Psychoinformatics - Week 13 (Exercises)

by Jake Luke Harrison (B09207038@ntu.edu.tw)

1 進一步研究CNN (4 points)

1.1 為何ResNet50會判斷小女孩照片為ping-pong_bal, bubble, or Band_Aid? (4 points)

In [ ]:
```
# Original imports
import numpy as np
import urllib.request
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions

# My imports
from PIL import Image, ImageEnhance

!pip install opencv-python
import cv2
from google.colab.patches import cv2_imshow

!pip install retina-face
from retinaface import RetinaFace
import matplotlib.pyplot as plt

!pip install lime
import os
import keras
from keras.applications.imagenet_utils import decode_predictions
from skimage.io import imread
%matplotlib inline

%load_ext autoreload
%autoreload 2
import os,sys
try:
    import lime
except:
    sys.path.append(os.path.join('..', '..')) # add the current directory
    import lime
```

```
from lime import lime_image

from skimage.segmentation import mark_boundaries
```

ResNet50

- ResNet-50 is a convolutional neural network that is 50 layers deep.
- Trained on more than a million images from the ImageNet database.
- Neural network has an image input size of 224-by-224.

Source: https://www.mathworks.com/help/deeplearning/ref/resnet50.html

In [4]:
```
def test(content):
  x = image.img_to_array(content) # convert image to NumPy array
  x = np.expand_dims(x, axis=0) # add batch dimension
  x = preprocess_input(x)
  preds = model.predict(x)
  # decode the results into a list of tuples (class, description, probability)
  # (one such list for each sample in the batch)
  print('Predicted:', decode_predictions(preds, top=3)[0]) # top 3 predictions
```

In [5]:
```
model = ResNet50(weights='imagenet') # weights trained on ImageNet data

urllib.request.urlretrieve('http://mil.psy.ntu.edu.tw/~tren/girl.jpg','girl.jpg')
img = image.load_img('girl.jpg', target_size=(224, 224)) # Or use cv2.resize to resize images

test(img)
```

```
1/1 [==============================] - 1s 1s/step
Predicted: [('n01930112', 'nematode', 0.11428822), ('n03041632', 'cleaver', 0.051235814), ('n03838899', 'oboe', 0.044624917)]
```

The cell above sometimes outputs 'nematode', 'cleaver', and 'oboe' as its predictions and other times predicts ping-pong ball, bubble, and band-aid. I have generated both outputs multiple times. In line with the question, I will primarily respond to the output of ping-pong ball, bubble, and band-aid.

In [ ]:
```
# Inspect image (pre-conversion)
img
```

Out[ ]:

I first observe that a leaf is obscuring the girl's nose. I hypothesise that this leaf may interfere with the correct classification of this image as a human face. To test this hypothesis, I input an image of an unobstructed face.

In [ ]:

```
# im1
# Testing an unobstructed face

im1 = image.load_img('/content/face.jpg', target_size=(224, 224))
im1
```

Out[ ]:

In [ ]:

```
test(im1)
```

```
1/1 [==============================] - 0s 29ms/step
Predicted: [('n04584207', 'wig', 0.41434947), ('n04599235', 'wool', 0.06030627), ('n02892767', 'brassiere', 0.05132565)]
```

Predictions for this image are also inaccurate. "Brassiere" is a seemingly bizarre prediction, like "Band_Aid."

If the model had accurately classified `im1` as a human face, I may have tentatively inferred that failure to accurately classify `img` is related to the leaf that obscures the nose.

Next, I hypothesise that the model is dysfunctional. For example, I may have incorrectly initialised the model. To test this hypothesis, I input an image of something other than a human face.

In [ ]:
```
# im2
# Testing an image of a house

im2 = image.load_img('/content/house.jpg', target_size=(224, 224))
im2
```

Out[ ]:

In [ ]:
```
test(im2)
```

```
1/1 [==============================] - 0s 23ms/step
Predicted: [('n03930313', 'picket_fence', 0.25259662), ('n02859443', 'boathouse', 0.2181337), ('n03899768', 'patio', 0.0918169
2)]
```

Predictions for `im2` seem to be more accurate. "Picket_fence" seems unusual, but it is not as outlandish as "brassiere" and "band_aid", and it does contextually relate to the exterior of a property.

It appears the model is not dysfunctional. However, I should perform a few more tests to support this conclusion. I will test the model with a football and a dog.

In [ ]:
```
# im3
# Testing an image of a football

im3 = image.load_img('/content/football.jpeg', target_size=(224, 224))
im3
```

Out[ ]:

In [ ]:
```
test(im3)
```

```
1/1 [==============================] - 0s 24ms/step
Predicted: [('n04254680', 'soccer_ball', 0.96205246), ('n03445777', 'golf_ball', 0.03675554), ('n03134739', 'croquet_ball', 0.0
0044769532)]
```

Predictions for `im3` also seem to be more accurate. There are no seemingly unusual or outlandish predictions.

In [ ]:
```
# im4
# Testing an image of a dog

im4 = image.load_img('/content/dog.webp', target_size=(224, 224))
im4
```

Out[ ]:

In [ ]:
```
test(im4)
```

```
1/1 [==============================] - 0s 26ms/step
Predicted: [('n02089973', 'English_foxhound', 0.46194214), ('n02088364', 'beagle', 0.35312226), ('n02089867', 'Walker_hound',
0.18015952)]
```

Predictions for this `im4` are seemingly accurate. The model specifies three breeds of dog, and the dog in this image resembles all three of these breeds.

I infer from these simple tests that the model does not invariably misclassify stimuli.

The model may specifically struggle with facial classification. However, I am not sure *why* the model may struggle with facial classification. To explore this, I iteratively manipulate multiple variables: colour, orientation, contrast, and sharpness.

In [13]:
```
# im5
# Removing colour

grey = Image.open('girl.jpg').convert('L')
grey.save('greyscale.png')
im5 = image.load_img('/content/greyscale.png', target_size=(224, 224))
im5
```

Out[13]:

In [ ]:
```
test(im5)
```

```
1/1 [==============================] - 0s 23ms/step
Predicted: [('n03481172', 'hammer', 0.12620373), ('n02786058', 'Band_Aid', 0.079242796), ('n03642806', 'laptop', 0.05972784)]
```

Greyscale conversion does not facilitate accurate classification. Predictions—"hammer", "Band_Aid", "laptop"—are still seemingly outlandish

In [ ]:
```
# im6
# Manipulating orientation

im = Image.open('girl.jpg').resize((224, 224))
```

```
im6 = im.rotate(180)
im6
```

Out[ ]:

In [ ]:
```
test(im6)
```

```
1/1 [==============================] - 0s 22ms/step
Predicted: [('n03476991', 'hair_spray', 0.6359716), ('n04584207', 'wig', 0.08727726), ('n02786058', 'Band_Aid', 0.08534749)]
```

Rotation does not facilitate accurate classification.

In [ ]:
```
# im7
# Manipulating contrast

im7 = ImageEnhance.Contrast(im).enhance(3)
im7
```

Out[ ]:

In [ ]:
```
test(im7)
```

```
1/1 [==============================] - 0s 23ms/step
Predicted: [('n03916031', 'perfume', 0.15124546), ('n03929660', 'pick', 0.11777672), ('n04370456', 'sweatshirt', 0.07496039)]
```

Increasing contrast does not facilitate more accurate predictions.

In [ ]:
```
# im8
# Manipulating sharpness

im8 = ImageEnhance.Sharpness(im).enhance(5)
im8
```

Out[ ]:

In [ ]:
```
test(im8)
```

```
1/1 [==============================] - 0s 22ms/step
Predicted: [('n03942813', 'ping-pong_ball', 0.22120278), ('n02786058', 'Band_Aid', 0.21622373), ('n03929660', 'pick', 0.0559285
47)]
```

Sharpening the image does not facilitate more accurate predictions. I once again observe 'ping-pong_ball' and 'Band_Aid'.

Next, I hypothesise that facial recognition may somehow be particularly challenging for **ResNet50**. It is possible that other models excel where **ResNet50** fails. In the following cell, I try the same classification task with five alternative models: **InceptionV3**, **Xception**, **VGG16**, **VGG19**, and **MobileNetV2**.

In [ ]:
```
# Trying different models

im9 = image.load_img('girl.jpg', target_size=(299, 299))

from keras.applications.inception_v3 import InceptionV3, preprocess_input, decode_predictions
model = InceptionV3(weights='imagenet')
print('InceptionV3:')
test(im9)
```

```python
from keras.applications.xception import Xception, preprocess_input, decode_predictions
model = Xception(weights='imagenet')
print('\nXception:')
test(im9)

from keras.applications.vgg16 import VGG16, preprocess_input, decode_predictions
model = VGG16(weights='imagenet')
print('\nVGG16')
test(img)

from keras.applications.vgg19 import VGG19, preprocess_input, decode_predictions
model = VGG19(weights='imagenet')
print('\nVGG19')
test(img)

from keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input, decode_predictions
model = MobileNetV2(weights='imagenet')
print('\nMobileNetV2')
test(img)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96112376/96112376 [==============================] - 0s 0us/step
InceptionV3:
1/1 [==============================] - 3s 3s/step
Predicted: [('n04409515', 'tennis_ball', 0.8837042), ('n07714990', 'broccoli', 0.012199741), ('n03929660', 'pick', 0.007957127)]
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels.h5
91884032/91884032 [==============================] - 1s 0us/step

Xception:
1/1 [==============================] - 2s 2s/step
Predicted: [('n02786058', 'Band_Aid', 0.4089417), ('n07714990', 'broccoli', 0.059544664), ('n04263257', 'soup_bowl', 0.027249973)]
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 11s 0us/step

VGG16
1/1 [==============================] - 1s 836ms/step
Predicted: [('n03788365', 'mosquito_net', 0.110914886), ('n15075141', 'toilet_tissue', 0.033581957), ('n04209239', 'shower_curtain', 0.02503936)]
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels.h5
574710816/574710816 [==============================] - 6s 0us/step
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7b9654b8db
40> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @t
f.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://ww
w.tensorflow.org/api_docs/python/tf/function for  more details.
VGG19
1/1 [==============================] - 0s 167ms/step
Predicted: [('n03788365', 'mosquito_net', 0.14649728), ('n04209239', 'shower_curtain', 0.036219217), ('n02804414', 'bassinet',
0.029251767)]
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ord
ering_tf_kernels_1.0_224.h5
14536120/14536120 [==============================] - 0s 0us/step

MobileNetV2
WARNING:tensorflow:6 out of the last 10 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7b9630374
310> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @t
f.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://ww
w.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 2s 2s/step
Predicted: [('n07742313', 'Granny_Smith', 0.4359165), ('n07615774', 'ice_lolly', 0.2582828), ('n09229709', 'bubble', 0.04982236
4)]
```

Alternative models are also unable to accurately classify the image.

Based on this, I infer that there is some commonality—or commonalities—between these models that prevents them from accurately classifying human faces.

Inspecting the documentation, I identify the **ImageNet** dataset as a commonality.

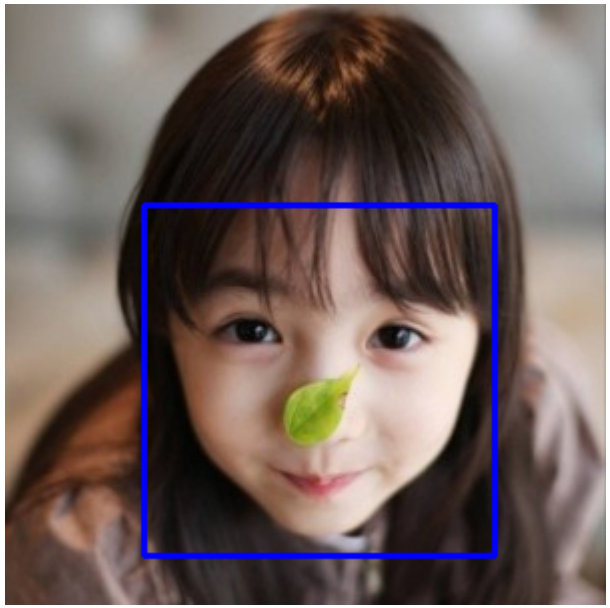
The labels in the ImageNet dataset can be inspected here: https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

It seems there is not a label for "human", "face", "eye", or "mouth".

Therefore, the model may be outputting available labels that most closely match the `girl.jpg` in the absence of labels such as "human face", "human head", "human", "face", and so on. Ping pong balls and bubbles are spherical, or circular, and in this feature they resemble `girl.jpg`.

In [ ]:
```python
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
img = cv2.imread(r'/content/girl.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x, y, w, h) in faces:
```

```
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
cv2_imshow(img)
```



In [ ]:

```
# https://github.com/serengil/retinaface
faces = RetinaFace.extract_faces(img_path = "girl.jpg", align = True)
for face in faces:
  plt.imshow(face)
  plt.show()
```

```
Directory  /root /.deepface created
Directory  /root /.deepface/weights created
retinaface.h5 will be downloaded from the url https://github.com/serengil/deepface_models/releases/download/v1.0/retinaface.h5
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/retinaface.h5
To: /root/.deepface/weights/retinaface.h5
100%|██████████| 119M/119M [00:00<00:00, 329MB/s]
```

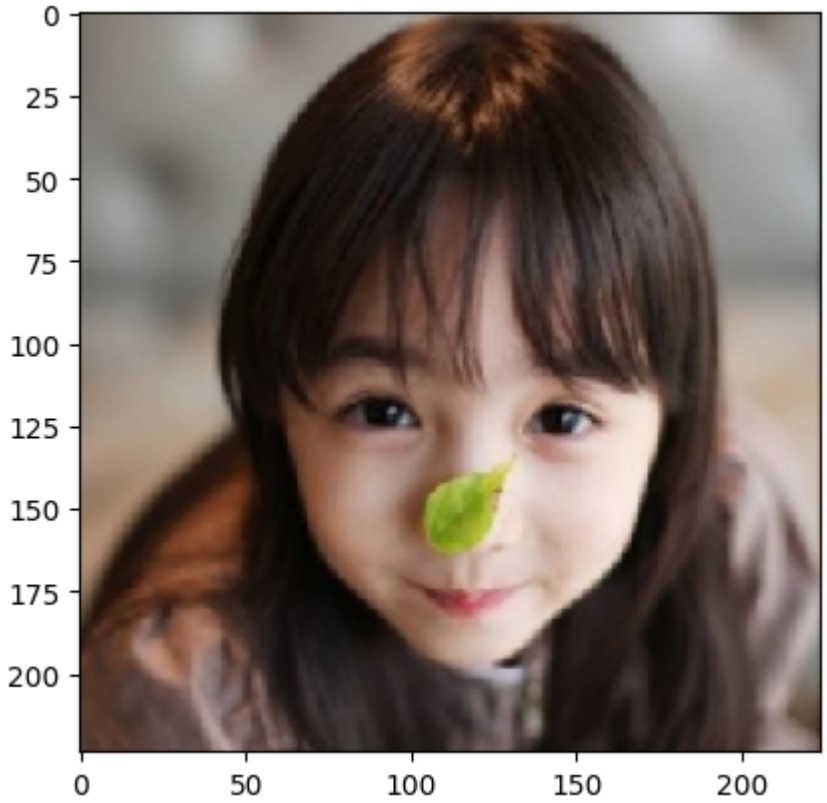
Interestingly, face detection models do not seem to identify the spherical shape of the head—extending beyond the upper border—as a key feature of the face itself.

In the following cells, I try to use **lime** to learn a bit more about how the model is classifying the image.

In [14]:
```
model = ResNet50(weights='imagenet')

def transform_img_fn(path_list):
    out = []
    for img_path in path_list:
        img = image.load_img(img_path, target_size=(224, 224))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)
        out.append(x)
    return np.vstack(out)

images = transform_img_fn([os.path.join('girl.jpg')])
plt.imshow(images[0] / 2 + 0.5)
preds = model.predict(images)
```

```
for x in decode_predictions(preds)[0]:
    print(x)
```

```
1/1 [==============================] - 1s 1s/step
('n01930112', 'nematode', 0.11428822)
('n03041632', 'cleaver', 0.051235814)
('n03838899', 'oboe', 0.044624917)
('n02667093', 'abaya', 0.026966078)
('n02783161', 'ballpoint', 0.025230963)
```

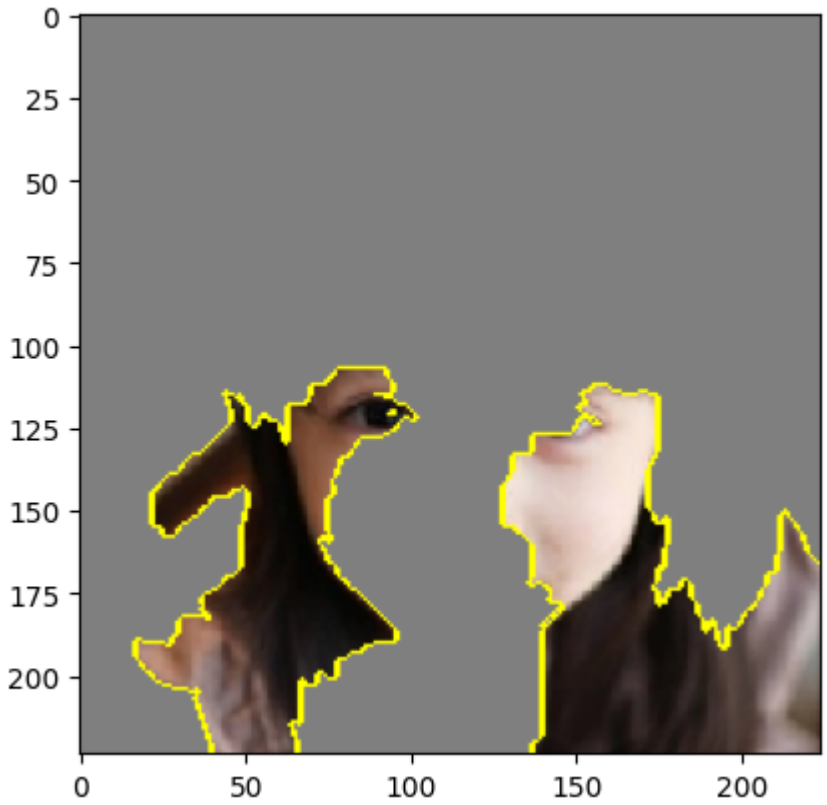


In [ ]:
```
explainer = lime_image.LimeImageExplainer()

explanation = explainer.explain_instance(
    images[0].astype('double'),
    model.predict,
    top_labels=5,
    hide_color=0,
    num_samples=1000
    )
```

In [16]:
```
temp, mask = explanation.get_image_and_mask(
    explanation.top_labels[0],
```

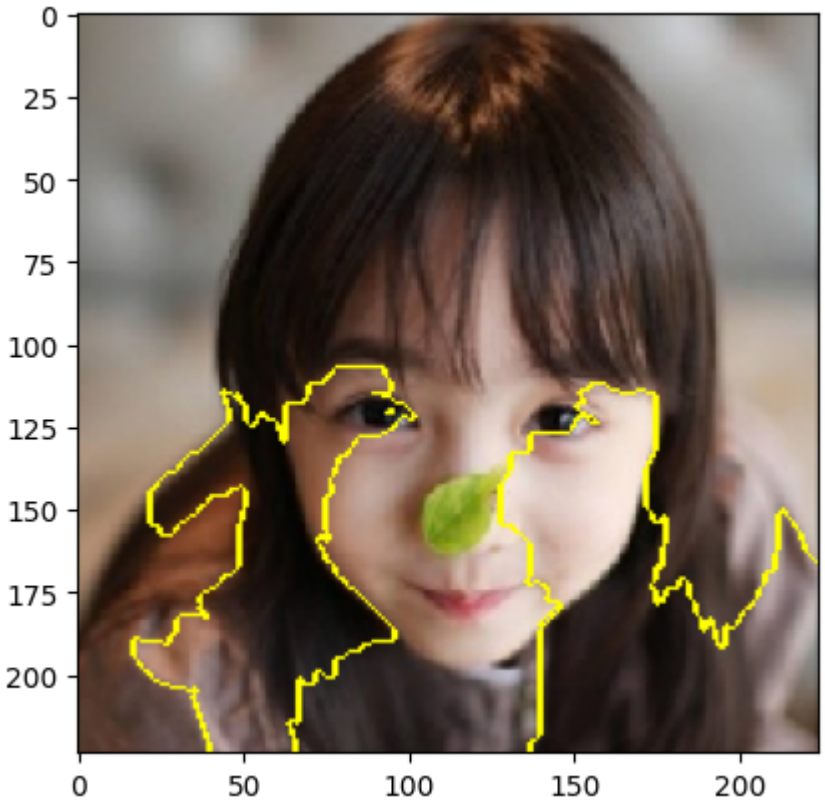
```
    positive_only=True,
    num_features=5,
    hide_rest=True
    )

plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

Out[16]:
```
<matplotlib.image.AxesImage at 0x7ba75e24ddb0>
```



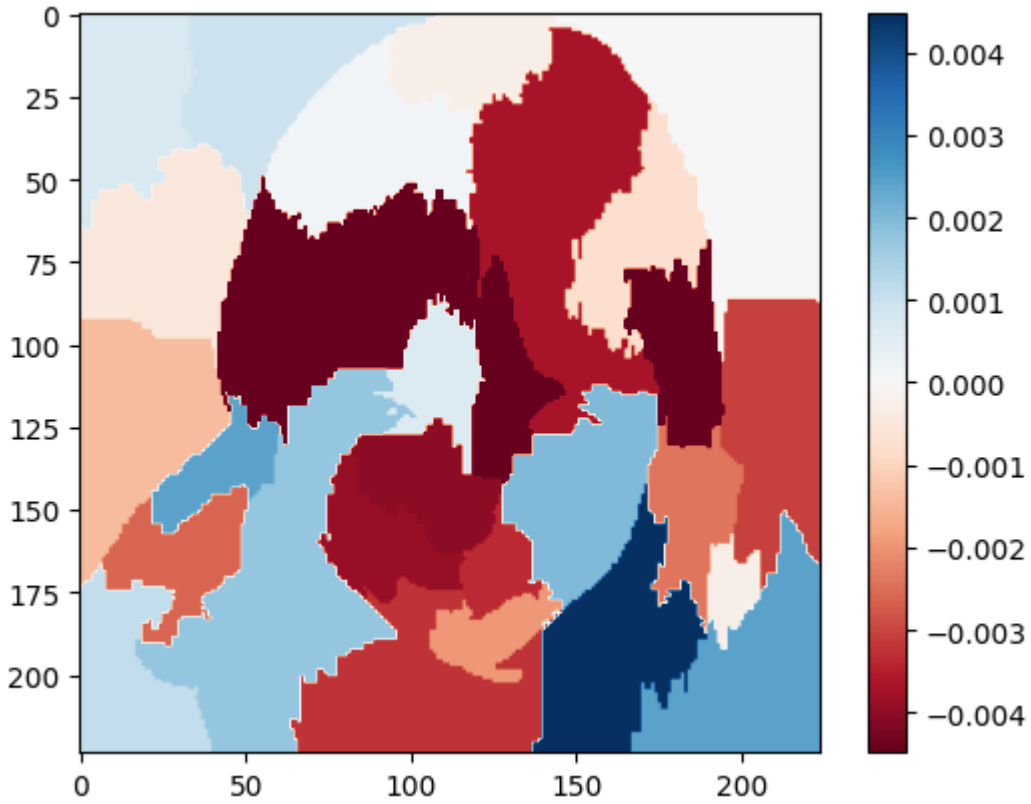
In [17]:
```
temp, mask = explanation.get_image_and_mask(
    explanation.top_labels[0],
    positive_only=True,
    num_features=5,
    hide_rest=False)

plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x7ba75e4c1cf0>
```



In [18]:
```
ind = explanation.top_labels[0]
dict_heatmap = dict(explanation.local_exp[ind])
heatmap = np.vectorize(dict_heatmap.get)(explanation.segments)
plt.imshow(heatmap, cmap = 'RdBu', vmin  = -heatmap.max(), vmax = heatmap.max())
plt.colorbar()
```

Out[18]:
```
<matplotlib.colorbar.Colorbar at 0x7ba75e10f640>
```



1.2 請展示有別人pre-trained好的Keras model可以成功辨認girl.jpg為人臉 (4 points)

I referenced the following page.

https://www.sitepoint.com/keras-face-detection-recognition/

In [ ]:
```
!pip install mtcnn
```

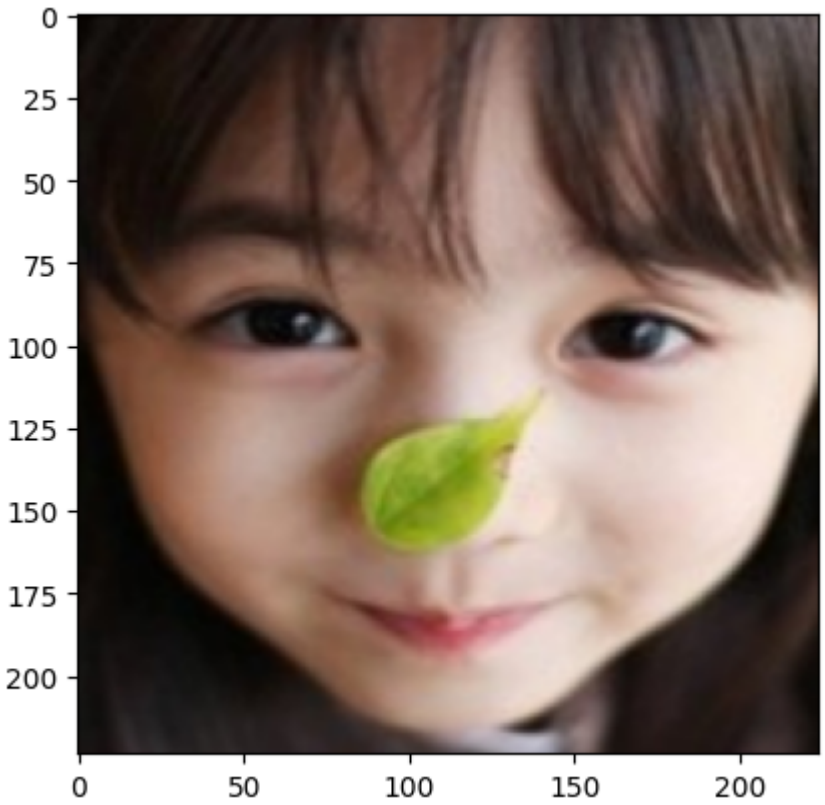
In [48]:
```
import keras
from matplotlib import pyplot as plt
from matplotlib.patches import Rectangle
from mtcnn.mtcnn import MTCNN
from numpy import asarray
```

The following cell contains a slightly modified version of a function from the aforementioned page.

In [49]:
```
def extract_face_from_image(image_path, required_size=(224, 224)):
  image = plt.imread(image_path)
```

```
  detector = MTCNN()
  faces = detector.detect_faces(image)
  face_images = []
  for face in faces:
    x1, y1, width, height = face['box']
    x2, y2 = x1 + width, y1 + height
    face_boundary = image[y1:y2, x1:x2]
    face_image = Image.fromarray(face_boundary)
    face_image = face_image.resize(required_size)
    face_array = asarray(face_image)
    face_images.append(face_array)
  return face_images

extracted_face = extract_face_from_image('girl.jpg')

plt.imshow(extracted_face[0])
plt.show()
```

```
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 37ms/step
3/3 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 476ms/step
```



This function successfully detects and outputs the human face.

In [35]:
```
type(extracted_face)
```

Out[35]:
```
list
```

The function returns a list. To check whether the function has detected a face, I can use `len()` and input this list. In other words, I can check whether or not the function returns an empty list. If the list is empty, it is because the function has not detected a face.

In [40]:
```
extracted_face_ball = extract_face_from_image('/content/football.jpeg')
```

```
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 311ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 79ms/step
1/1 [==============================] - 1s 538ms/step
```

In [41]:
```
len(extracted_face_ball)
```

Out[41]:
```
0
```

If I input an image of a football, the function does not detect a face, and therefore the list that it outputs is empty.

In [44]:
```
extracted_face_house = extract_face_from_image('/content/house.jpeg')
```

```
1/1 [==============================] - 0s 323ms/step
1/1 [==============================] - 0s 387ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 197ms/step
```

In [45]:
```
len(extracted_face_house)
```

Out[45]:
```
0
```

Similarly, if I input an image of a house, the function does not detect a face, and therefore `len(extracted_face_house)` outputs 0.

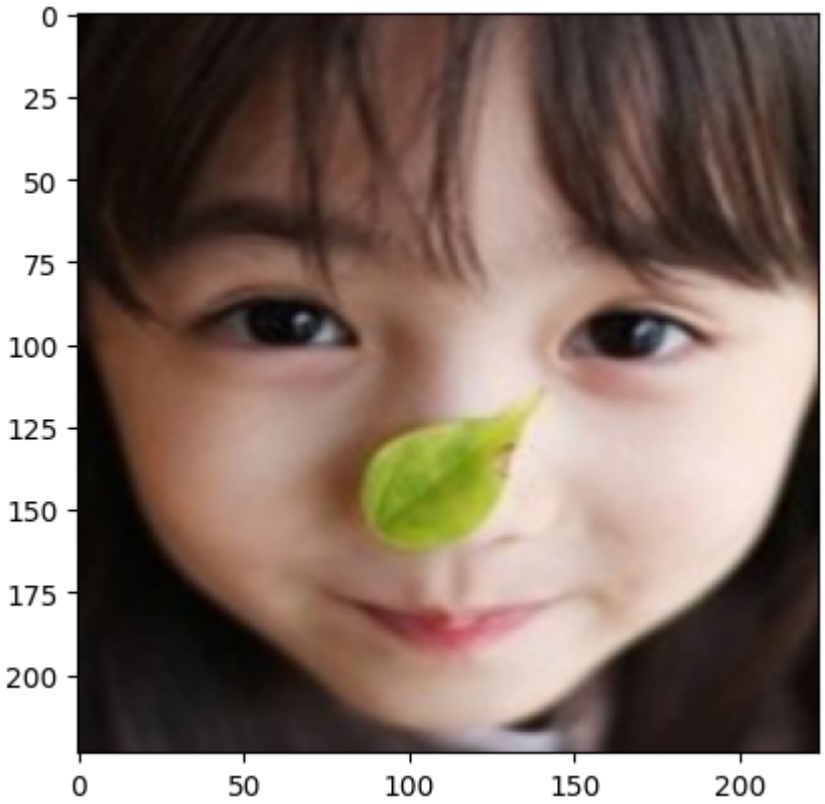
In the next cell, I further modify the function to apply the **InceptionV3** model if a face is not detected.

In [89]:
```
from keras.applications.inception_v3 import InceptionV3, preprocess_input, decode_predictions

def detect_face(image_path, required_size=(224, 224)):
  im = plt.imread(image_path)
  detector = MTCNN()
  faces = detector.detect_faces(im)
  face_images = []
  for face in faces:
    x1, y1, width, height = face['box']
    x2, y2 = x1 + width, y1 + height
```

```
    face_boundary = im[y1:y2, x1:x2]
    face_image = Image.fromarray(face_boundary)
    face_image = face_image.resize(required_size)
    face_array = asarray(face_image)
    face_images.append(face_array)
  if len(face_images) > 0:
    print('\nface\n')
    plt.imshow(face_images[0])
    plt.show
  else:
    model = InceptionV3(weights='imagenet')
    img = image.load_img(image_path, target_size=(299, 299))
    print()
    return test(img)
```

In [79]:
```
detect_face('girl.jpg')
```

```
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
3/3 [==============================] - 0s 11ms/step
1/1 [==============================] - 0s 156ms/step

face
```



If I apply this function to `girl.jpg`, it identifies and displays the face.

In [90]:
```
detect_face('football.jpeg')
```

```
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 221ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 145ms/step

1/1 [==============================] - 0s 250ms/step
Predicted: [('n04254680', 'soccer_ball', 0.9999596), ('n04540053', 'volleyball', 4.536449e-06), ('n02319095', 'sea_urchin', 1.4
046591e-06)]
```

If I apply this function to `football.jpeg`, it successfully classifies the image as "soccer_ball".