Bioimage Computer Vision

Luca Pagano

POLITECNICO MILANO 1863 NECST laboratory CAMP

Goal

The objective of the project is fixing the dataset issues (CT scans in Dicom/NIfTI format) and then segmenting the different regions of interest (kidney and tumor) through the use of various techniques related to the Computer Vision field.

Data Analysis

Firstly, I browsed through the various patients to see the format of the CTs. I realize the number of slide isn't constant; through this code I search for the minimum set of slice

```python
min_slice = 50000
case_val = 0
img = []
img_path = DATA_CT

for i in range(0, 300):
    img = nib.load(PATH+'/case_'+f'{i:05d}'+img_path)
    slice_val = img.shape[0]
    if min_slice >= slice_val:
        case_val = i
        min_slice = slice_val


print(min_slice)
print(case_val)
```

Python

```
29
61
```

Further Analysis

The minimum number of slices is 29, found in case 61. The size of the images is 512x512 in all cases but 161, which is 256x256. In addition, only the first 209 patients have a mask. All the CT's looked similar to each other so I've just normalized them. The mask's pixels take the values 0 (background), 1 (kidney), 2 (tumor).

Consequences of hardware limitations

Since the free version of Colab has limited RAM and CPU/GPU power, I’ve made the following choices

- Resampled the images to 256x256
- Resampled the height of CTs arbitrarily to the minimum set of slices found previously (29)
- For simplicity, I'll consider only the first 100 patients, 75% for training and 25% for testing

```python
def get_images(start, stop, img_path):
  images = []


  for cases in range(start, stop):
    # loading the images
    img = nib.load(PATH+'/case_'+f'{cases:05d}'+img_path)
    img = img.get_fdata()

    #for hardware reasons I'm resampling the images with size 256x256 and a number of slices equal to the minimum present
    in the set (29 case_00061)
    img = resize(img, (29,256,256), order=1, preserve_range=True)


    # I add them to the set of images by normalizing them at the same time
    for j in range(0, img.shape[0]):
      images.append(img[j, :, :]/255)
    print(cases)

  return images
```

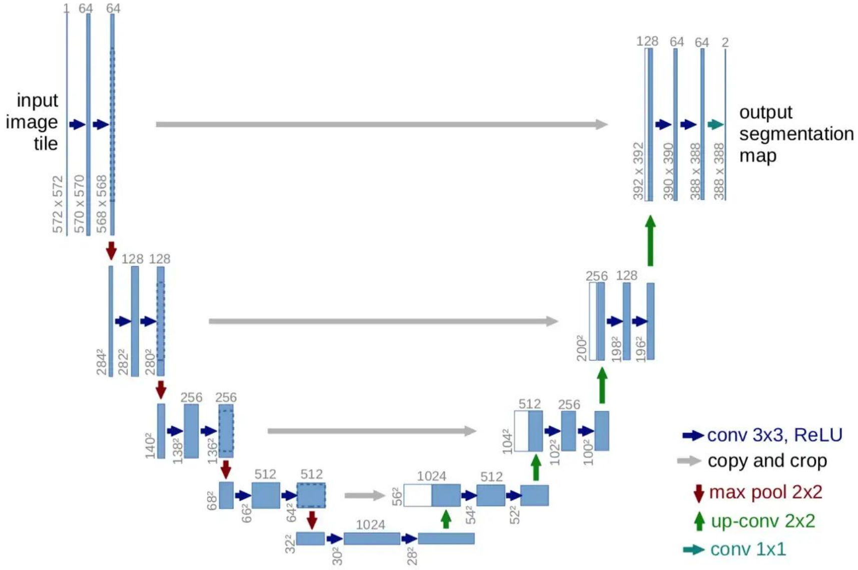
Python

Binary Segmentation

```
trainSet_seg[trainSet_seg > 0.001] = 1
trainSet_seg[trainSet_seg <= 0.001] = 0
```

In the first code, to get acquainted with the Keras library, I restrict myself to only recognizing tumors from patient images, since it is a simpler task. Therefore, I transform the mask values into binary ones.

The dataset seems sufficient enough for the training as the predicted images show.



UNet is a neural network architecture designed for image segmentation tasks. It is made up of a contracting path (encoder) and an expanding path (decoder).

This model is widely used in medical image analysis and has also been applied to other types of image segmentation tasks.

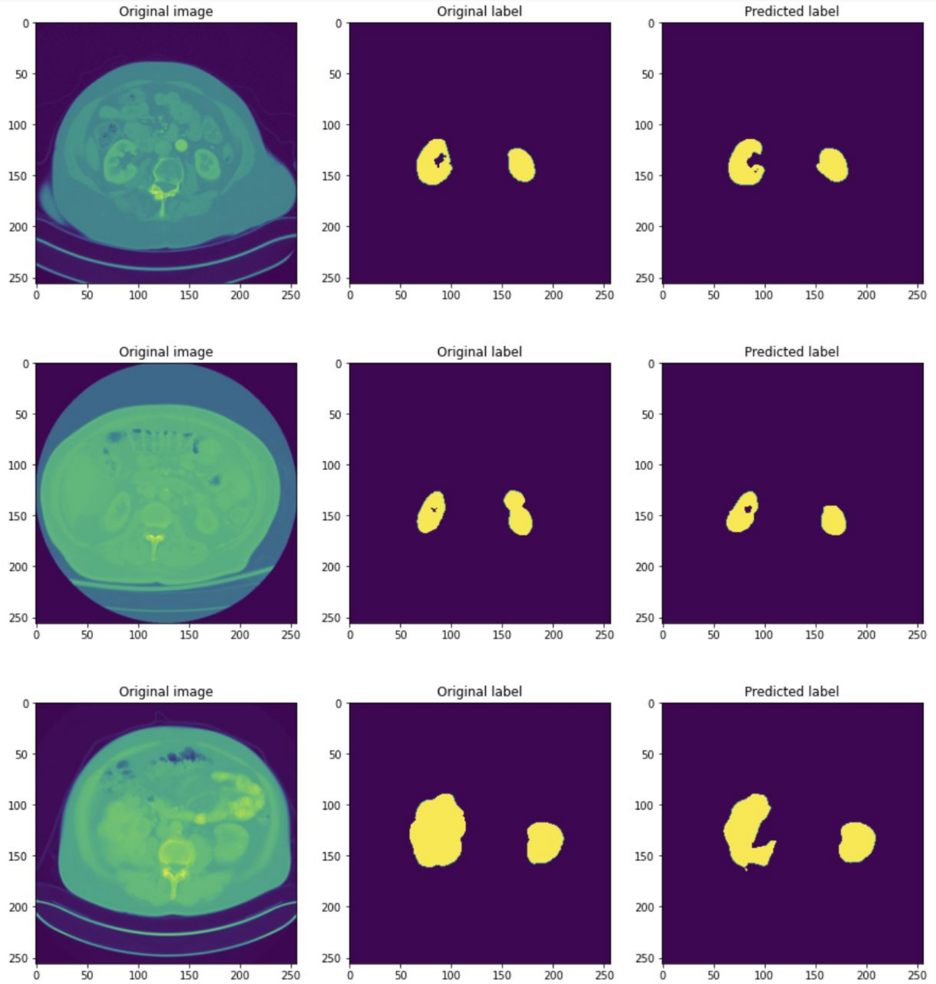
```python
def uNet():

    #Contraction path
    # 256x256
    c1 = tf.keras.layers.Conv2D(
        16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same',
        input_shape=(256, 256, 1))(inputs)
    c1 = tf.keras.layers.Dropout(0.1)(c1)
    c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(c1)

    p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)
    # 128x128
    c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(p1)

    c2 = tf.keras.layers.Dropout(0.1)(c2)
    c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(c2)

    p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)
    # 64x64
    c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(p2)
    c3 = tf.keras.layers.Dropout(0.2)(c3)
    c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(c3)

    p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)
    # 32x32
    c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(p3)
    c4 = tf.keras.layers.Dropout(0.2)(c4)
    c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(c4)

    p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)
    # 16x16
    c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(p4)
    c5 = tf.keras.layers.Dropout(0.3)(c5)
    c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',
                                kernel_initializer='he_normal', padding='same')(c5)
```

```python
#Expansive path
u6 = tf.keras.layers.Conv2DTranspose(
    128, (2, 2), strides=(2, 2), padding='same')(c5)
# 32x32
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(c6)

# 64x64
u7 = tf.keras.layers.Conv2DTranspose(
    64, (2, 2), strides=(2, 2), padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7, c3])
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(u7)
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(c7)

# 128x128
u8 = tf.keras.layers.Conv2DTranspose(
    32, (2, 2), strides=(2, 2), padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(u8)
c8 = tf.keras.layers.Dropout(0.1)(c8)
c8 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(c8)

# 256x256
u9 = tf.keras.layers.Conv2DTranspose(
    16, (2, 2), strides=(2, 2), padding='same')(c8)
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(u9)
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
                            kernel_initializer='he_normal', padding='same')(c9)

outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

return outputs
```

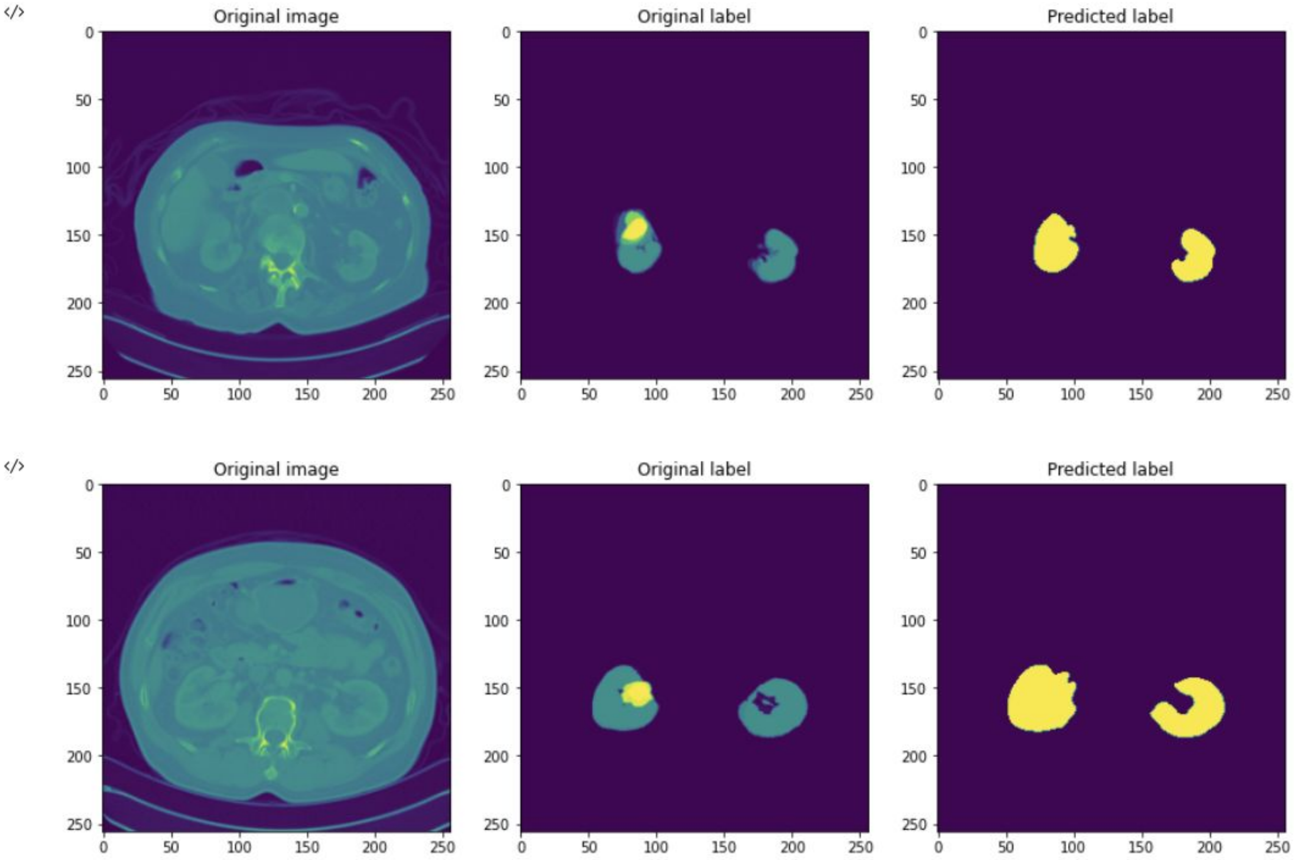
Training

```python
print('-----------------------')
print('Building the model...')
inputs = tf.keras.layers.Input((256, 256, 1))

outputs = uNet()

model = tf.keras.Model(inputs, outputs)
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])

#The following function prints the summary of the model parameters
model.summary()

#Modelcheckpoint -> allows you to save the state of the model
checkpointer = tf.keras.callbacks.ModelCheckpoint('unet_bioimage.h5', verbose=1, save_best_only=True)

# With 'EarlyStopping' stop training if, after "patience" epoch, validation loss does not improve
# With' TensorBoard' we get info about training

callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
    tf.keras.callbacks.TensorBoard(log_dir='logs')
    ]

results = model.fit(trainSet_ct, trainSet_seg, validation_split=0.1,
                    batch_size=8, epochs=5, callbacks=callbacks)
```

Random predicted training images



Random predicted test images



Multiclass Segmentation

Differences

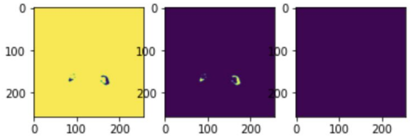
In the second code `multiclass_segmentation.ipynb` I'm trying to distinguish tumors and kidneys; firstly, I triplicated the CT images so that they are three-channel and transformed the masks via ***one_hot_encoding*** so that I have three indices, each one for each class (background, kidney and tumor); this way I could do multiclass segmentation and add specific weights, based on the percentage that each class occupied in the mask. In this case, the dataset proved to be sufficient in detecting kidney but not sufficient in detecting tumors, and I'm guessing this was because of the size of the dataset not being big enough.

```python
def ConvertOneHotEncoded(images):
    '''
    This function converts the mask to one-hot-encoding so that index 0
    is 'backround', index 1 is 'kidneys', index 2 is 'tumor'
    '''
    one_hot_images = []

    N = 256

    # Iterate over the images in the list
    for image in images:
        # Initialize the one-hot encoded array with all elements set to 0
        image_one_hot = np.zeros((N, N, 3), dtype=np.float64)

        # Iterate over the rows and columns of the image
        for i in range(N):
            for j in range(N):
                # Get the pixel value at position (i, j)
                pixel_value = int(image[i, j])
                # Set the corresponding element in the one-hot encoded
                array to 1
                image_one_hot[i, j, pixel_value] = 1

        # Add the one-hot encoded image to the list
        one_hot_images.append(image_one_hot)

    return np.array(one_hot_images)
```

```python
def greyscaleToThreeChannel(original):
    '''Returns the numpy array triplicated '''
    result = np.stack((original,)*3, axis=-1)
    return result
```

```python
def show_classes(y, slice_ex = 10):



    background_ex = y[slice_ex,:,:,0]
    kidney_ex = y[slice_ex,:,:,1]
    tumor_ex = y[slice_ex,:,:,2]

    # Create a figure with a 3x1 grid of subplots
    fig, ax = plt.subplots(1, 3)

    ax[0].imshow(background_ex)

    ax[1].imshow(kidney_ex)

    ax[2].imshow(tumor_ex)

    plt.show()

show_classes(y)
```

...



```python
def calculatePercentageClasses(mask):
    background = mask[:,:,:,0]
    kidney = mask[:,:,:,1]
    tumor = mask[:,:,:,2]

    x = np.count_nonzero(background)
    y = np.count_nonzero(kidney)
    z = np.count_nonzero(tumor)

    return (x/background.size*100, y/kidney.size*100, z/tumor.size*100)

print(calculatePercentageClasses(y))
```

```
(99.53942941249102, 0.392303116020115, 0.06826747148886494)
```

```python
w_b = int(100/(99.54))
w_k = int(100/0.39)
w_t = int(100/0.06)

print(w_b, w_k, w_t)
```

```
1 256 1666
```

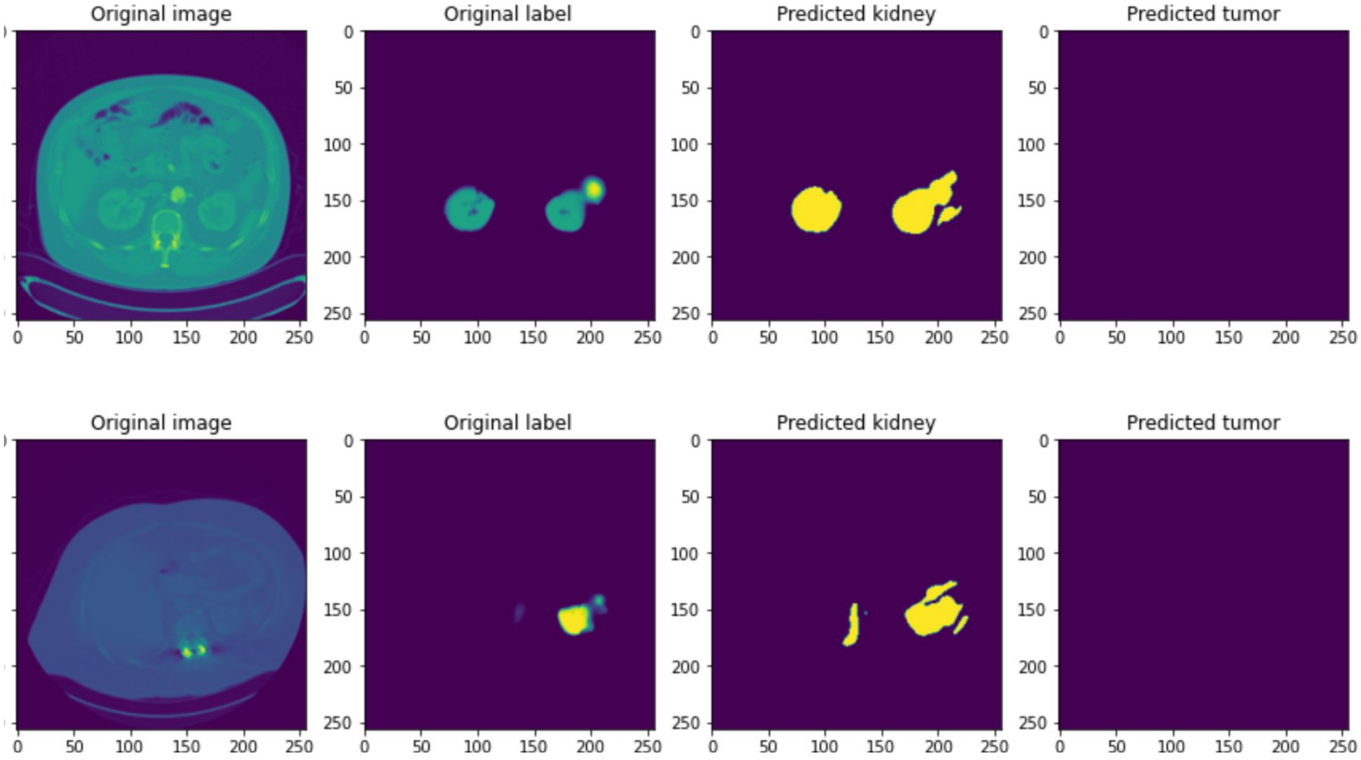
Since I have imbalanced classes, firstly I calculated the percentage of each class in the mask and then calculated the weight I would use for the training

Training

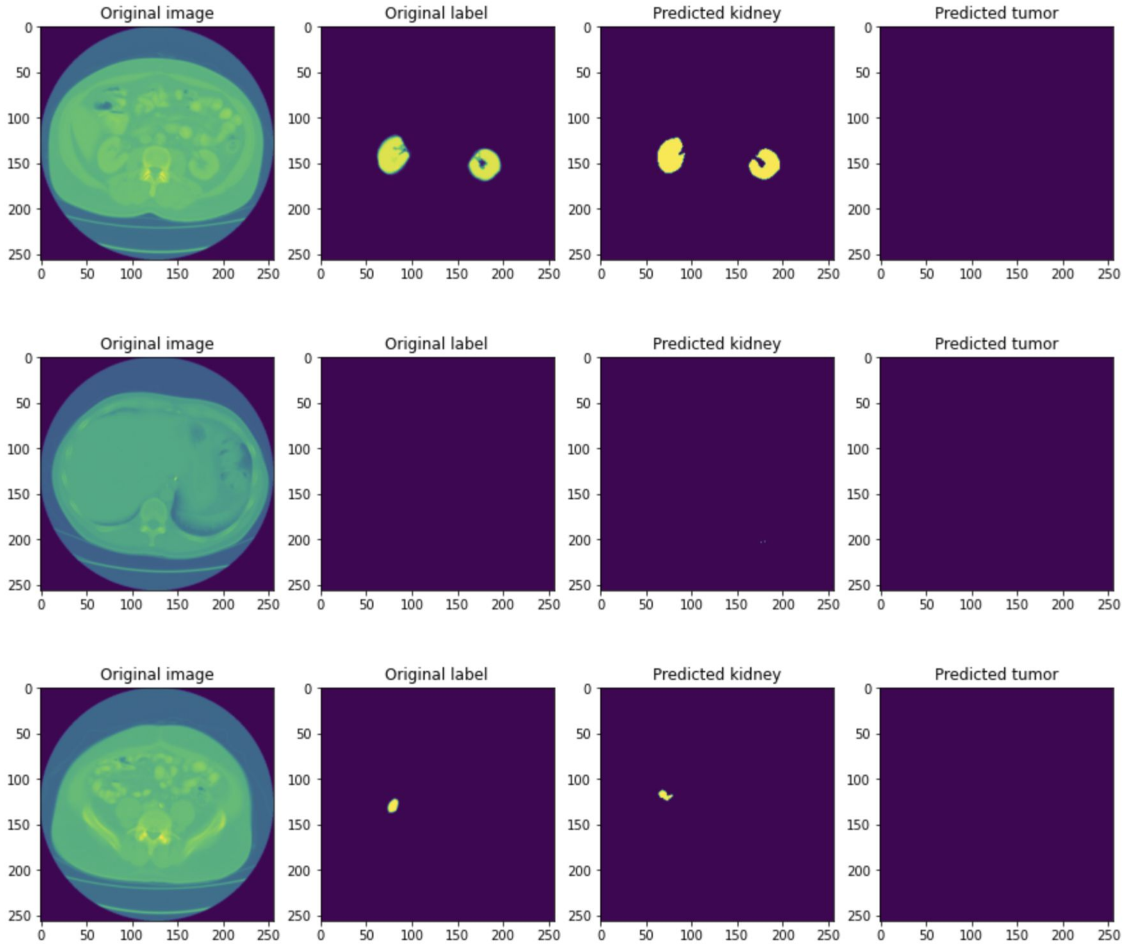
For the training I'm using Keras together with "***segmentation_models***" library, so that I could use as a model "***efficientnetb3***" and add different weight to each class

```python
import segmentation_models as sm
from keras.layers import Input, Conv2D
from keras.models import Model
import numpy as np
import keras

keras.backend.set_image_data_format('channels_last')


model = sm.Unet('efficientnetb3', classes = 3, activation = 'softmax')

total_loss = sm.losses.CategoricalCELoss(class_weights=np.array([w_b, w_k, w_t]))

metrics = [sm.metrics.IOUScore(threshold=0.5, class_weights = np.array([0, 1, 1])), sm.metrics.FScore(threshold=0.5, class_weights = np.
array([0, 1, 1]))]

model.compile(
    'Adam',
    loss = total_loss,
    metrics= metrics,
)

callbacks = [
    keras.callbacks.ModelCheckpoint('/content/drive/MyDrive/best_model.h5', save_weights_only=True, save_best_only=True, mode='min'),
    keras.callbacks.ReduceLROnPlateau(),
]


model.fit(
   x,
   y,
   batch_size=16,
   epochs=30,
   validation_split = 0.1,
  callbacks = callbacks
)
```

Random predicted training image



Random predicted test images



Ways to solve this problem...

To solve the issues of this model one solution could have been to increase the number of patients for training/the number of slices per patient, or even to some data augmentation, by mirroring the image or rotating it, however I could not verify that because of the hardware limitations mentioned above.

The hard way...

I realize that one could have solved the limited RAM problem by implementing an iterator by hand, since the library one does not support the `**.nii**` format, so as to load the images time by time, and take advantage of Keras' *ImageDataGenerator* yet, despite my attempts, my prior knowledge did not allow its implementation. A somewhat cruder solution might have been to convert the numpy arrays to an array of .jpeg images and take advantage of the library generator but I didn't even try that as it seemed an unintelligent solution.

References

https://www.kaggle.com/code/shakshyathedetector/image-segmentation-using-u-net https://github.com/qubvel/segmentation_models

https://github.com/qubvel/segmentation_models/blob/master/examples/multiclass%20segmentation%20(camvid).ipynb

https://github.com/neheller/kits19

https://medium.com/analytics-vidhya/write-your-own-custom-data-generator-for-tensorflow-keras-1252b64e41c3]