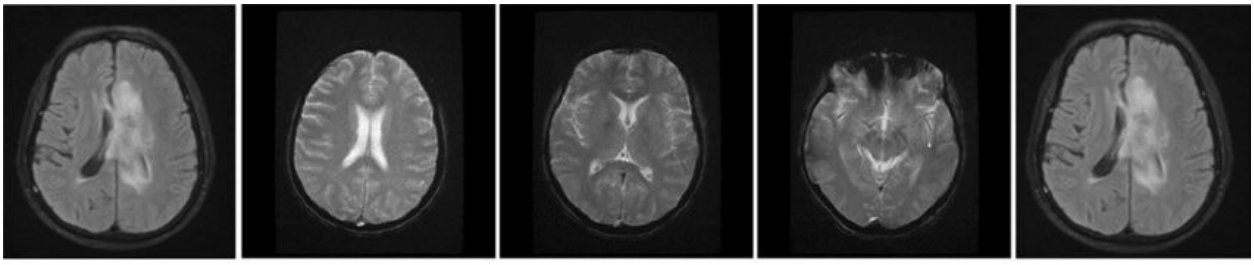
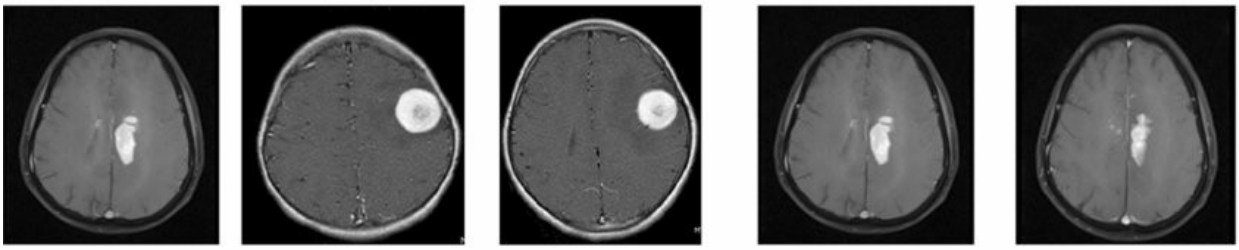
Brain Tumor Classification using DenseNet with Attention Mechanism and visual-explanations-gradcam-gradcam



(a)



(b)

```
import numpy as np
import tensorflow as tf
import pandas as pd
import cv2
import pathlib
import os
import string
from PIL import Image
from keras import backend as K
import matplotlib.pyplot as plt

from keras.preprocessing.image import ImageDataGenerator
ge = ImageDataGenerator(rescale = 1/255,
                        rotation_range=0.2,
                        width_shift_range=0.05,
                        height_shift_range=0.05,
                        fill_mode = 'constant',
                        validation_split = 0.2,
                        horizontal_flip = True,
                        vertical_flip = True,
```

```
                        zoom_range = 0.2
                        )

datasetfolder = '/kaggle/input/brian-tumor-dataset/Brain Tumor Data
Set/Brain Tumor Data Set'
dataflowtraining = ge.flow_from_directory(directory = datasetfolder,
                                  target_size = (224, 224),
                                  color_mode = 'rgb',
                                  batch_size = 32,
                                  shuffle = True,
                                  subset = 'training')
dataflowvalidation = ge.flow_from_directory(directory = datasetfolder,

                                            target_size = (224, 224),
                                            color_mode = 'rgb',
                                            batch_size = 32,
                                            shuffle = True,
                                            subset = 'validation')
```

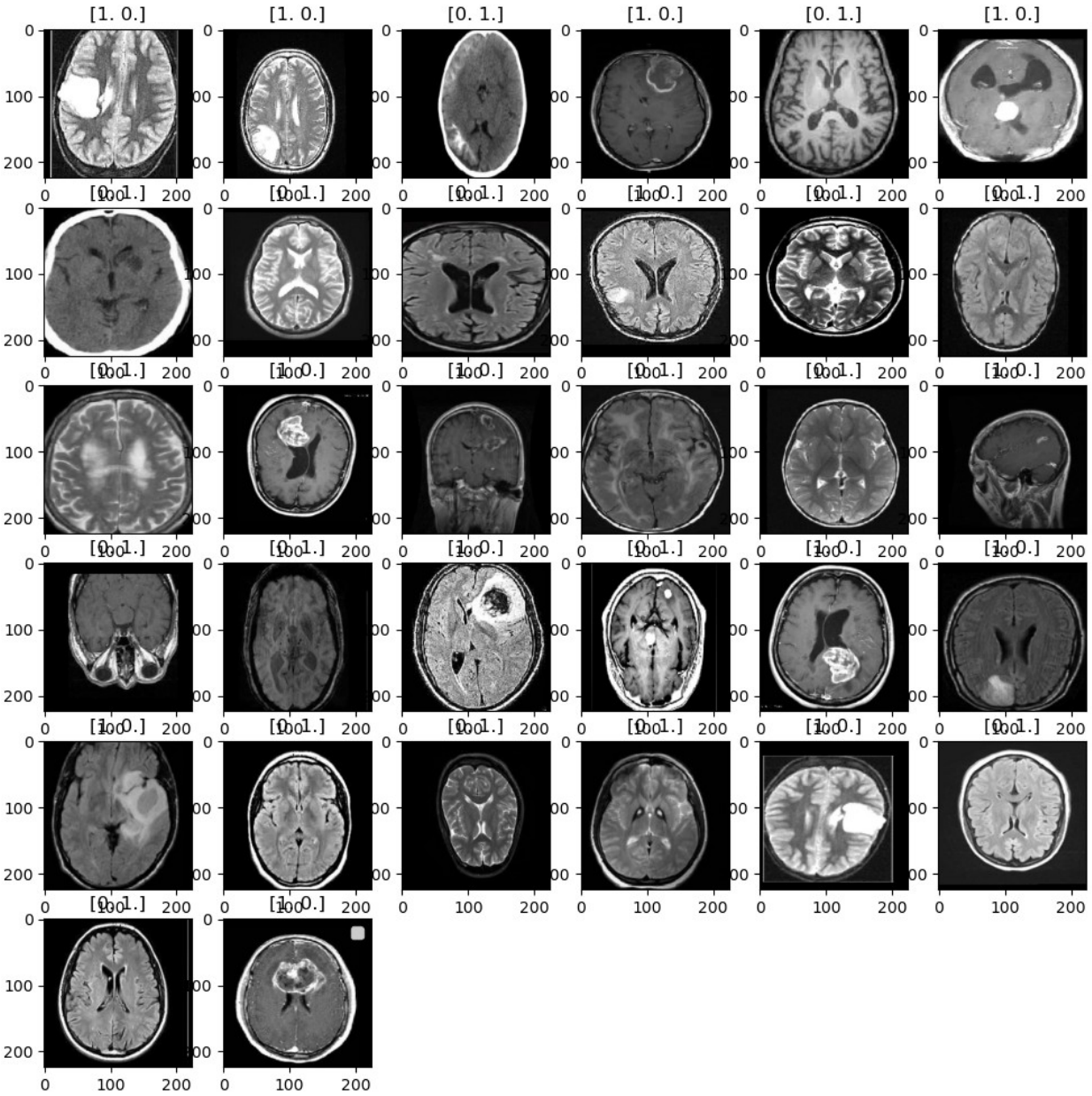
```
Found 3681 images belonging to 2 classes.
Found 919 images belonging to 2 classes.
```

```
images, labels = dataflowvalidation.next()
np.min(images), np.max(images)
```

```
(0.0, 1.0)
```

```
import matplotlib.pyplot as plt
images, labels = dataflowvalidation.next()
plt.figure(figsize = (12, 12))
for i in range(32):
    plt.subplot(6, 6, (i + 1))
    plt.imshow(images[i])
    plt.title(labels[i])
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f263952f670>
```




```
from keras.applications import DenseNet121
import tensorflow as tf
from tensorflow.keras import layers

class MultiHeadSelfAttention(layers.Layer):
    def __init__(self, embed_dim, num_heads=8):
        super(MultiHeadSelfAttention, self).__init__()
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        if embed_dim % num_heads != 0:
            raise ValueError(f"embedding dimension = {embed_dim}
should be divisible by number of heads = {num_heads}")
```

```
        self.projection_dim = embed_dim // num_heads
        self.query_dense = layers.Dense(embed_dim)
        self.key_dense = layers.Dense(embed_dim)
        self.value_dense = layers.Dense(embed_dim)
        self.combine_heads = layers.Dense(embed_dim)

    def attention(self, query, key, value):
        score = tf.matmul(query, key, transpose_b=True)
        dim_key = tf.cast(tf.shape(key)[-1], tf.float32)
        scaled_score = score / tf.math.sqrt(dim_key)
        weights = tf.nn.softmax(scaled_score, axis=-1)
        output = tf.matmul(weights, value)
        return output, weights

    def separate_heads(self, x, batch_size):
        x = tf.reshape(x, (batch_size, -1, self.num_heads,
self.projection_dim))
        return tf.transpose(x, perm=[0, 2, 1, 3])

    def call(self, inputs):
        batch_size = tf.shape(inputs)[0]
        query = self.query_dense(inputs)
        key = self.key_dense(inputs)
        value = self.value_dense(inputs)
        query = self.separate_heads(query, batch_size)
        key = self.separate_heads(key, batch_size)
        value = self.separate_heads(value, batch_size)
        attention, weights = self.attention(query, key, value)
        attention = tf.transpose(attention, perm=[0, 2, 1, 3])
        concat_attention = tf.reshape(attention, (batch_size, -1,
self.embed_dim))
        output = self.combine_heads(concat_attention)
        return output

basemodel = DenseNet121(weights='imagenet', include_top=False,
input_shape=(224, 224, 3), pooling=None)

for layer in basemodel.layers:
    layer.trainable = False

base_output = basemodel.output
gap = layers.GlobalAveragePooling2D()(base_output)
gap_expanded = layers.Reshape((1, gap.shape[-1]))(gap)
embed_dim = 512
projected_gap = layers.Dense(embed_dim)(gap_expanded)
attention_output = MultiHeadSelfAttention(embed_dim, num_heads=8)
(projected_gap)
pooled_attention = layers.GlobalAveragePooling1D()(attention_output)
x = layers.Dropout(0.7)(pooled_attention)
x = layers.BatchNormalization()(x)
```

```
x = layers.Dense(16, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.BatchNormalization()(x)
output = layers.Dense(2, activation='softmax')(x)

m = tf.keras.models.Model(inputs=basemodel.input, outputs=output)

m.compile(loss='binary_crossentropy',
          optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
          metrics=['accuracy',
tf.keras.metrics.Precision(name='precision'),
                   tf.keras.metrics.Recall(name='recall')])

hist = m.fit(dataflowtraining, epochs = 5, batch_size = 32,
             validation_data = dataflowvalidation,
             callbacks = [
                tf.keras.callbacks.EarlyStopping(patience = 8, monitor
= 'val_loss', mode = 'min',
                                                 restore_best_weights =
True),
                tf.keras.callbacks.ReduceLROnPlateau(patience = 6,
monitor = 'val_loss',
                                                     mode =
'min', factor = 0.1)
             ])
```

```
Epoch 1/5
116/116 [==============================] - 60s 445ms/step - loss:
0.6851 - accuracy: 0.6832 - precision: 0.6832 - recall: 0.6832 -
val_loss: 0.4214 - val_accuracy: 0.8357 - val_precision: 0.8357 -
val_recall: 0.8357 - lr: 1.0000e-04
Epoch 2/5
116/116 [==============================] - 49s 426ms/step - loss:
0.5088 - accuracy: 0.8090 - precision: 0.8090 - recall: 0.8090 -
val_loss: 0.3615 - val_accuracy: 0.8825 - val_precision: 0.8825 -
val_recall: 0.8825 - lr: 1.0000e-04
Epoch 3/5
116/116 [==============================] - 49s 423ms/step - loss:
0.4358 - accuracy: 0.8424 - precision: 0.8424 - recall: 0.8424 -
val_loss: 0.3241 - val_accuracy: 0.9021 - val_precision: 0.9021 -
val_recall: 0.9021 - lr: 1.0000e-04
Epoch 4/5
116/116 [==============================] - 48s 415ms/step - loss:
0.4055 - accuracy: 0.8650 - precision: 0.8650 - recall: 0.8650 -
val_loss: 0.2982 - val_accuracy: 0.9075 - val_precision: 0.9075 -
val_recall: 0.9075 - lr: 1.0000e-04
Epoch 5/5
116/116 [==============================] - 48s 418ms/step - loss:
0.3728 - accuracy: 0.8818 - precision: 0.8818 - recall: 0.8818 -
```

```
val_loss: 0.2700 - val_accuracy: 0.9227 - val_precision: 0.9227 -
val_recall: 0.9227 - lr: 1.0000e-04
```

Results

```
m.evaluate(dataflowtraining)
```

```
116/116 [==============================] - 39s 332ms/step - loss:
0.2667 - accuracy: 0.9150 - precision: 0.9150 - recall: 0.9150
```

```
[0.26670822501182556, 0.914968729019165, 0.914968729019165,
0.914968729019165]
```

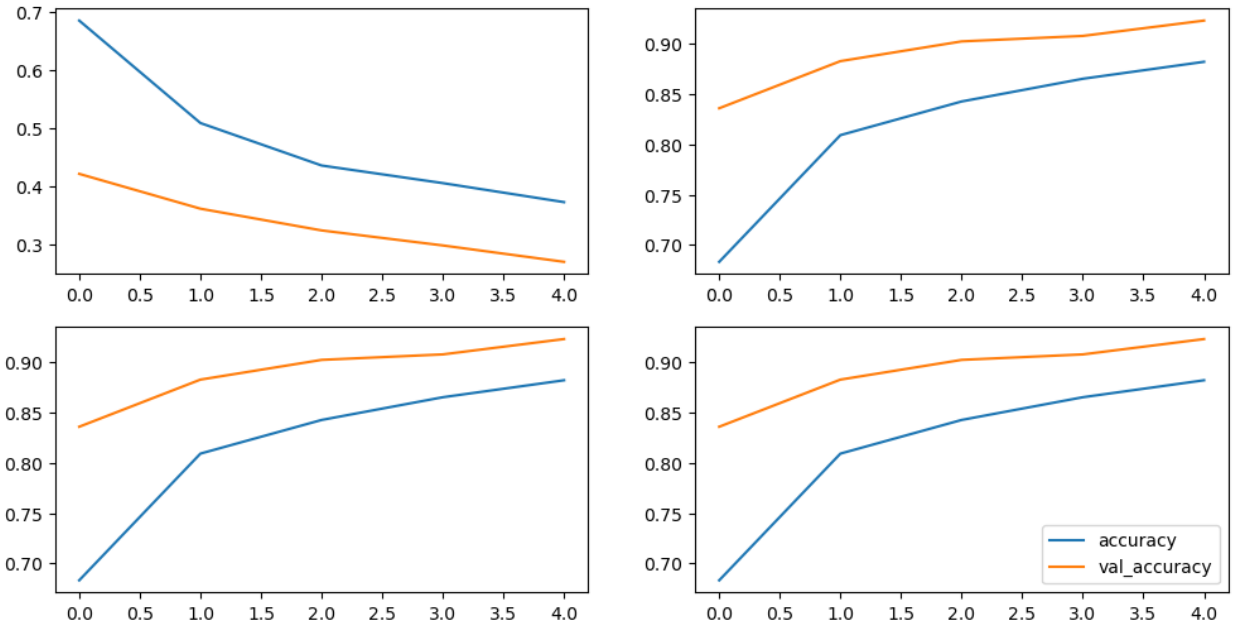
```
m.evaluate(dataflowvalidation)
```

```
29/29 [==============================] - 10s 341ms/step - loss: 0.2833
- accuracy: 0.9042 - precision: 0.9042 - recall: 0.9042
```

```
[0.28333917260169983,
 0.9042437672615051,
 0.9042437672615051,
 0.9042437672615051]
```

```
import matplotlib.pyplot as plt
plt.figure(figsize = (12, 6))
metrics = ['loss', 'precision', 'recall', 'accuracy']
for i in range(4):
    plt.subplot(2, 2, (i + 1))
    plt.plot(hist.history[metrics[i]], label = metrics[i])
    plt.plot(hist.history['val_{}'.format(metrics[i])], label =
'val_{}'.format(metrics[i]))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f21e55738b0>
```




```
m.save('/kaggle/working/final_tumor_model.h5')
```

```
def readtumorImages(imagespathes):
    images = []
    for img in imagespathes:
        img = cv2.imread(str(img))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = img/255
        img = cv2.resize(img, (224, 224))
        images.append(img)
    return np.array(images)

images = readtumorImages(list(pathlib.Path('/kaggle/input/brian-tumor-
dataset/Brain Tumor Data Set/Brain Tumor Data Set/Brain
Tumor').glob('*.*'))[:1000])
labels = tf.keras.utils.to_categorical(tf.zeros(shape =
(images.shape[0])), num_classes = 2)
images.shape, labels.shape
```

```
((1000, 224, 224, 3), (1000, 2))
```

```
indexs = np.random.choice(range(images.shape[0]), size =
(images.shape[0], ))
images = images[indexs]
labels = labels[indexs]
images.shape, labels.shape
```

```
((1000, 224, 224, 3), (1000, 2))
```

```
from keras.models import load_model
```

```
model = load_model('/kaggle/working/final_tumor_model.h5',
                   custom_objects={'MultiHeadSelfAttention':
MultiHeadSelfAttention})
```

Grad-Cam

```
def gradCam(image, true_label, layer_conv_name):
    model_grad = tf.keras.models.Model(inputs = m.input,
                                       outputs =
[m.get_layer(layer_conv_name).output,
                                               m.output])
    with tf.GradientTape() as tape:
        conv_output, predictions = model_grad(image)
        tape.watch(conv_output)
        loss = tf.losses.binary_crossentropy(true_label, predictions)
    grad = tape.gradient(loss, conv_output)
    grad = K.mean(tf.abs(grad), axis = (0, 1, 2))
    conv_output = np.squeeze(conv_output.numpy())
    for i in range(conv_output.shape[-1]):
        conv_output[:,:, i] = conv_output[:,:, i]*grad[i]
    heatmap = tf.reduce_mean(conv_output, axis = -1)
    heatmap = np.maximum(heatmap, 0)
    heatmap = heatmap/tf.reduce_max(heatmap)
    heatmap = cv2.resize(heatmap.numpy(), (224, 224))
    return np.squeeze(heatmap), np.squeeze(image)

def getHeatMap(images, labels):
    heatmaps = []
    for index in range(128):
        heatmap, image = gradCam(images[index: index + 1],
                                                labels[index: index +
1],
                                            'relu')
        heatmaps.append(heatmap)
    return np.array(heatmaps)

heatmaps = getHeatMap(images, labels)
heatmaps.shape
```

```
(128, 224, 224)
```

Grad-Cam++

```
def grad_cam_plus_plus(image, true_label, conv_layer):
    gradModel = tf.keras.models.Model(inputs = m.input, outputs = [
        m.get_layer(conv_layer).output,
```

```
        m.output
    ])
    with tf.GradientTape() as tape:
        conv_output, predict = gradModel(image)
        tape.watch(conv_output)
        score = predict[:, np.argmax(predict,)]
    grads = tape.gradient(score, conv_output)
    grads = K.mean(tf.abs(grads), axis = [0, 1, 2])
    first = K.exp(score)*grads
    second = K.exp(score)*grads*grads
    third = K.exp(score)*grads*grads*grads
    conv_output = np.squeeze(conv_output)
    x = conv_output
    x = np.sum(np.sum(x, axis = 0), axis = 0)
    grads = (second)/(2*second + third*x)
    conv_output = np.array(conv_output)
    for i in range(conv_output.shape[-1]):
        conv_output[:,:,i] = conv_output[:,:,i]*grads[i]
    conv_output = tf.reduce_mean(conv_output, axis = -1)
    heatmap = np.maximum(conv_output, 0)
    heatmap = heatmap/np.max(heatmap)
    heatmap = cv2.resize(heatmap, (224, 224))
    return heatmap

def getHeatMap_plus_plus(images, labels):
    heatmaps = []
    for index in range(128):
        heatmap = grad_cam_plus_plus(images[index: index + 1],
                                                labels[index: index +
1],
                                            'relu')
        heatmaps.append(heatmap)
    return np.array(heatmaps)

GradCamplusheatmaps = getHeatMap_plus_plus(images, labels)
GradCamplusheatmaps.shape
```

```
(128, 224, 224)
```

```
def draw_compare(images, gradcam_heatmaps,
                 gradcamplus_heatmaps, labels):
  plt.figure(figsize = (12, 50))
  index = 0
  n = 0
  for i in range(120):
    plt.subplot(20, 6, (i + 1))
    if index == 0:
      plt.imshow(images[n])
      plt.title('Image-class: {}'.format(labels[n]))
      index = 1
```

```
    elif index == 1:
      plt.imshow(images[n])
      plt.imshow(gradcam_heatmaps[n], alpha = 0.6, cmap = 'jet')
      plt.title('Grad-Cam')
      index = 2
    elif index == 2:
      plt.imshow(images[n])
      plt.imshow(gradcamplus_heatmaps[n], alpha = 0.6, cmap = 'jet')
      plt.title('Grad-Cam++')
      index = 0
      n = n + 1
  plt.legend()
```

```
l = np.argmax(labels, axis = 1)
l.shape
```

```
(1000,)
```

```
import matplotlib.pyplot as plt
draw_compare(images[:40], heatmaps[:40],
                 GradCamplusheatmaps[:40], l[:40])
```

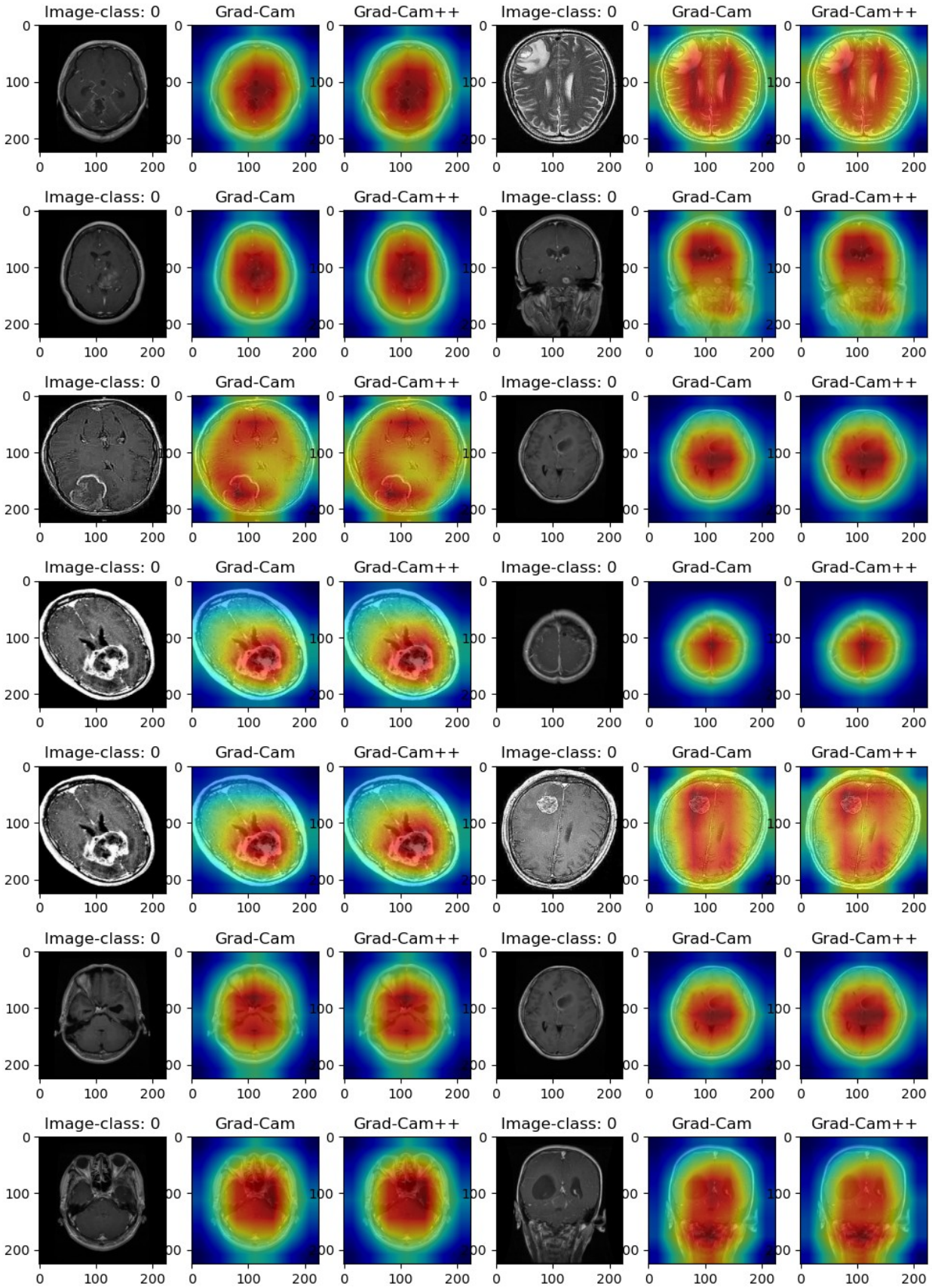


Image-class: 0
Grad-Cam
Grad-Cam++
Image-class: 0
Grad-Cam
Grad-Cam++
0
100
200

```
draw_compare(images[40:80], heatmaps[40:80],
                 GradCamplusheatmaps[40:80], l[40:80])
```

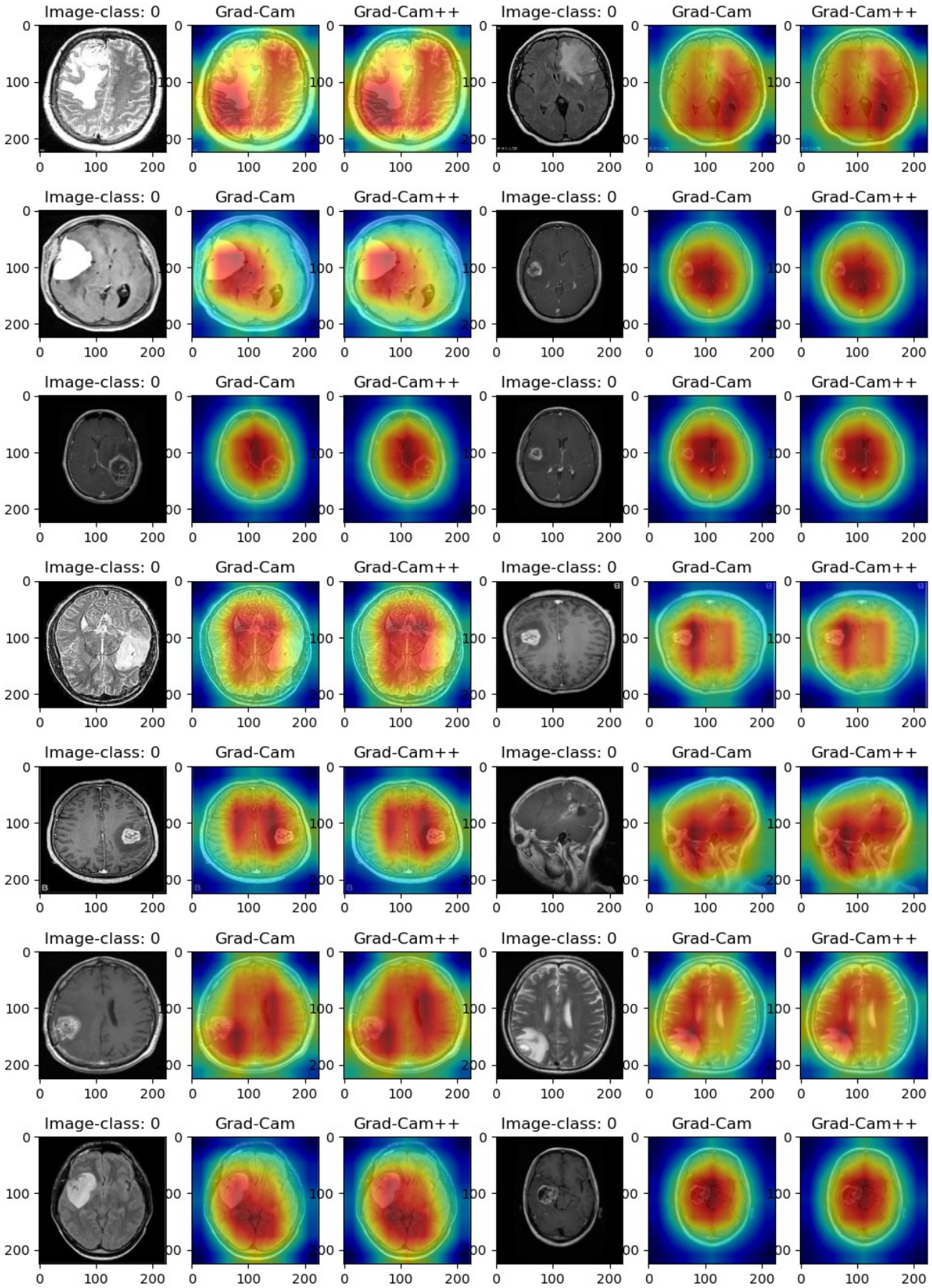


Image-class: 0
Grad-Cam
Grad-Cam++
Image-class: 0
Grad-Cam
Grad-Cam++

```
draw_compare(images[80:120], heatmaps[80:120],
                 GradCamplusheatmaps[80:120], l[80:120])
```

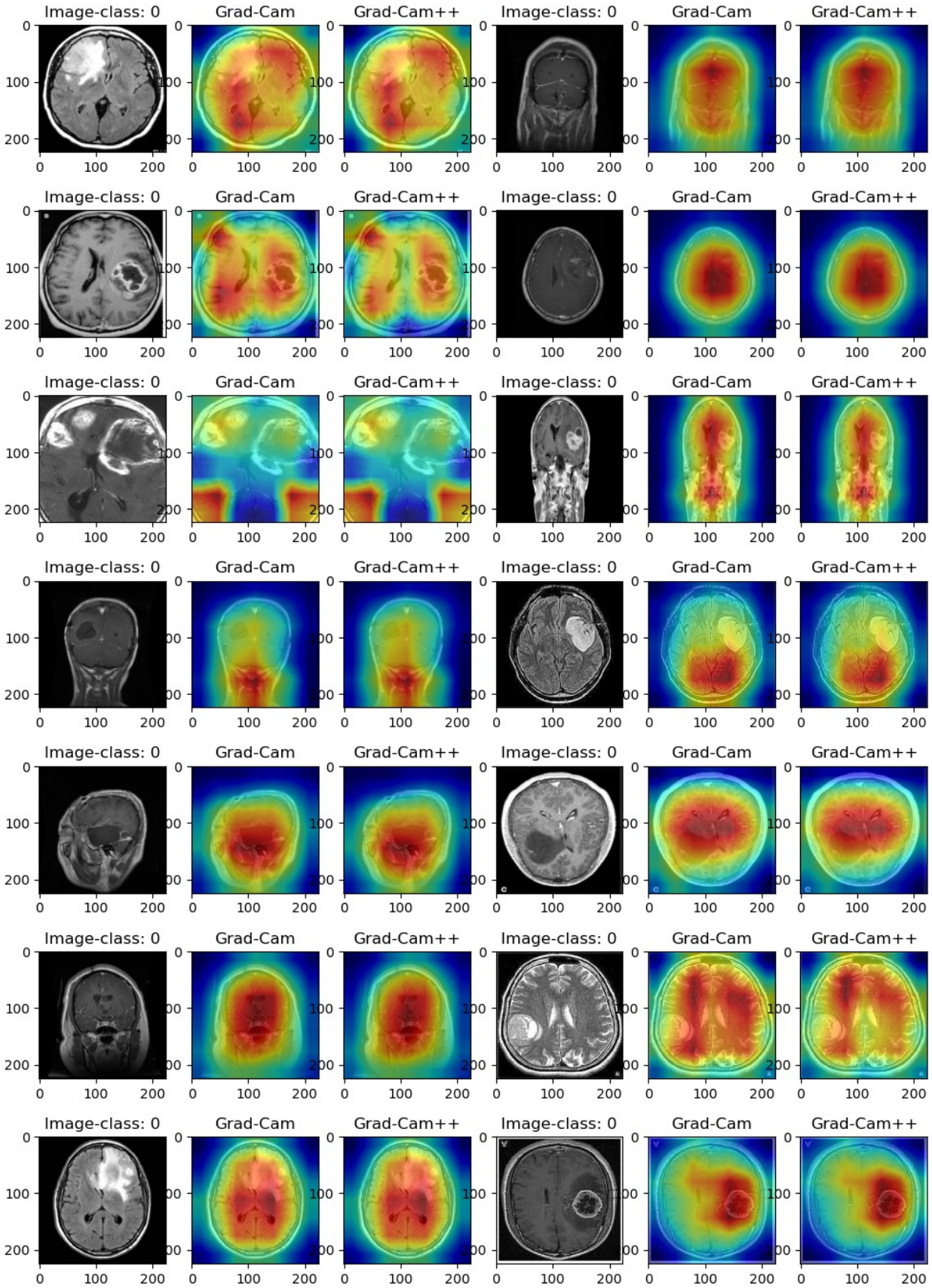


Image-class: 0
Grad-Cam
Grad-Cam++
Image-class: 0
Grad-Cam
Grad-Cam++

References:

- https://arxiv.org/pdf/1610.02391.pdf
- https://arxiv.org/pdf/1710.11063.pdf
- https://arxiv.org/pdf/1910.01279.pdf
- https://arxiv.org/ftp/arxiv/papers/2008/2008.00299.pdf
- https://arxiv.org/pdf/2008.02312.pdf