TensorFlow (TF): Boundless is published on **TensorFlow Hub**; its APIs (signatures, tensors, dtypes) are TF-native. You could use PyTorch, but you'd need a port—TF is lower friction here.

TensorFlow Hub: stable, versioned **pretrained models** via URLs—no custom weight fetching and signature compatibility out of the box.

PIL: straightforward image IO with *open*, *crop*, *resize*. OpenCV is fine too, but reads **BGR** (needs conversion) and adds heavier dependency for this simple workflow.

NumPy: universal array layer; easy normalization *[0,1]*, reshaping, and TF interop (*tf.constant* from NumPy).

Matplotlib: quick inline visualization to compare inputs/outputs.

Tech: *Image.open* typically returns **RGB**, preserving metadata and avoiding extra conversions.

```
import tensorflow as tf
import tensorflow_hub as hub
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt

image = Image.open('/content/1888-vincent-van-gogh-the-sower.jpg')
```

Why: returning the image displays it inline—initial sanity check (correct file? orientation?).

Tech: notebook front-ends have a repr for *PIL.Image.Image*; no extra copy overhead.

```
image
```

Why: informs cropping/resizing strategy. Vision models often need **fixed input sizes**; Boundless uses 257×257.

Tech: size returns (*width*, *height*); foundation for robust square cropping.

```
image.size
```

```
(900, 718)
```

Why: readability and avoiding W/H swaps.

Tech: explicit variables improve traceability for subsequent crop/mask steps.

```
width, height = image.size
print(width, height)
```

```
900 718
```

Why crop before resize?

- **Avoid distortion**: Resizing a rectangular image directly to 257×257 stretches/squashes objects.
- **Preserve local scale**: Models like Boundless are trained on square inputs; crop → resize keeps natural proportions.
- **Mask consistency**: Outpainting/inpainting pipelines often assume well-behaved known/unknown regions under a 1:1 aspect ratio.

Why use height on both axes?

- It assumes width ≥ height, so a height × height square fits without overflow.
- If height > width, the box would exceed the image; PIL pads out-of-bounds with black — undesirable for inference (introduces artificial borders).

```
image = image.crop((0, 0, height, height))
image
```

- **Resize 257x257**: That's the expected input size for Boundless variants on TF Hub. Other sizes may fail or trigger unwanted internal resampling.
- **Interpolation**: Pillow defaults to BILINEAR for RGB—good tradeoff between smoothness and aliasing for downstream CNNs.
- **Alternatives**: LANCZOS can preserve detail at higher cost; practical difference is minor here.
- **Tech risk**: multiple resizes degrade detail—hence "crop → single resize".

```
image = image.resize((257, 257))
image
```

Sanity check to ensure 257×257 prior to normalization.

```
image.size
```

```
(257, 257)
```

Confirm we're still on PIL.Image.Image prior to array conversion (after resize/crop).

```
type(image)
```

```
PIL.Image.Image
def __init__() -> None
```

```
/usr/local/lib/python3.12/dist-packages/PIL/Image.py
This class represents an image object.  To create
:py:class:`~PIL.Image.Image` objects, use the appropriate factory
functions.  There's hardly ever any reason to call the Image constructor
directly.
```

Why now?: TF takes np.ndarray directly; easier **normalization, batch dim**, and inspection (shape, min/max).

Tech: becomes uint8 [0..255]; we'll explicitly manage dtype/scale next.

```
image = np.array(image)
```

Expected <class 'numpy.ndarray'>. A failure hints IO/corruption issues.

```
type(image)
```

```
numpy.ndarray
```

Checking eyeballing values/shape helps catch wrong conversions (e.g., accidental float scaling).

```
image
```

```
ndarray (257, 257, 3) show data
```

Expected: (257, 257, 3). Channels-last matches TF's default on CPU/GPU, avoiding transposes

```
image.shape
```

```
(257, 257, 3)
```

Why float32 and [0,1]?: Most TF Hub models assume normalized floats; this stabilizes numerics (BatchNorm/Conv expect this scale).

Why add batch dim?: Hub models expect batches: (N, H, W, C). N=1 keeps the default signature happy.

Alternatives: keep uint8 and normalize in-graph, but that mixes preprocessing with inference, hurting debuggability.

```
image = np.expand_dims(image.astype(np.float32) / 255., axis = 0)
```

Expected (1, 257, 257, 3); a mismatch will break model invocation.

```
image.shape
```

```
(1, 257, 257, 3)
```

Useful to confirm values are truly [0,1] (no overflow/underflow).

```
image
```

```
array([[[[0.5411765 , 0.4862745 , 0.09803922],
         [0.47843137, 0.47058824, 0.07843138],
         [0.4627451 , 0.47843137, 0.09411765],
         ...,
         [0.7647059 , 0.5137255 , 0.09411765],
         [0.78039217, 0.49411765, 0.11372549],
         [0.7490196 , 0.5372549 , 0.10588235]],

        [[0.5803922 , 0.52156866, 0.10588235],
         [0.5882353 , 0.57254905, 0.10980392],
         [0.5764706 , 0.56078434, 0.13333334],
         ...,
         [0.75686276, 0.5137255 , 0.09019608],
         [0.7529412 , 0.5647059 , 0.10980392],
         [0.7647059 , 0.7058824 , 0.16078432]],

        [[0.5882353 , 0.5137255 , 0.08235294],
         [0.60784316, 0.60784316, 0.13333334],
         [0.4627451 , 0.4117647 , 0.04313726],
         ...,
         [0.7882353 , 0.64705884, 0.14901961],
         [0.7607843 , 0.6666667 , 0.14509805],
         [0.7254902 , 0.69803923, 0.14117648]],

        ...,

        [[0.27450982, 0.29411766, 0.15294118],
         [0.29411766, 0.33333334, 0.21568628],
         [0.15294118, 0.22352941, 0.1882353 ],
         ...,
         [0.25882354, 0.34117648, 0.3372549 ],
         [0.23921569, 0.3529412 , 0.4392157 ],
         [0.25490198, 0.3882353 , 0.5568628 ]],
```

```
          [[0.47058824, 0.59607846, 0.53333336],
           [0.44705883, 0.58431375, 0.54509807],
           [0.14901961, 0.29803923, 0.32941177],
           ...,
           [0.28627452, 0.41960785, 0.41960785],
           [0.30980393, 0.36862746, 0.44313726],
           [0.34117648, 0.3137255 , 0.32941177]],

          [[0.2509804 , 0.38431373, 0.29803923],
           [0.5058824 , 0.69803923, 0.69411767],
           [0.14509805, 0.30980393, 0.34901962],
           ...,
           [0.32156864, 0.39607844, 0.37254903],
           [0.38039216, 0.30588236, 0.25490198],
           [0.40392157, 0.25490198, 0.10588235]]]], dtype=float32)
```

Model variant selection(@param)

- **What changes?**: The unknown region size to be filled. Quarter hides ~25%, Half ~50%, Three Quarters ~75%, and the model hallucinates content using available context.
- **Why variants?**: Training with different mask ratios yields specialization—better quality for specific missing fractions.
- **Tech implication**: the larger the missing area, the more the model relies on learned priors and global receptive fields.

```
model_name = 'Boundless Half' # @param ['Boundless Half', 'Boundless Qua
```

model_name: Boundless Half

Explicit versioning (/1) ensures reproducibility. Different URLs mean different weights/signatures tuned to the ratio.

```
model_paths = {
    'Boundless Half' : 'https://tfhub.dev/google/boundless/half/1',
    'Boundless Quarter' : 'https://tfhub.dev/google/boundless/quarter/1',
    'Boundless Three Quarters' : 'https://tfhub.dev/google/boundless/three_quarter/1'
}
```

Simplifies downstream: the variable now holds the final URL (avoids double indexing).

```
model_paths = model_paths[model_name]
```

Audit/debug — when saving outputs you'll know precisely which variant/URL generated them.

```
print('Model: ', model_name)
print('Path: ', model_paths)
```

```
Model:  Boundless Half
Path:  https://tfhub.dev/google/boundless/half/1
```

Hub "load": downloads and freezes the exported SavedModel with signatures. More stable than re-implementing the net locally.

Tech: the returned object exposes signatures as callable ConcreteFunctions.

```
model = hub.load(model_paths)
```

- **Why signatures['default']?**: The SavedModel defines a default callable, bundling expected pre/post-processing (like internal masking).
- **Why tf.constant?**: Materializes a non-trainable tensor, perfect for inference; avoids autograd side effects.
- **Outputs**: a dict with keys like default (generated image) and masked_image (input after mask).
- **Tech detail**: calling the signature enforces the exported IO contract (shapes/dtypes).

```
result = model.signatures['default'](tf.constant(image))
```

Models may expose multiple routes (default, serving_default, predict). Knowing names avoids KeyErrors and reveals extra outputs.

```
print(model.signatures.keys())
```

```
KeysView(_SignatureMap({'default': <ConcreteFunction (images: TensorSpec(shape=(None, 257, 257, 3), dtype=tf.float32, name='images')) -> Dict[['masked_
```

Why: confirm names/shapes (e.g., result['default'] is (1,257,257,3) and [0,1]).

Tech: check dtype/range; some models emit [-1,1], here we expect [0,1].

```
result
```

```
          ...,
          [0.76179296, 0.8220856 , 0.559431  ],
          [0.7878084 , 0.8927564 , 0.5278193 ],
          [0.82840097, 0.89768136, 0.51295286]],

         ...,

         [[0.2676859 , 0.2947865 , 0.17290196],
          [0.29547518, 0.3415777 , 0.22107303],
          [0.15351033, 0.23268798, 0.17706037],
          ...,
          [0.65369713, 0.56347173, 0.44815767],
          [0.57992893, 0.47306892, 0.31967443],
          [0.45894012, 0.37245882, 0.2001093 ]],

         [[0.48307428, 0.60116917, 0.506724  ],
          [0.4434549 , 0.5911444 , 0.5363683 ],
          [0.17675072, 0.30140546, 0.30189496],
          ...,
          [0.63672173, 0.54484713, 0.4519775 ],
          [0.60277206, 0.5053545 , 0.35939413],
          [0.488329  , 0.42067096, 0.27820894]],

         [[0.29660308, 0.42837688, 0.36447567],
          [0.4904039 , 0.67252487, 0.65067184],
          [0.17475107, 0.32350457, 0.31742385],
          ...,
          [0.48735395, 0.4141456 , 0.30409807],
          [0.53452885, 0.457762  , 0.2924615 ],
          [0.56852657, 0.50549585, 0.36323482]]], dtype=float32)>}
```

Semantics:

- *masked_image*: input after masking (unknown part hidden).
- *new_image*: the reconstruction/outpainting.

Why two outputs?: To visualize what the network saw vs. what it invented—useful for edge consistency and semantic fidelity evaluation.

```
new_image = result['default']
masked_image = result['masked_image']
```

```
new_image.shape, masked_image.shape
```

```
(TensorShape([1, 257, 257, 3]), TensorShape([1, 257, 257, 3]))
```

Why np.squeeze?: Drops the batch dim (1, H, W, C) → (H, W, C) for Matplotlib.

Why titles and axis('off')?: Emphasizes visual diffs without axis clutter.

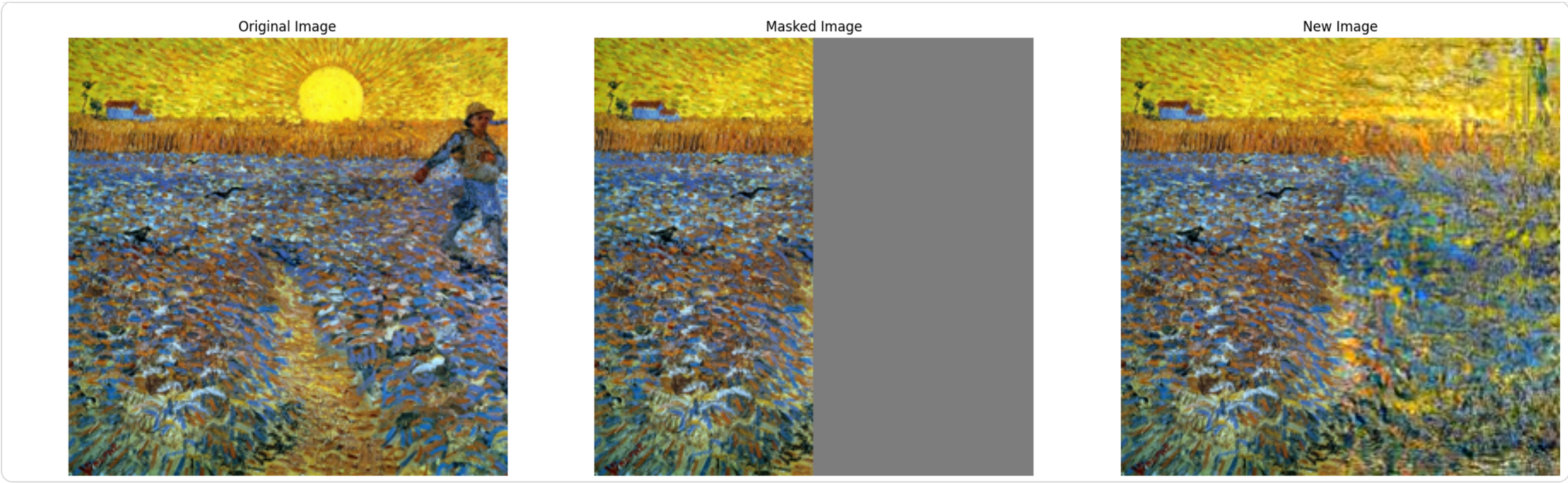
Alternatives: gridspec; imshow with vmin/vmax to guard ranges; here we assume [0,1].

```
def show_images(original_image, masked_image, new_image):
  plt.figure(figsize=(24,12))
  plt.subplot(131)
  plt.imshow((np.squeeze(original_image)))
  plt.title('Original Image')
  plt.axis('off')

  plt.subplot(132)
  plt.imshow((np.squeeze(masked_image)))
  plt.title('Masked Image')
  plt.axis('off')

  plt.subplot(133)
  plt.imshow((np.squeeze(new_image)))
  plt.title('New Image')
  plt.axis('off')

  plt.show()
```

Qualitative inspection—check texture continuity, edge blending, perspective, semantic consistency (e.g., sky/ground/architecture).

Test multiple images and ratios (Quarter/Half/Three Quarters) to feel where the model starts to “hallucinate” too much.

```
show_images(image, masked_image, new_image)
```



This project implemented a complete **outpainting** pipeline to extend images using the **Boundless** variants from **TensorFlow Hub**, covering everything from preprocessing to visualization and qualitative analysis of the results.

What we built

- **Robust preprocessing**: PIL-based reading, EXIF-aware orientation handling, **square crop** (preferably centered) to guarantee a **1:1 aspect ratio**, normalization to **[0,1]**, and addition of the **batch dimension** `(1, H, W, C)`.
- **Model compatibility**: resizing to `257×257` as expected by Boundless variants on TF Hub.
- **Stable model loading**: `hub.load()` with versioned URLs, ensuring reproducibility.
- **Inference via signature**: explicit call to `model.signatures['default'](...)`, honoring the exported I/O contract (shapes and dtypes).
- **Interpretable outputs**: extraction of `masked_image` (what the network actually "saw") and `new_image` (the generated extension), with side-by-side visualization.

Technical decisions (and why)

- **PIL instead of OpenCV**: simpler API for I/O and basic image ops; avoids BGR→RGB conversions.
- **Crop → Resize (instead of warp)**: prevents **distortion**, preserving proportions that models expect.
- **Boundless variants (Quarter/Half/Three Quarters)**: specialization by **size of the missing region**; the larger the area to complete, the stronger the reliance on learned priors and the higher the risk of semantic "hallucination".
- **Matplotlib for inspection**: straightforward **original × masked × generated** comparisons to assess texture continuity, edge blending, and semantic coherence.

Key learnings

- **Aspect ratio matters**: normalizing input geometry (1:1) reduces artifacts and improves mask behavior.
- **SavedModel signatures**: using the `default` signature avoids surprises with pre/post-processing and ensures version compatibility.
- **Fill trade-off**: the more area is missing, the more the model has to invent; results depend on context, texture, and global structure.

Observed limitations

- **Hallucination on large gaps**: with high missing fractions (*Three Quarters*), elements may appear without plausible support in the context.
- **Edges and seams**: transitions may need **post-processing** (e.g., feathering/blending) to become imperceptible.
- **Fixed-size dependency**: unexpected input sizes can break the signature or trigger undesirable resampling.

Best practices adopted

- `ImageOps.exif_transpose` before any crop/resize.
- **Centered square crop** with `s = min(w, h)` to keep the main subject.
- **Single resize** after cropping to minimize detail loss.
- **Sanity checks**: `image.size`, `image.shape`, `[0,1]` range, and dtypes.

Reproducibility

- **Versioning**: TF Hub URLs with explicit version suffixes (`/1`) and explicit logging of `model_name`.
- **Environment**: record versions of `tensorflow`, `tensorflow_hub`, `Pillow`, `numpy`, and `matplotlib`.
- **Determinism**: when relevant, fix seeds and record hardware (CPU/GPU/TPU) and backend.

Next steps (suggestions)

- **Post-processing**: edge feathering, **Poisson blending**, or color adjustments to soften seams.
- **Model comparisons**: evaluate alternative inpainting/outpainting approaches (e.g., LaMa, guided/prompted inpainting) and different mask strategies.
- **Batch inference**: process entire folders and log visual metrics (SSIM/LPIPS) for systematic analysis.
- **Upscaling**: apply *super-resolution* (e.g., ESRGAN) after extension to enhance final detail.

Wrap-up
With a simple, stable, and reproducible pipeline, we **extend images coherently** with the visible context, balancing **local fidelity** (textures/edges) and **global consistency** (structure/semantics). This project establishes a solid base for more advanced experiments in **inpainting/outpainting**, **post-processing** integration, and comparative model evaluation.