Lab 7: ASR, Facial keypoints, and Face emotion recognition

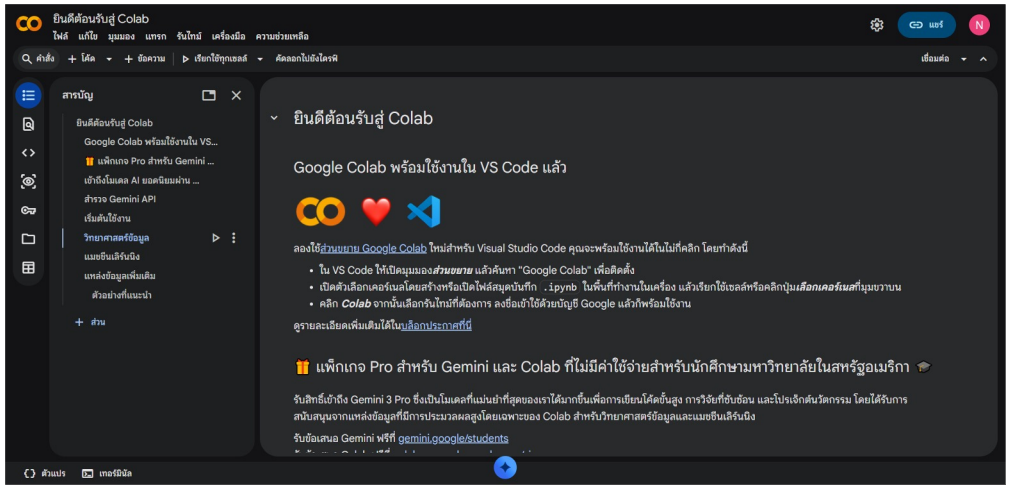
3099704 AI for Digital Health (2025/2)

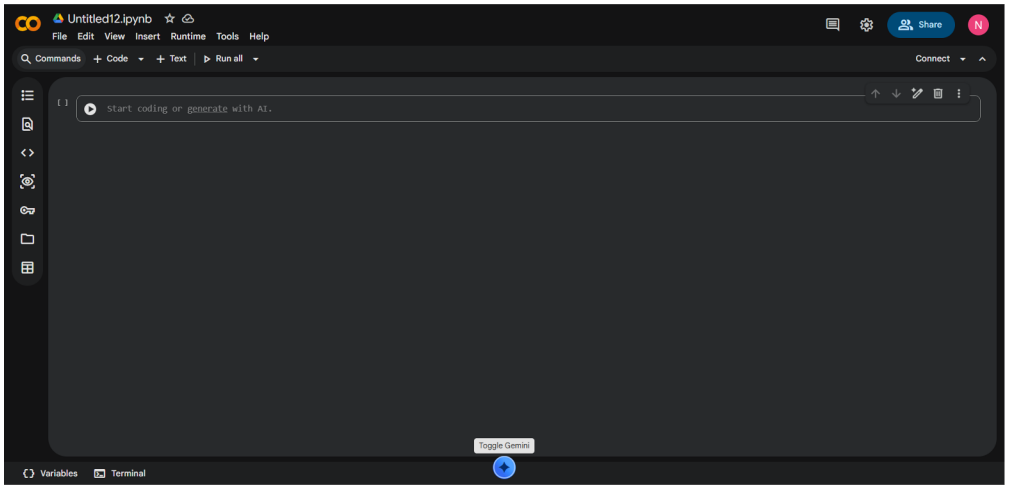
Objective

- Use pre-trained models to perform Thai Automatic Speech Recognition (ASR) and face recognition.
- Explore additional AI topics, such as model interpretability.

Material

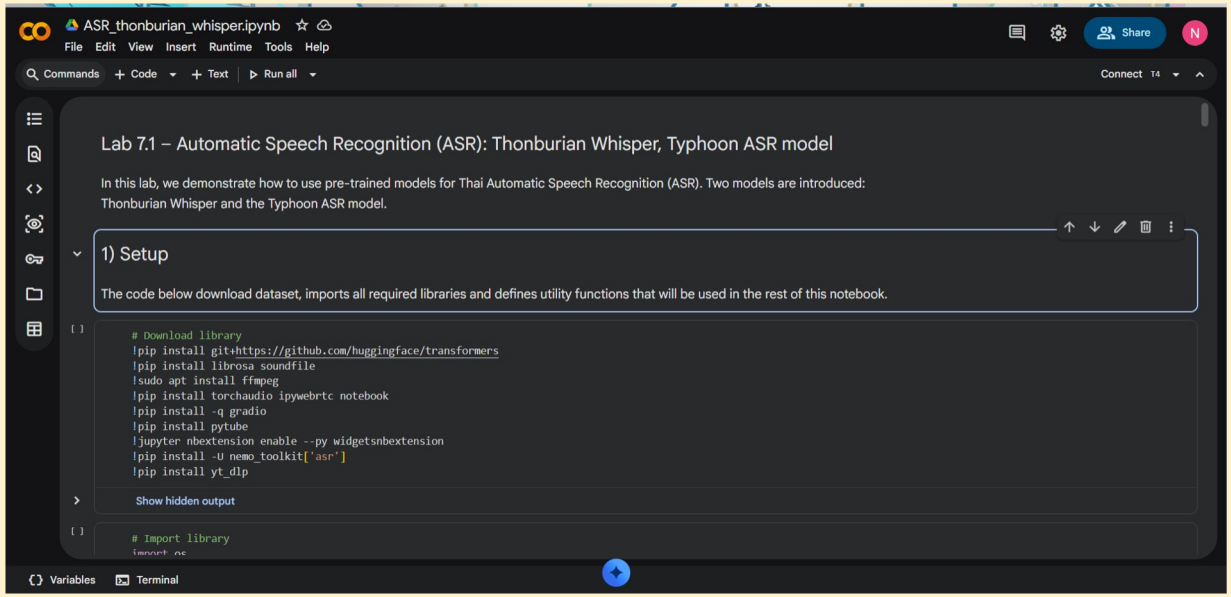
- With **Google Colab**, you don't need to install any software. All you need is a Google account, and you can start using it right away. Simply visit: https://colab.research.google.com/ or select NEW NOTEBOOK to start a new file.





Lab 7.1: Automatic Speech Recognition (ASR)

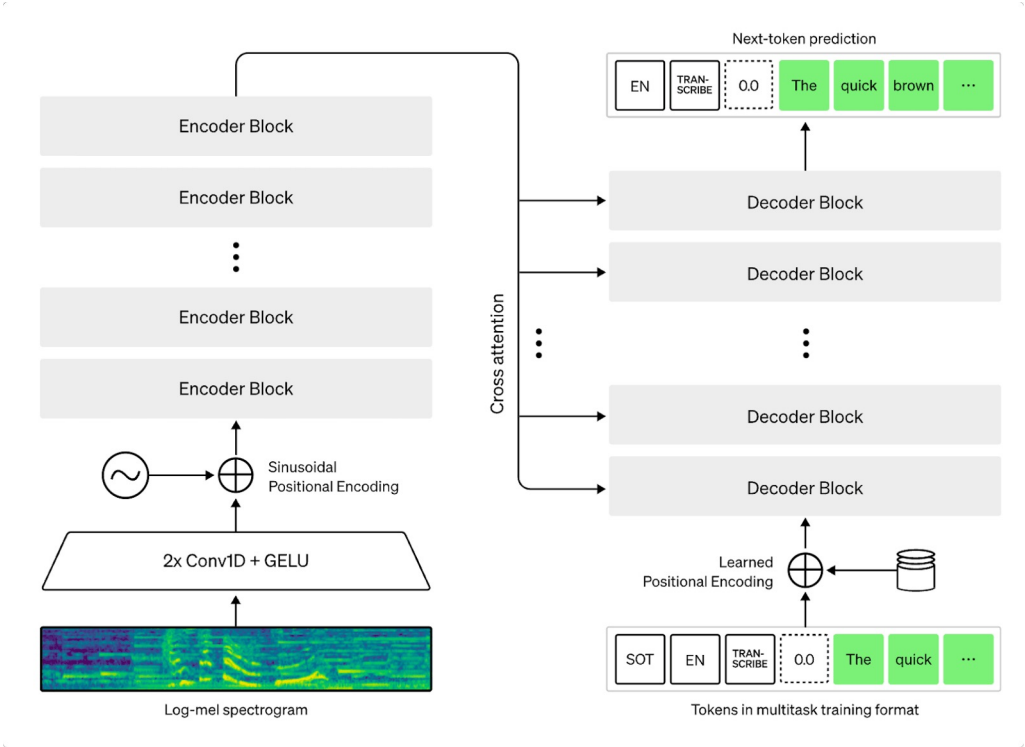
In this lab (Lab_7_1_ASR.ipynb), we demonstrate how to use pre-trained models for Thai Automatic Speech Recognition (ASR). Two models are introduced: Thonburian Whisper and the Typhoon ASR model.



1) Thonburian Whisper (looloo, 2022)

- A Thai ASR model based on OpenAI Whisper, fine-tuned with Thai speech datasets **(e.g., Commonvoice 13, Gowajee corpus, Thai Elderly Speech, Thai Dialect datasets).**

- Designed to work well in real-world conditions (background noise) and in **specialized domains like healthcare/medical and finance**.



2) Typhoon ASR (SCB10X, 2025)

- Open-source Thai streaming ASR optimized for low-latency transcription.
- Built on FastConformer Transducer (RNNT) for real-time performance.
- Designed to run efficiently on CPU (good for local / limited hardware).

How it differs from Thonburian Whisper

- **Thonburian Whisper:** Whisper-based, strong general transcription and robust to noise; typically used in **batch/recording** mode.
- **Typhoon ASR: Streaming / real-time** focus; faster and more efficient for live transcription.

How to load model

Thonburian Whisper (pipeline)

```
MODEL_NAME = "biodatlab/whisper-th-medium-timestamp"
lang = "th"

device = "cuda:0" if torch.cuda.is_available() else "cpu"

# Load processor explicitly
processor = WhisperProcessor.from_pretrained(MODEL_NAME)

pipe = pipeline(
    task="automatic-speech-recognition",
    model=MODEL_NAME,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    chunk_length_s=30,
    device=device,
    return_timestamps=True,
)
```

Typhoon ASR (NeMo)

```
# Select processing device
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f"Using device: {device}")

# Load Typhoon ASR Real-Time model
print("Loading Typhoon ASR Real-Time...")
asr_model = nemo_asr.models.ASRModel.from_pretrained(
    model_name="scb10x/typhoon-asr-realtime",
    map_location=device
)
```

Data

Input 1: Recorded voice (from the notebook)

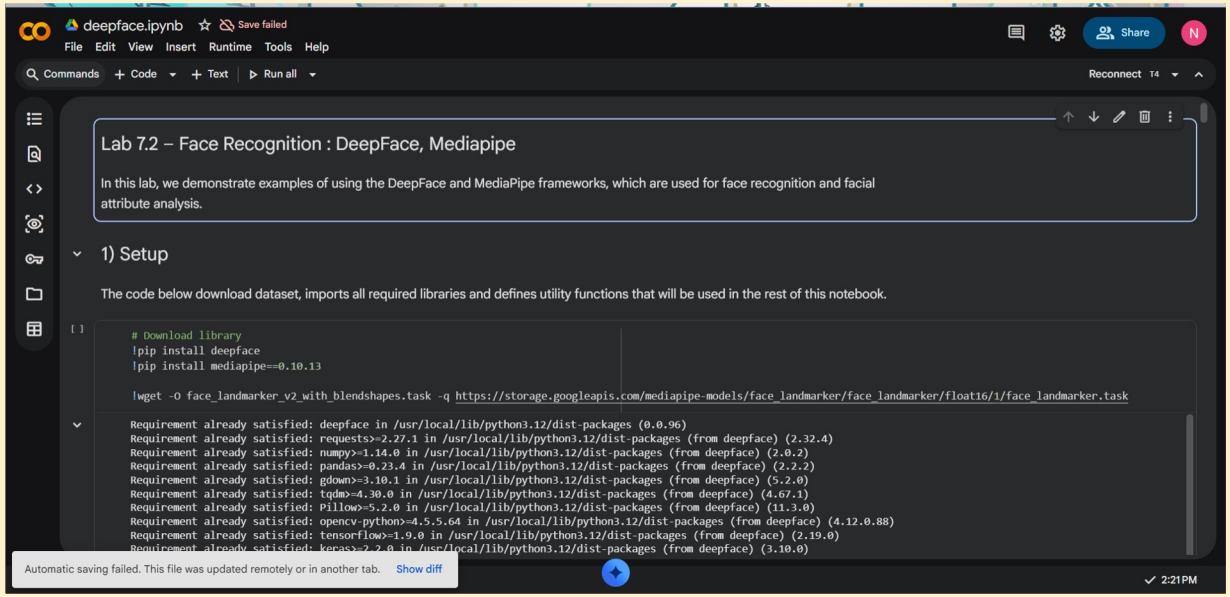
- File: recorder.mp3
- Created by recording your microphone inside Colab

Input 2: Uploaded / existing audio file

- You choose any audio file you want (e.g., .mp3 / .wav) and upload it into the .ipynb (Colab).

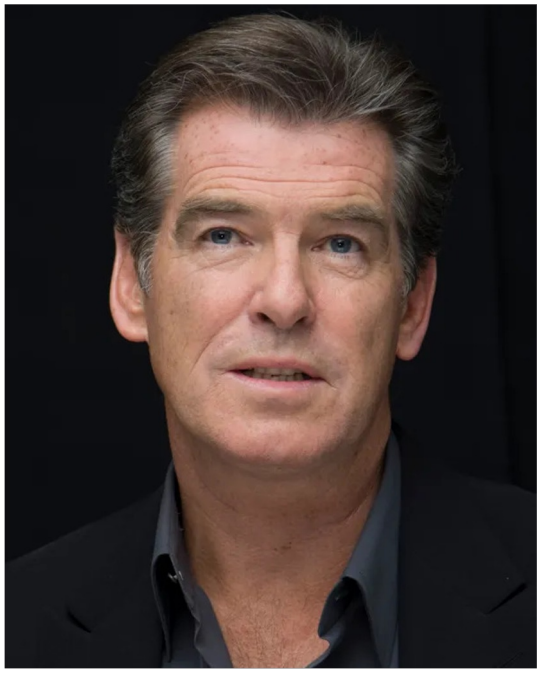
Lab 7.2: Face Recognition

In this lab (Lab_7_2_FaceRecognition.ipynb), we demonstrate examples of using the DeepFace and MediaPipe frameworks, which are used for face recognition and facial attribute analysis.



Data

For this lab, we use **img.zip**, a small set of face images of famous public figures (known for their handsome faces?) as our demo data.



DeepFace

DeepFace is a Python library that load pre-train deep learning.

It can:

- Perform face recognition / verification (compare faces to see if they are the same person) using face embeddings
- Do basic facial attribute analysis (e.g., age/gender/emotion estimates) as a demo feature

In this lab, we will:

1. **Convert** it into a face embedding (a numeric “face signature”)
2. **Compare embeddings** to match / verify identities
3. **Analyze facial attributes** from images, including age, gender, race, and emotion.

DeepFace (Output)

DeepFace.represent

```
# embedding vector
print(f"embedding len: {len(embedding)}")
print(embedding)
```

```
embedding len: 512
[1.6526964902877808, 1.0965301990509033, 1.8078064918518066, 0.12550930678844452, -1.5757787227630615, 1.791746735572815,
```

DeepFace (Output)

DeepFace.verify

```
{'verified': True,
 'distance': 0.266768,
 'threshold': 0.3,
 'confidence': 57.27,
 'model': 'Facenet512',
 'detector_backend': 'opencv',
 'similarity_metric': 'cosine',
 'facial_areas': {'img1': {'x': 115,
   'y': 113,
   'w': 396,
   'h': 396,
   'left_eye': (378, 273),
   'right_eye': (241, 262)},
  'img2': {'x': 508,
   'y': 122,
   'w': 475,
   'h': 475,
   'left_eye': None,
   'right_eye': None}},
 'time': 2.71}
```

DeepFace.analyze

```
[{'age': 45,
  'region': {'x': 115,
   'y': 113,
   'w': 396,
   'h': 396,
   'left_eye': (378, 273),
   'right_eye': (241, 262)},
  'face_confidence': 0.9,
  'gender': {'Woman': np.float32(0.47135082), 'Man': np.float32(99.528656)},
  'dominant_gender': 'Man',
  'race': {'asian': np.float32(0.0056317816),
   'indian': np.float32(0.0003178974),
   'black': np.float32(3.2617309e-06),
   'white': np.float32(99.27343),
   'middle eastern': np.float32(0.34461755),
   'latino hispanic': np.float32(0.3760022)},
  'dominant_race': 'white',
  'emotion': {'angry': np.float32(11.738707),
   'disgust': np.float32(0.002389708),
   'fear': np.float32(5.2961206),
   'happy': np.float32(0.823493),
   'sad': np.float32(22.521772),
   'surprise': np.float32(0.091061),
   'neutral': np.float32(59.526455)},
  'dominant_emotion': 'neutral'}]
```

MediaPipe (Google)

is a toolkit for real-time computer vision.

In this lab, we will:

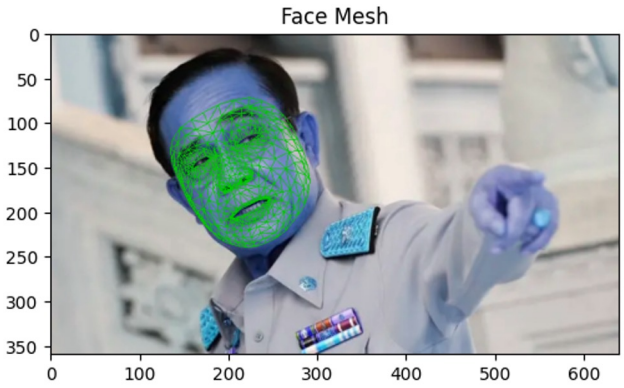
1. **Detect** a face in an image
2. **Extract facial landmarks** / mesh points (geometry of the face)
3. **Visualize the results** (overlay landmarks/mesh on the image)

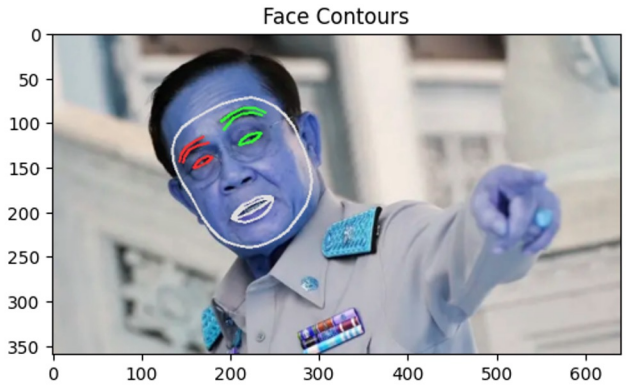
How **MediaPipe** differs from **DeepFace**:

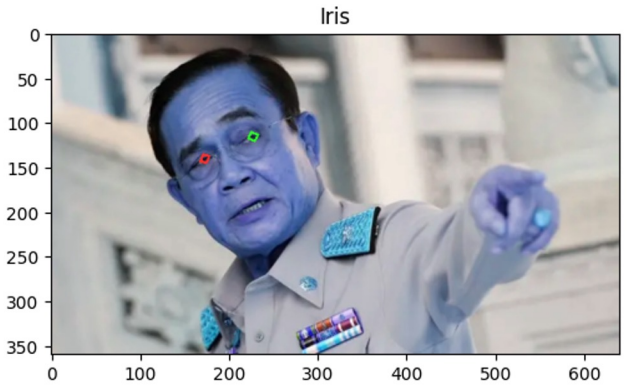
1. **DeepFace:** “Who is this?” (identity matching via embeddings)
2. **MediaPipe:** “Where are the facial features?” (landmarks/mesh for tracking, measurement, expression/motion cues)

MediaPipe (Output)

```
n point: 478
FaceLandmarkerResult(face_landmarks=[[NormalizedLandmark(x=0.5131750702857971, y=0.5360471606254578, z=-0.06546761840581894, visibil
          3.79884839e-01],
        [-6.02275878e-02,  9.98175025e-01, -4.34446009e-03,
          2.14245963e+00],
        [-4.20112014e-02,  1.81368412e-03,  9.99115467e-01,
         -3.23696938e+01],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
          1.00000000e+00]])])
```

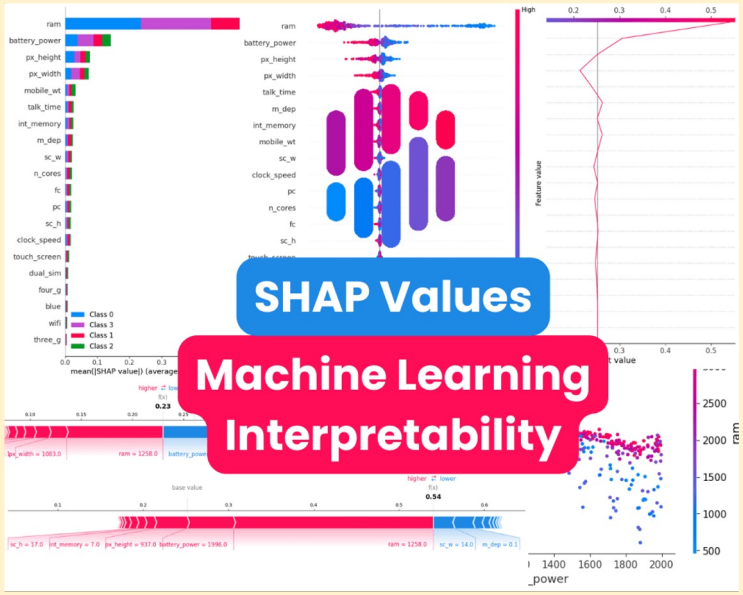






Lab 7.3: ShapValues [URL]

In this lab (Lab_7_3_ShapValues.ipynb), we use SHAP to explain our heart-disease model by showing how each feature contributes to the prediction globally (overall importance) and locally (per patient).



Data

From: UCI Heart Disease Data

Inputs (features): age, sex, chest pain type (cp), resting blood pressure (trestbps), cholesterol (chol), max heart rate (thalch), exercise-induced angina (exang), ST depression (oldpeak), and other clinical indicators.

Label (target): heart disease status from num

- num = 0 → no disease
- num > 0 → disease present (we convert to binary for this lab).

Note: Some fields may contain missing values (e.g., NaN or placeholder values), so we apply imputation during preprocessing.

SHAP Values

SHAP values assign a *contribution score* to each feature for a prediction.
For one patient:

- **Prediction = baseline (average) + sum of SHAP contributions**
- **Positive SHAP** → pushes the model toward **higher heart-disease risk**
- **Negative SHAP** → pushes toward **lower risk**

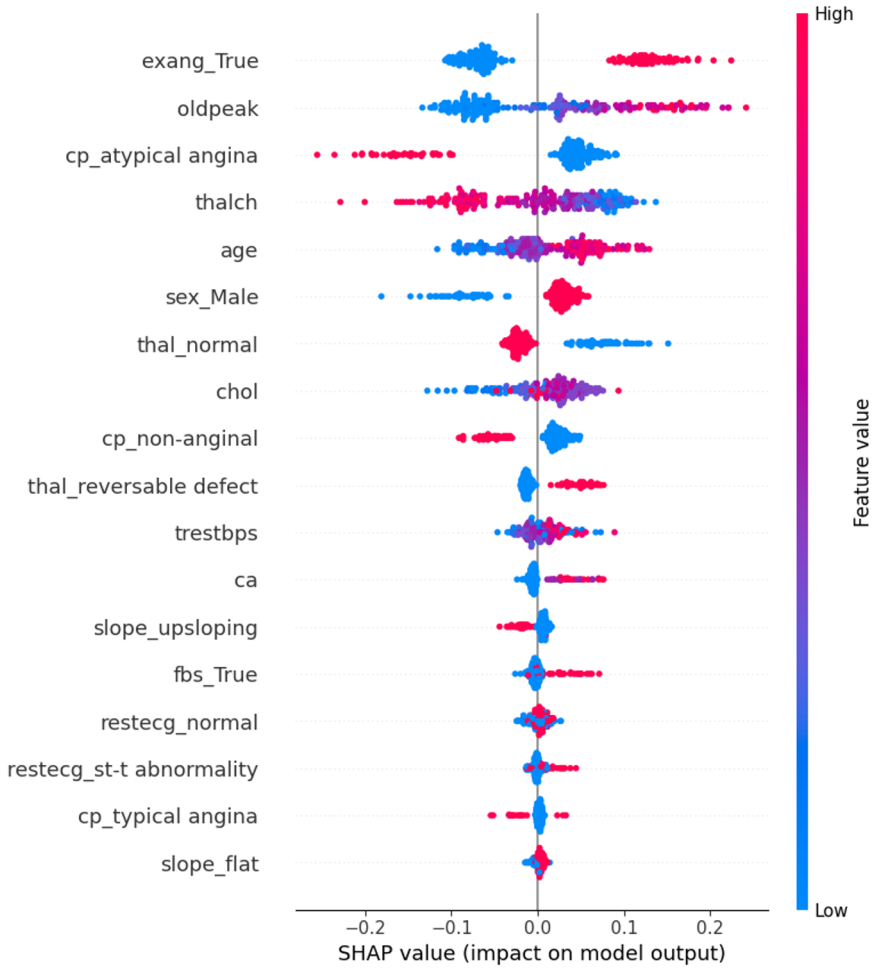
How we use SHAP in this lab (our workflow)

- Train a heart-disease risk model.
- Compute **SHAP values** on the test set to **explain** the trained model:
  - **Global explanation:** summary/bar plots show which clinical factors matter most overall.
  - **Local explanation:** The waterfall plot shows why a specific case is predicted to be high risk or low risk.

Expected Output [Global]

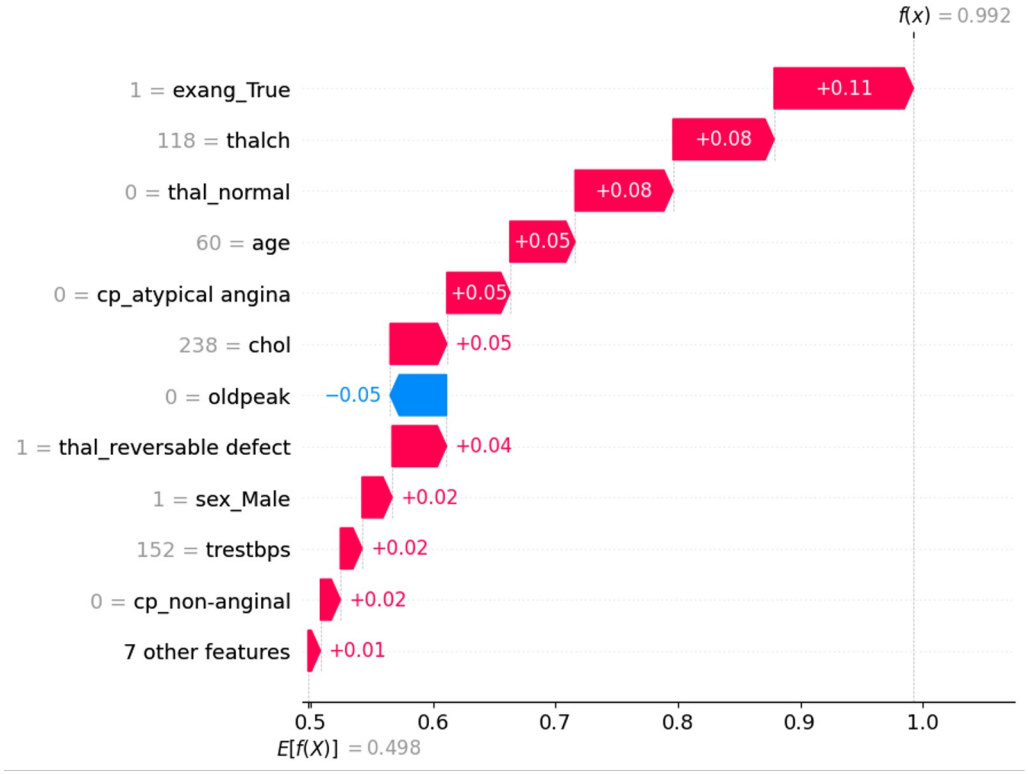
SHAP summary plot: features are ranked by **importance**, and each dot shows whether a patient's feature value pushes the model toward **higher risk (right)** or **lower risk (left)**, with color indicating low vs high feature values.

Ex. exang_True: When this feature is **high (red = 1, exercise-induced angina)**, dots are mostly on the **right**, meaning it **pushes the model toward higher heart-disease risk**.



Expected Output [Local]

Local SHAP waterfall plot: starting from the **baseline prediction**, **each feature** pushes the patient's **risk up (red)** or **down (blue)** until reaching the final predicted risk.

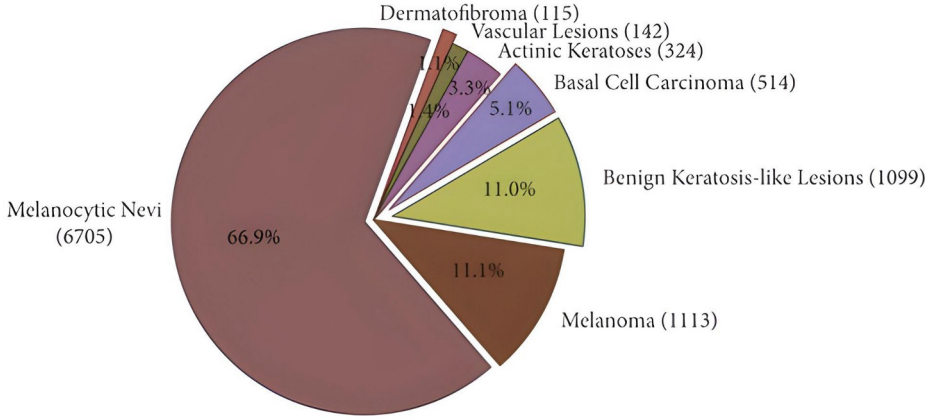


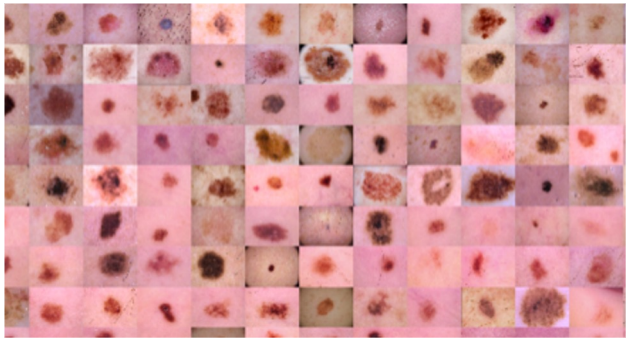
Lab 7.4: Grad-CAM

In this lab (Lab_7_4_GradCAM.ipynb), we demonstrate the use of Grad-CAM to generate heatmaps that highlight the important regions of an image influencing the model's predictions. In this example, we apply Grad-CAM to a skin cancer classification task (Lab_3_2_PyTorch.ipynb).

Dataset: Skin Cancer MNIST: HAM10000

- The dataset consists of 10015 images with 10013 labeled objects belonging to 7 skin cancer classes.
- The data contains image in JPG format and documents in JSON format
- In the experiment, we reduced the amount of data and formatted it to simplify the experiment.





Grad-CAM

The stepwise process to generate CAMs can be shown as follows:

1. Decide for which class and convolutional layer in the neural network the CAM is to be generated.
2. Then, calculate the activations simply by passing the input to the convolutional layer.
3. Fetch the gradient values from the same layer with respect to the class.
4. Calculate the mean of the gradients within each output channel.
5. Calculate the weighted activation map and then compute mean.
6. Upscale the weighted activation map outputs to the same size as the input of the image.
7. Finally, overlay the activation map onto the input image.

Grad-CAM (Output)