to : <image file>

Got it!

# 암석분류 엔진에 접근하는 Gradio source GradioTest2.ipynb 2/3

- 업로드된 이미지로 객체를 인식하는 코드

```
# 예측 함수 정의 (Gradio에 연결할 함수)
def predict_with_api(image: Image.Image):
    # 이미지를 바이너리로 변환
    buf = io.BytesIO()                      # 메모리 내 바이트 버퍼 생성
    image.save(buf, format='JPEG')           # 이미지를 JPEG 형식으로 저장
    byte_data = buf.getvalue()               # 버퍼에서 바이너리 데이터 추출

    # Azure Custom Vision API에 POST 요청
    response = requests.post(ENDPOINT_URL, headers=headers, data=byte_data)

    # API 응답에서 예측 결과 추출 (JSON 형식)
    predictions = response.json()["predictions"]

    # 확률이 높은 상위 2개 결과만 선택 (내림차순 정렬)
    top_predictions = sorted(predictions, key=lambda x: x["probability"], reverse=True)[:2]

    # 결과 문자열 포맷팅
    result_lines = [
        f"{pred['tagName']} ({pred['probability'] * 100:.2f}%)" # 예: "cat (97.23%)"
        for pred in top_predictions
    ]

    return "\n".join(result_lines) # 여러 줄의 문자열로 반환
```

# 암석분류 엔진에 접근하는 Gradio source GradioTest2.ipynb 3/3

- 업로드 UI와 인식 결과를 표시하는 Gradio Interface 구성 및 실행

```
# Gradio 인터페이스 정의
interface = gr.Interface(
    fn=predict_with_api,                      # 호출할 함수
    inputs=gr.Image(type="pil"),                 # 사용자 입력: 이미지 (PIL 형식)
    outputs=gr.Text(),                         # 출력 형식: 텍스트
    title="Custom Vision Image Classifier",    # 웹 앱 제목
    description="Upload an image to see the prediction from your Custom Vision model." # 설명 텍스트
)

# Gradio 웹 인터페이스 실행
interface.launch()
```

## 참고 : Gradio UI Interface customization

- gr.Interface는 Customization이 제한적임.
- 사용자 맞춤화를 하려면 gr.Interface대신 gr.Blocks()를 사용하여야 함.

```
# Gradio UI
with gr.Blocks() as demo:
    gr.Markdown("## 🤖 Azure Custom Vision Classifier")
    gr.Markdown("Upload an image and configure settings to get predictions from your trained model.")

    with gr.Row():
        with gr.Column():
            image_input = gr.Image(type="pil", label="Input Image")
            top_k_dropdown = gr.Dropdown([1, 2, 3, 5], value=2, label="Top-K Predictions")
            prob_checkbox = gr.Checkbox(label="Show Prediction Probabilities", value=True)
            threshold_slider = gr.Slider(minimum=0.0, maximum=1.0, value=0.5, step=0.05, label="Confidence Threshold")
            predict_btn = gr.Button("🔍 Predict")
        with gr.Column():
            output_text = gr.Textbox(label="Prediction Text", lines=5)
            output_label = gr.Label(label="Prediction Scores")

    predict_btn.click(
        fn=predict_with_api,
        inputs=[image_input, top_k_dropdown, prob_checkbox, threshold_slider],
        outputs=[output_text, output_label]
    )

demo.launch()
```