In [1]:
```python
import torch
from transformers import pipeline as model
from PIL import Image
import matplotlib.pyplot as plt
import os
```

```
D:\UVG\RESPAI\RESPAI-SHAP-MobileNetV2\.venv\lib\site-packages\tqdm\auto.py:21: TqdmW
arning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywi
dgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```python
pipeline = model(
    task="image-classification",
    model="google/mobilenet_v2_1.4_224",
    dtype=torch.float16,
    device=0 if torch.cuda.is_available() else -1,
)
```

```
Fetching 1 files: 100%|██████████| 1/1 [00:00<00:00, 999.83it/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved w
ith this model. `use_fast=True` will be the default behavior in v4.52, even if the m
odel was saved with a slow processor. This will result in minor differences in outpu
ts. You'll still be able to use a slow processor with `use_fast=False`.
Device set to use cpu
```

In [3]:
```python
# Images from https://github.com/ndb796/Small-ImageNet-Validation-Dataset-1000-Clas
images_path = "data/"
num_files = len(os.listdir(images_path))
images = [
    Image.open(images_path + f"image_{i}.jpg")
    for i in range(1, num_files + 1)
]
```

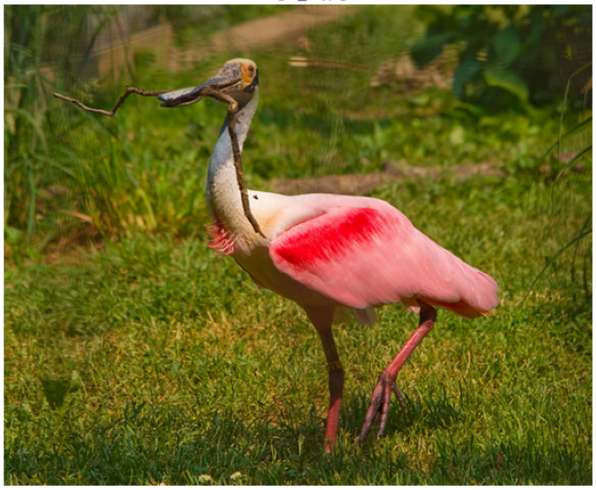
In [4]:
```python
k = 3
```

In [5]:
```python
predictions = pipeline(images, top_k=k)
```

In [6]:
```python
for i, (img, preds) in enumerate(zip(images, predictions), 1):
    # Create a figure with two subplots: one for the image, one for the text
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5), gridspec_kw={'width_ratio

    # Display the image on the left subplot
    ax1.imshow(img)
    ax1.set_title(f"image_{i}.jpg")
    ax1.axis('off')

    # Prepare the prediction text
    pred_text = f"Top {k} Predictions:\n\n"
    # Replaced '\t' with spaces to avoid the UserWarning
    pred_text += "\n".join([f"    {p['label']}: {p['score']:.4f}" for p in preds])

    # Display the predictions as text on the right subplot
    ax2.text(0, 0.5, pred_text, ha='left', va='center', fontsize=12, wrap=True)
    ax2.axis('off')
```

```python
plt.tight_layout()
plt.show()
```

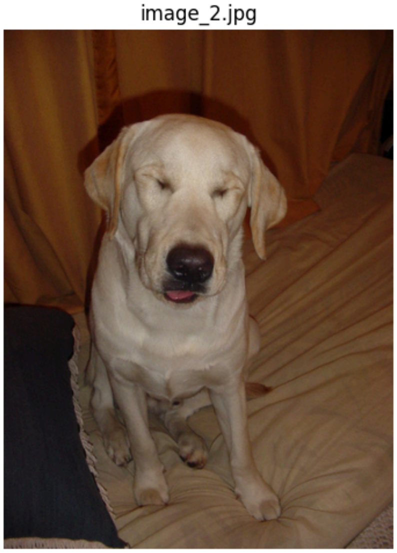
image_1.jpg



Top 3 Predictions:

spoonbill: 0.8641
flamingo: 0.0187
crane: 0.0055

image_2.jpg



Top 3 Predictions:

Labrador retriever: 0.8650
golden retriever: 0.0111
Chesapeake Bay retriever: 0.0078

image_3.jpg

Top 3 Predictions:

bakery, bakeshop, bakehouse: 0.3392
diaper, nappy, napkin: 0.0927
bath towel: 0.0892

image_4.jpg

Top 3 Predictions:

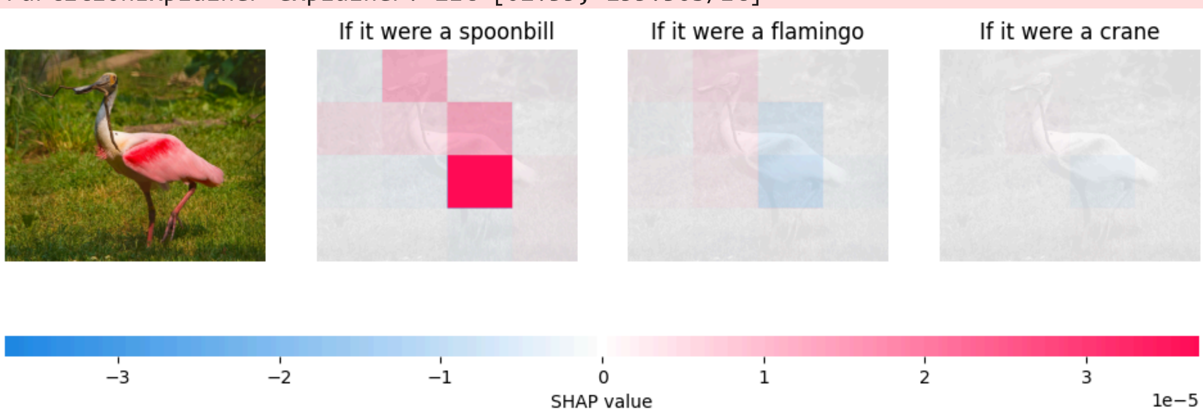
safe: 0.9876
combination lock: 0.0064
sewing machine: 0.0003

In [17]:

```python
import numpy as np
import shap
from PIL import Image

# Select the first image to explain
image_to_explain = images[0]
# Convert the PIL image to a numpy array for the explainer
image_np = np.array(image_to_explain)

# Define a wrapper function for the pipeline to handle data type conversions
def f(x):
    """
    This function converts numpy arrays from SHAP back to PIL images
    for the pipeline and formats the model's output scores into a numpy array.
    """
    # Convert masked numpy arrays back to PIL images
    pil_images = [Image.fromarray(img.astype('uint8')) for img in x]

    # Run predictions through the pipeline
    predictions = pipeline(pil_images, top_k=k)

    # Pre-allocate a numpy array for the scores
    scores = np.zeros((len(predictions), k))
    for i, pred_list in enumerate(predictions):
        for j, pred in enumerate(pred_list):
            scores[i, j] = pred['score']

    return scores

# 1. Create a masker for the image
# The masker generates perturbations of the image to explain predictions
masker = shap.maskers.Image("blur(128,128)", image_np.shape)

# 2. Create an explainer object
# It uses the wrapper function 'f' to get model predictions for masked images
explainer = shap.Explainer(f, masker)

# 3. Calculate SHAP values
```

```
# 'max_evals' is the number of model evaluations to run. Lower for speed, higher fo
# 'batch_size' is the number of masked images to pass to the model at once.
shap_values = explainer(
    image_np[np.newaxis, ...], # Pass the image as a batch of one
    max_evals=100,           # Reduced for faster computation
    batch_size=64             # Increased for potentially better GPU utilization
)

# Add the original image to the SHAP values object for plotting
shap_values.data = image_np[np.newaxis, ...]

# Get the top k predictions for the original image to use as plot titles
original_preds = pipeline([image_to_explain], top_k=k)[0]
class_labels = [pred['label'] for pred in original_preds]
# Format the labels into titles, taking the first part of the label for brevity
formatted_labels = [f"If it were a {label.split(', ')[0]}" for label in class_label

# Visualize the SHAP explanations with custom titles
shap.image_plot(shap_values, labels=formatted_labels)
```

```
  0%|          | 0/98 [00:00<?, ?it/s]

 51%|█████     | 50/98 [00:47<00:45,  1.05it/s]

106it [02:16,  1.34s/it]



PartitionExplainer explainer: 2it [02:35, 155.36s/it]
```



In [ ]:

```
import numpy as np
import shap
from PIL import Image
from skimage.segmentation import slic

# Select the first image to explain
image_to_explain = images[0]
# Convert the PIL image to a numpy array for the explainer
```

```python
image_np = np.array(image_to_explain)

# Define a wrapper function for the pipeline to handle data type conversions
def f(x):
    """
    This function converts numpy arrays from SHAP back to PIL images
    for the pipeline and formats the model's output scores into a numpy array.
    """
    # Convert masked numpy arrays back to PIL images
    pil_images = [Image.fromarray(img.astype('uint8')) for img in x]

    # Run predictions through the pipeline
    predictions = pipeline(pil_images, top_k=k)

    # Pre-allocate a numpy array for the scores
    scores = np.zeros((len(predictions), k))
    for i, pred_list in enumerate(predictions):
        for j, pred in enumerate(pred_list):
            scores[i, j] = pred['score']

    return scores

# 1. Create a masker for the image with more segments for finer detail
# The slic algorithm segments the image into superpixels.
# More segments result in smaller squares.
segments_slic = slic(image_np, n_segments=200, compactness=30, sigma=3, start_label
masker = shap.maskers.Image("blur(128,128)", partition_tree=shap.utils.partition_tr


# 2. Create an explainer object
# It uses the wrapper function 'f' to get model predictions for masked images
explainer = shap.Explainer(f, masker)

# 3. Calculate SHAP values
# 'max_evals' is the number of model evaluations to run. Lower for speed, higher fo
# 'batch_size' is the number of masked images to pass to the model at once.
shap_values = explainer(
    image_np[np.newaxis, ...], # Pass the image as a batch of one
    max_evals=100,            # Reduced for faster computation
    batch_size=64             # Increased for potentially better GPU utilization
)

# Add the original image to the SHAP values object for plotting
shap_values.data = image_np[np.newaxis, ...]

# Get the top k predictions for the original image to use as plot titles
original_preds = pipeline([image_to_explain], top_k=k)[0]
class_labels = [pred['label'] for pred in original_preds]
# Format the labels into titles, taking the first part of the label for brevity
formatted_labels = [f"If it were a {label.split(', ')[0]}" for label in class_label

# Visualize the SHAP explanations with custom titles
shap.image_plot(shap_values, labels=formatted_labels)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[18], line 4
      2 import shap
      3 from PIL import Image
----> 4 from skimage.segmentation import slic
      6 # Select the first image to explain
      7 image_to_explain = images[0]

File D:\UVG\RESPAI\RESPAI-SHAP-MobileNetV2\.venv\lib\site-packages\skimage\segmentat
ion\__init__.py:13
     11 from ._join import join_segmentations, relabel_sequential
     12 from ._watershed import watershed
---> 13 from ._chan_vese import chan_vese
     14 from .morphsnakes import (
     15     morphological_geodesic_active_contour,
     16     morphological_chan_vese,
   (...)
     19     checkerboard_level_set,
     20 )
     21 from ..morphology import flood, flood_fill

File <frozen importlib._bootstrap>:1027, in _find_and_load(name, import_)

File <frozen importlib._bootstrap>:1006, in _find_and_load_unlocked(name, import_)

File <frozen importlib._bootstrap>:688, in _load_unlocked(spec)

File <frozen importlib._bootstrap_external>:879, in exec_module(self, module)

File <frozen importlib._bootstrap_external>:975, in get_code(self, fullname)

File <frozen importlib._bootstrap_external>:1074, in get_data(self, path)

KeyboardInterrupt:
```

In [ ]: