fyp-1

December 10, 2023

[1]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[3]:
```
batch_size = 16
img_height = 224
img_width = 224
data_dir = '/content/drive/MyDrive/dataset'
```

[4]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)


val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

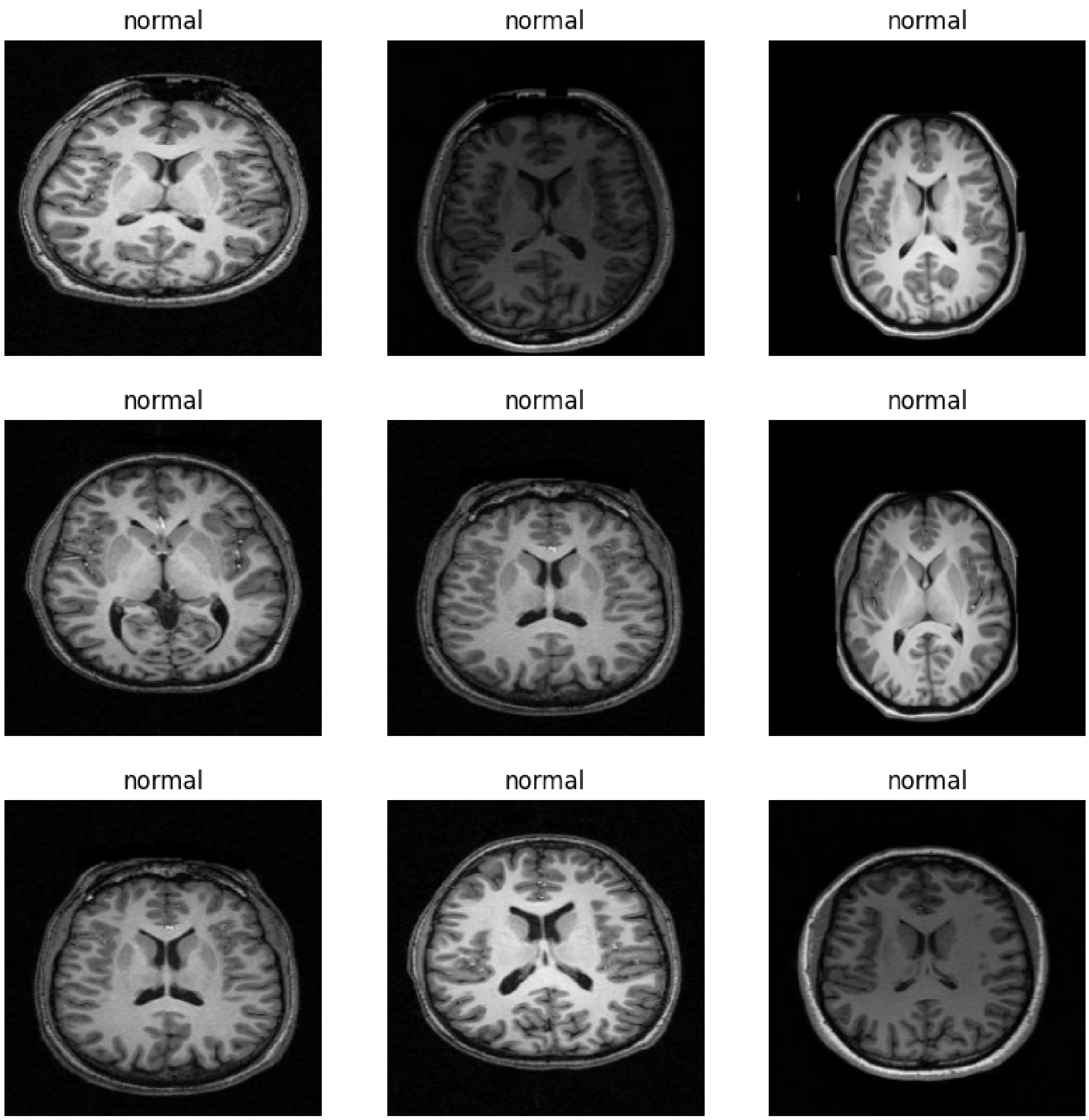
```
Found 137 files belonging to 2 classes.
```

```
Using 110 files for training.
Found 137 files belonging to 2 classes.
Using 27 files for validation.
```

[5]:
```
class_names = train_ds.class_names
print(class_names)
```

```
['normal', 'ventriculomegaly']
```

[6]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



[7]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

[8]:
```
normalization_layer = layers.Rescaling(1./255)
```

[9]:
```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
```

TRAINING AND TESTING

[10]:

```python
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

[11]:

```python
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[12]:

```python
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 224, 224, 16)      448

 max_pooling2d (MaxPooling2  (None, 112, 112, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 112, 112, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 64)        0
 g2D)

 dropout (Dropout)           (None, 28, 28, 64)        0

 flatten (Flatten)           (None, 50176)             0
```

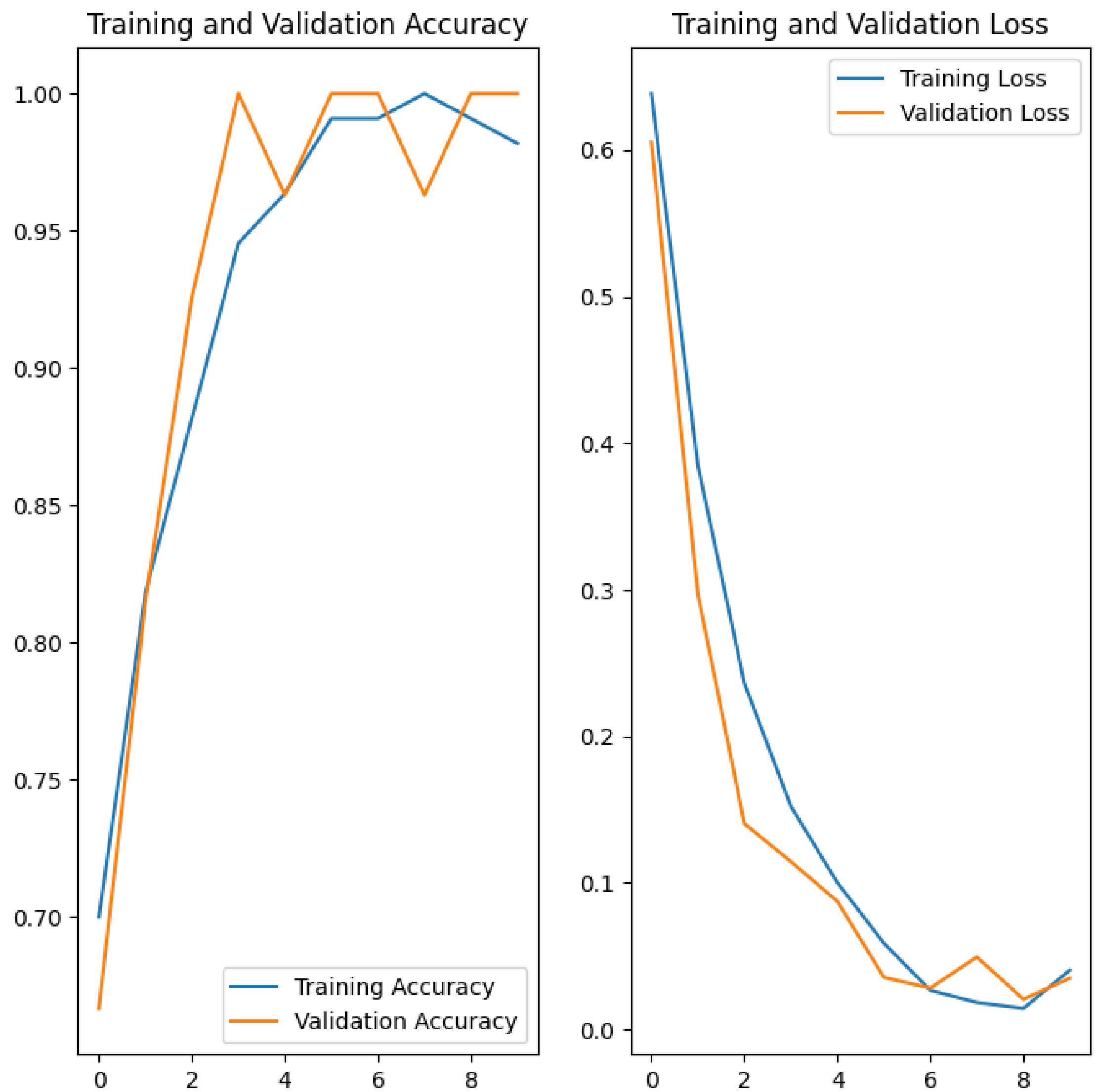
```
 dense (Dense)               (None, 128)               6422656

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 6446498 (24.59 MB)
Trainable params: 6446498 (24.59 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[13]:
```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
7/7 [==============================] - 10s 1s/step - loss: 0.6385 - accuracy:
0.7000 - val_loss: 0.6052 - val_accuracy: 0.6667
Epoch 2/10
7/7 [==============================] - 6s 889ms/step - loss: 0.3850 - accuracy:
0.8182 - val_loss: 0.2971 - val_accuracy: 0.8148
Epoch 3/10
7/7 [==============================] - 6s 921ms/step - loss: 0.2363 - accuracy:
0.8818 - val_loss: 0.1405 - val_accuracy: 0.9259
Epoch 4/10
7/7 [==============================] - 8s 1s/step - loss: 0.1523 - accuracy:
0.9455 - val_loss: 0.1146 - val_accuracy: 1.0000
Epoch 5/10
7/7 [==============================] - 6s 791ms/step - loss: 0.1002 - accuracy:
0.9636 - val_loss: 0.0875 - val_accuracy: 0.9630
Epoch 6/10
7/7 [==============================] - 8s 1s/step - loss: 0.0589 - accuracy:
0.9909 - val_loss: 0.0355 - val_accuracy: 1.0000
Epoch 7/10
7/7 [==============================] - 6s 955ms/step - loss: 0.0267 - accuracy:
0.9909 - val_loss: 0.0282 - val_accuracy: 1.0000
Epoch 8/10
7/7 [==============================] - 6s 838ms/step - loss: 0.0184 - accuracy:
1.0000 - val_loss: 0.0495 - val_accuracy: 0.9630
Epoch 9/10
7/7 [==============================] - 7s 1s/step - loss: 0.0144 - accuracy:
0.9909 - val_loss: 0.0206 - val_accuracy: 1.0000
Epoch 10/10
7/7 [==============================] - 8s 1s/step - loss: 0.0405 - accuracy:
0.9818 - val_loss: 0.0350 - val_accuracy: 1.0000
```

[14]:

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



[15]:

```
model.save('fetal_img_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

[16]:

```
new_model = tf.keras.models.load_model('fetal_img_model.h5') #Load Model
```

[17]:

```
def predict_new_images(img_paths,model):

  final_preds = []
```

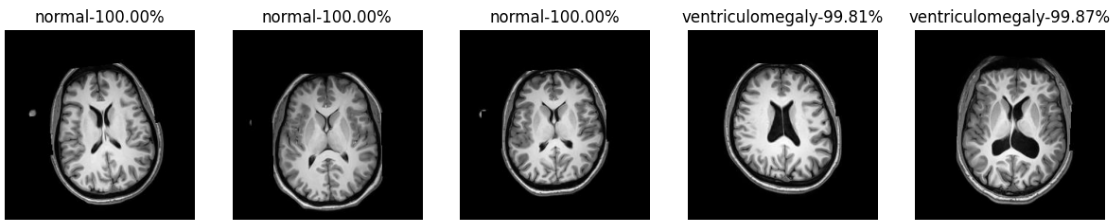
```
  fig = plt.figure(figsize = (15, 15))
  cnt = 0

  for i in img_paths:

    ground_value = i.split('/')[5]

    img = tf.keras.utils.load_img(
      i, target_size=(img_height, img_width)
    )

    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = new_model.predict(img_array,verbose=0)
    score = tf.nn.softmax(predictions[0])

    print(
        "Actual - {} | Predicted - {} with a {:.2f} percent confidence."
        .format(ground_value,class_names[np.argmax(score)], 100 * np.max(score))
    )

    pred_title = "{}-{:.2f}%".format(class_names[np.argmax(score)], 100 * np.
↪max(score))

    ax = fig.add_subplot(1, len(img_paths), cnt+1)
    plt.imshow((plt.imread(i)), cmap='gray')
    plt.axis('off')
    plt.title(pred_title)
    cnt+=1

    tup = (f'{class_names[np.argmax(score)]}-{(100 * np.max(score)):.2f} %',␣
↪img_array)
    final_preds.append(tup)


  return final_preds
```

[28]:

```
#Add Image Paths into the array for prediction

images_to_predict = [

    # Fetal Abdomen - 3 (num_feat)

    # '/content/drive/MyDrive/FYP-XAI/FETAL_PLANES_ZENODO/Fetal-abdomen/
↪Patient01789_Plane2_2_of_2.png',
```

```python
    # '/content/drive/MyDrive/FYP-XAI/FETAL_PLANES_ZENODO/Fetal-abdomen/
↪Patient00960_Plane2_2_of_4.png',
    '/content/drive/MyDrive/img/image.0087.png',


    # Fetal Thorax - 8/5 (num_feat)

    #'/content/drive/MyDrive/FYP-XAI/FETAL_PLANES_ZENODO/Fetal-thorax/
↪Patient00811_Plane6_1_of_3.png',
    '/content/drive/MyDrive/img/image.0085.png',


    # Fetal Femur - 3 (num_feat)

    #'/content/drive/MyDrive/FYP-XAI/FETAL_PLANES_ZENODO/Fetal-femur/
↪Patient00168_Plane5_1_of_2.png',
    '/content/drive/MyDrive/img/image.0083.png',

    #Maternal Cervix - 8 (num_feat)

    '/content/drive/MyDrive/img/image.0091.png',
    #'/content/drive/MyDrive/FYP-XAI/FETAL_PLANES_ZENODO/Maternal-cervix/
↪Patient00239_Plane4_1_of_1.png',


    #Other

    '/content/drive/MyDrive/img/image.0079.png',
    ##'/content/drive/MyDrive/FYP-XAI/FETAL_PLANES_ZENODO/Other/
↪Patient00002_Plane1_12_of_20.png',

]

final_preds = predict_new_images(images_to_predict,new_model)
```

```
Actual - image.0087.png | Predicted - normal with a 100.00 percent confidence.
Actual - image.0085.png | Predicted - normal with a 100.00 percent confidence.
Actual - image.0083.png | Predicted - normal with a 100.00 percent confidence.
Actual - image.0091.png | Predicted - ventriculomegaly with a 99.81 percent
confidence.
Actual - image.0079.png | Predicted - ventriculomegaly with a 99.87 percent
confidence.
```



LIME

[19]:
```
!pip install lime &> /dev/null
```

[20]:
```
import lime
from lime import lime_image
from lime import submodular_pick
from skimage.segmentation import mark_boundaries
```

[21]:
```
def explainer_predict_fn(img_array):
  return new_model.predict(img_array,verbose = 0)
```

[22]:
```
def lime_exp(img_array,model):

  explainer = lime_image.LimeImageExplainer()
  exp = explainer.explain_instance(img_array[0].numpy(),
                                   explainer_predict_fn,
                                   top_labels=5,
                                   hide_color=0,
                                   num_samples=1000)

  return exp
```

[23]:
```
def generate_prediction_sample(exp, exp_class, show_positive = True, hide_background = True):

    image, mask = exp.get_image_and_mask(exp_class,
                                         positive_only=show_positive,
                                         num_features=8,
                                         hide_rest=hide_background
                                        )
    img_boundry = mark_boundaries(image, mask)
    return img_boundry
```

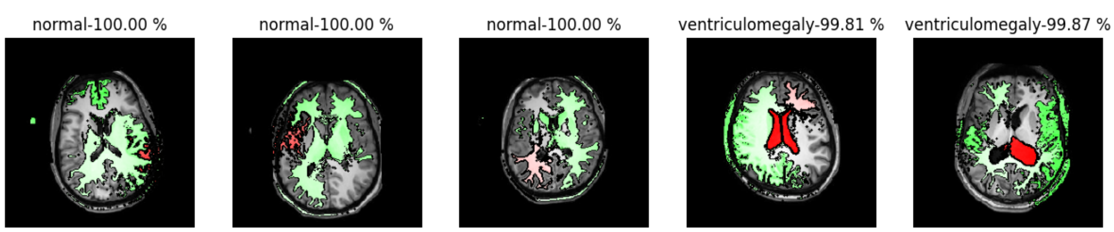
[24]:
```
def show_images_to_pred(images_to_predict):

  fig = plt.figure(figsize = (15, 15))
  cnt = 0
```

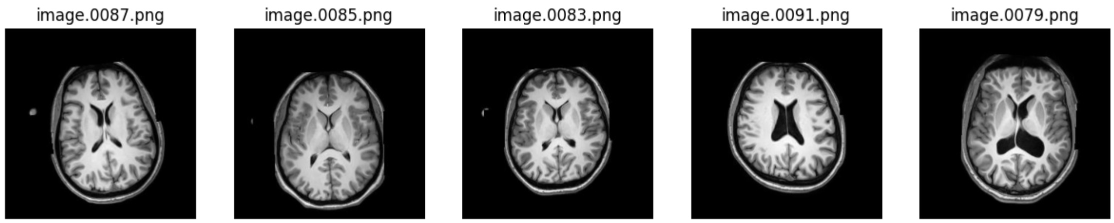
```
  for i in images_to_predict:

    ax = fig.add_subplot(1, len(images_to_predict), cnt+1)
    ground = plt.imread(images_to_predict[cnt])
    plt.imshow(ground,cmap=plt.cm.gray)
    plt.axis('off')
    plt.title(images_to_predict[cnt].split('/')[5])
    cnt+=1
```

[25]:
```
def get_explanations(final_preds,images_to_predict,model):

  fig = plt.figure(figsize = (15, 15))
  cnt = 0

  for i in final_preds:

    exp = lime_exp(i[1],model)
    img_boundry = generate_prediction_sample(exp, exp.top_labels[0],␣
 ↪show_positive = False, hide_background = False)

    ax = fig.add_subplot(1, len(final_preds), cnt+1)
    plt.imshow(img_boundry.astype('uint8'))
    plt.axis('off')
    plt.title(i[0])
    cnt+=1

  show_images_to_pred(images_to_predict)
```

[29]:
```
get_explanations(final_preds,images_to_predict,new_model)
```

```
0%|          | 0/1000 [00:00<?, ?it/s]

0%|          | 0/1000 [00:00<?, ?it/s]

0%|          | 0/1000 [00:00<?, ?it/s]

0%|          | 0/1000 [00:00<?, ?it/s]

0%|          | 0/1000 [00:00<?, ?it/s]
```





GRADCAM

[30]:
```python
import os

os.environ["KERAS_BACKEND"] = "tensorflow"

import numpy as np
import tensorflow as tf
import keras

# Display
from IPython.display import Image, display
import matplotlib as mpl
import matplotlib.pyplot as plt
```

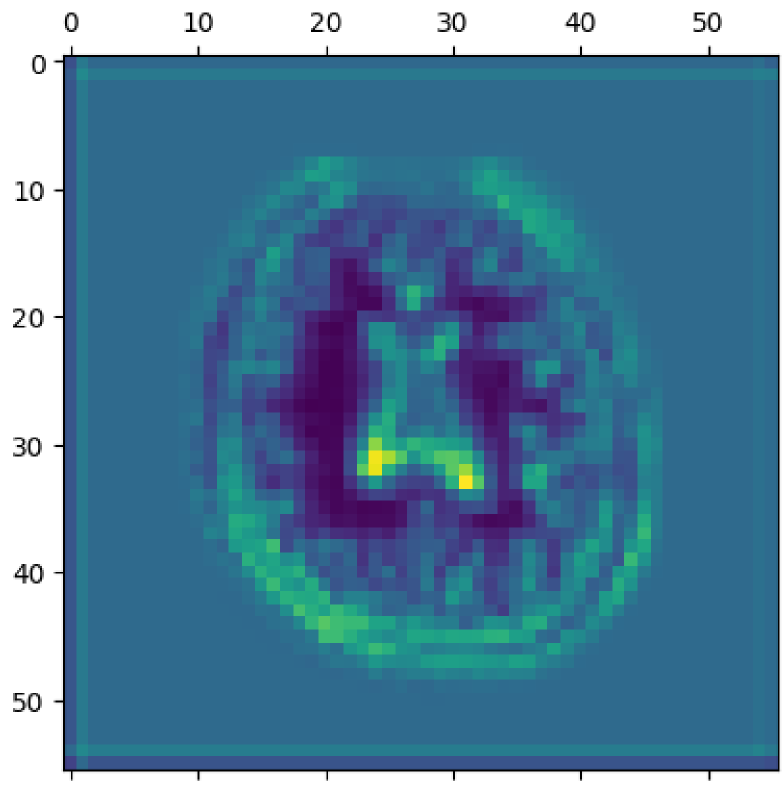
[39]:
```python
model_builder = keras.applications.xception.Xception
img_size = (224, 224)
preprocess_input = keras.applications.xception.preprocess_input
decode_predictions = keras.applications.xception.decode_predictions

last_conv_layer_name = "conv2d_2"

# The local path to our target image
img_path =  '/content/drive/MyDrive/img/image.0091.png'
display(Image(img_path))
```

[40]:
```
def get_img_array(img_path, size):
    # `img` is a PIL image of size 299x299
    img = keras.utils.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = keras.utils.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 299, 299, 3)
    array = np.expand_dims(array, axis=0)
    return array


def make_gradcam_heatmap(img_array, model, last_conv_layer_name,␣
↪pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = keras.models.Model(
        model.inputs, [model.get_layer(last_conv_layer_name).output, model.
↪output]
    )

    # Then, we compute the gradient of the top predicted class for our input␣
↪image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
```

```
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 &
 ↪1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

[41]:
```
from tensorflow.keras.applications.imagenet_utils import decode_predictions,
 ↪preprocess_input
import matplotlib.pyplot as plt
import numpy as np

# Assuming get_img_array and make_gradcam_heatmap functions are correctly
 ↪defined

img_array = preprocess_input(get_img_array(img_path, size=img_size))

# Make model
model = new_model

# Remove last layer's softmax
model.layers[-1].activation = None

# Print what the top predicted class is
preds = model.predict(img_array)
predicted_class = np.argmax(preds[0])
print("Predicted class index:", predicted_class)

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)

# Display heatmap
plt.matshow(heatmap)
plt.show()
```

```
1/1 [==============================] - 0s 38ms/step
Predicted class index: 0
```



[42]:
```python
def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.4):
    # Load the original image
    img = keras.utils.load_img(img_path)
    img = keras.utils.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = mpl.colormaps["jet"]

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # Create an image with RGB colorized heatmap
    jet_heatmap = keras.utils.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = keras.utils.img_to_array(jet_heatmap)
```

```
    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = keras.utils.array_to_img(superimposed_img)

    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
    display(Image(cam_path))


save_and_display_gradcam(img_path, heatmap)
```