Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation

Name	Roll No-Year-Branch-Div
RUSHIL SHAH	1911122-TY-COMPS-B
AAYUSH MALDE	1911090-TY-COMPS-B

INTRODUCTION

Nowadays DL based approach (CNN in particular) provides state-of-the-art performance for image classification, segmentation, detection, tracking and captioning tasks for several reasons:

- First, activation functions resolve training problems in DL approaches.
- Second, dropout helps regularize the networks.
- Third, several efficient optimization techniques are available for training CNN models.

Due to the slow process and tedious nature of manual segmentation approaches, there is a significant demand for computer algorithms that can do tasks like segmentation, classification, detection, registration, and medical information processing quickly and accurately without human interaction.

One deep learning technique, U-Net, has become one of the most popular for these applications. In this implementation, a Recurrent Convolutional Neural Network (RCNN) as well as a Recurrent Residual Convolutional Neural Network (RRCNN) based on U-Net models, which are named RU-Net and R2U-Net is constructed to perform medical image segmentation of Blood Vessels in Retina.

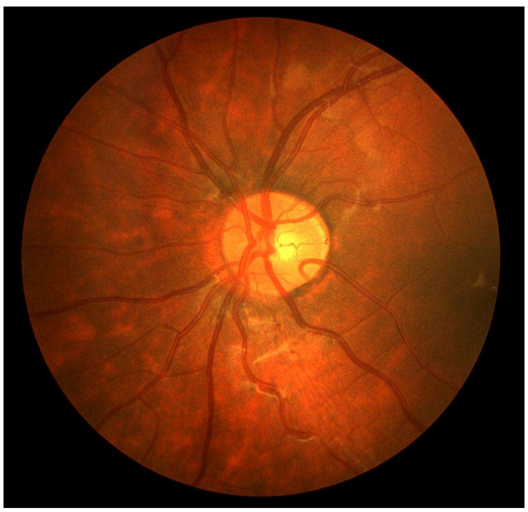
DATABASE:

https://blogs.kingston.ac.uk/retinal/chasedb1/

The database contains 28 images of size 999 x 960 px of human retina and also their segmented images i.e desired output of the model

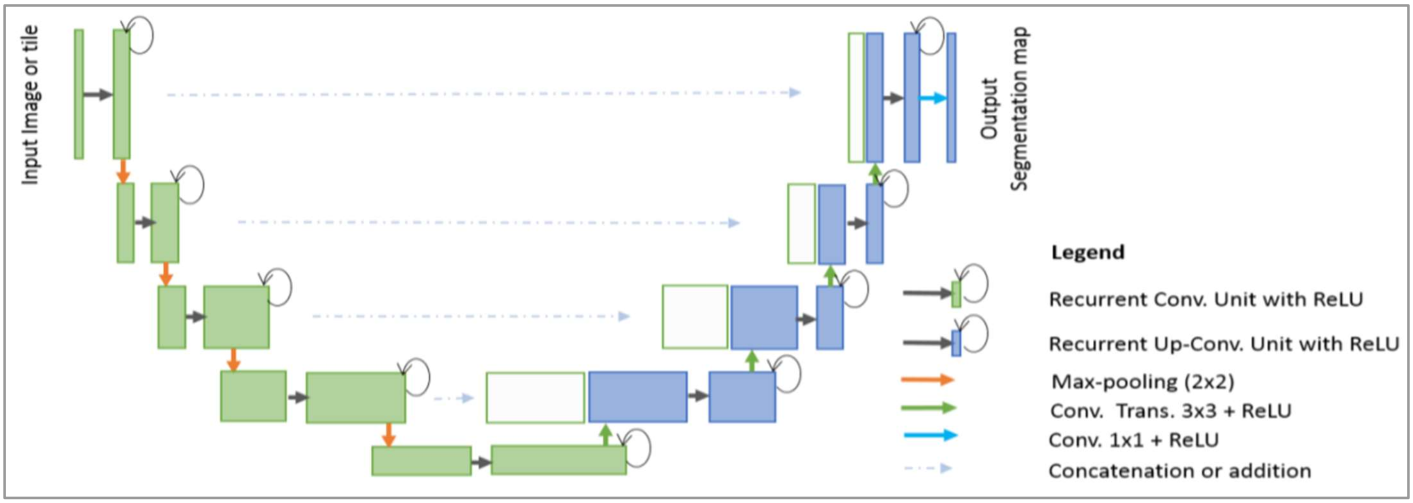
We will divide the images such that, training set will contain 20 images and their respective output images. The testing set will contain the remaining 8 images and their output images.

Example of input image:



Corresponding desired output image:

MODEL ARCHITECTURE:



The architecture of R2Unet comprises of the following blocks:

Convolution Blocks

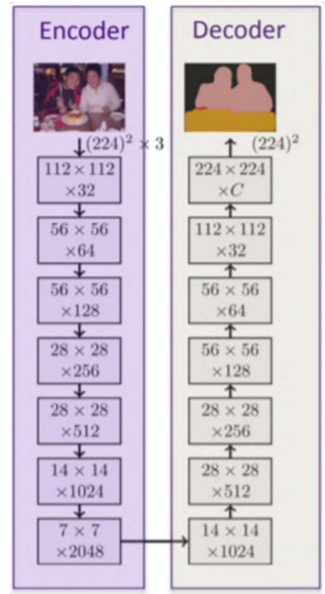
- This is the first layer and one of the main building blocks of a Convolutional Neural Networks (CNNs).
- They hold the raw pixel values of the training image as input.
- This layer ensures the spatial relationship between pixels by learning image features using small squares of input data.

Recurrent Convolutional Block

- Feature accumulation with recurrent convolutional layers ensures better feature representation for segmentation tasks.
- Recurrent network learns from neighbouring units which helps us to include context information of an image.

Encoding Block

- Takes an input image and generates a high dimensional feature vector.
- Aggregate features at multiple levels.

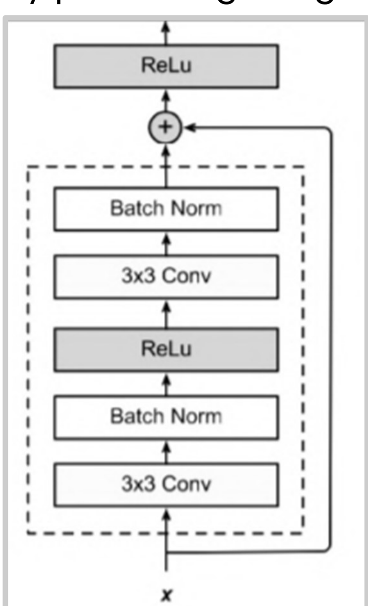


Decoding Block

- Takes a high dimensional feature vector and generates a semantic segmentation mask.
- Decode features aggregated by encoder at multiple levels.

Skip Connections

- The information from the initial layers is passed to deeper layers by matrix addition.
- The presence of the residual blocks prevents the loss of performance whenever the activations tend to vanish or explode by preserving the gradient.



MODEL SUMMARY:

The below image is the summary of the first encoding block as given in the architecture diagram (green blocks).

Similarly, there are several more encoding blocks with similar summary but different shapes.

```
Model: "R2U-Net"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 320, 320, 3) 0
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 320, 320, 64) 1792        input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 320, 320, 64) 256         conv2d_2[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 320, 320, 64) 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 320, 320, 64) 0           activation_1[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 320, 320, 64) 256         input_1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 320, 320, 64) 36928       dropout_1[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 320, 320, 64) 256         conv2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 320, 320, 64) 256         conv2d_3[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 320, 320, 64) 0           batch_normalization_4[0][0]
                                                                 batch_normalization_3[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 320, 320, 64) 0           add[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 160, 160, 64) 0           activation_2[0][0]
```

The below image is that of the first decoding blocks seen in blue color in the architecture diagram. Images decode on several such decoding blocks until they reach their original dimension (320x320 is image size taken for the implementation)

```
__________________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 80, 80, 256)  524544      activation_17[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 80, 80, 512)  0           conv2d_transpose_1[0][0]
                                                                 activation_8[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 80, 80, 256)  1179904     concatenate_1[0][0]
__________________________________________________________________________________________________
batch_normalization_32 (BatchNo (None, 80, 80, 256)  1024        conv2d_32[0][0]
__________________________________________________________________________________________________
activation_19 (Activation)      (None, 80, 80, 256)  0           batch_normalization_32[0][0]
__________________________________________________________________________________________________
dropout_13 (Dropout)            (None, 80, 80, 256)  0           activation_19[0][0]
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 80, 80, 256)  131328      concatenate_1[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 80, 80, 256)  590080      dropout_13[0][0]
__________________________________________________________________________________________________
batch_normalization_34 (BatchNo (None, 80, 80, 256)  1024        conv2d_34[0][0]
__________________________________________________________________________________________________
batch_normalization_33 (BatchNo (None, 80, 80, 256)  1024        conv2d_33[0][0]
__________________________________________________________________________________________________
add_6 (Add)                     (None, 80, 80, 256)  0           batch_normalization_34[0][0]
                                                                 batch_normalization_33[0][0]
__________________________________________________________________________________________________
activation_20 (Activation)      (None, 80, 80, 256)  0           add_6[0][0]
__________________________________________________________________________________________________
```

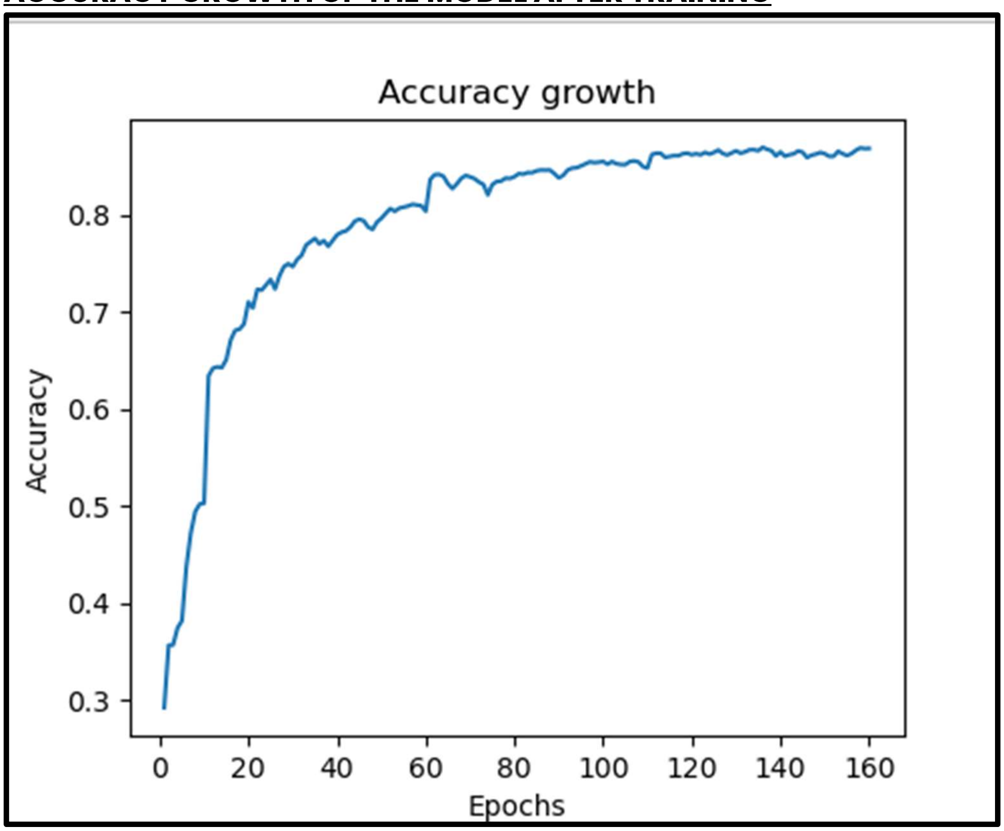
Total parameters in the model:

```
__________________________________________________________________________________________________
activation_26 (Activation)      (None, 320, 320, 64) 0           add_8[0][0]
__________________________________________________________________________________________________
conv2d_45 (Conv2D)              (None, 320, 320, 1)  65          activation_26[0][0]
==================================================================================================
Total params: 32,462,849
Trainable params: 32,445,185
Non-trainable params: 17,664
__________________________________________________________________________________________________
```

MODEL TRAINING ON PC:

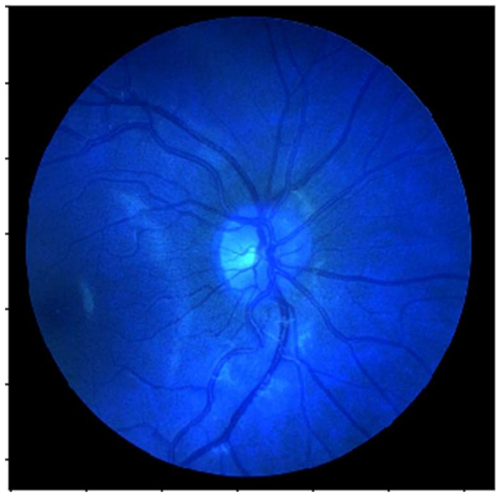
```
 freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were
18/18 [==============================] - 24s 409ms/step - loss: -282.0580 - accuracy: 0.8629 - val_loss: -208.7177 - val_accuracy: 0.8659
C:\Program Files\Anaconda3\lib\site-packages\keras\utils\generic_utils.py:494: CustomMaskWarning: Custom mask layers require a config and m
t_config. When loading, the custom mask layer must be passed to the custom_objects argument.
  warnings.warn('Custom mask layers require a config and must override '
Epoch 2/50
18/18 [==============================] - 6s 322ms/step - loss: -282.0518 - accuracy: 0.8641 - val_loss: -217.1205 - val_accuracy: 0.8520
Epoch 3/50
18/18 [==============================] - 6s 325ms/step - loss: -282.0636 - accuracy: 0.8641 - val_loss: -208.0406 - val_accuracy: 0.8680
Epoch 4/50
18/18 [==============================] - 6s 327ms/step - loss: -281.9944 - accuracy: 0.8599 - val_loss: -212.5617 - val_accuracy: 0.8531
Epoch 5/50
18/18 [==============================] - 6s 327ms/step - loss: -282.0011 - accuracy: 0.8607 - val_loss: -214.2726 - val_accuracy: 0.8547
Epoch 6/50
18/18 [==============================] - 6s 325ms/step - loss: -282.0136 - accuracy: 0.8617 - val_loss: -213.6662 - val_accuracy: 0.8485
Epoch 7/50
18/18 [==============================] - 6s 326ms/step - loss: -282.0250 - accuracy: 0.8615 - val_loss: -212.9982 - val_accuracy: 0.8429
Epoch 8/50
18/18 [==============================] - 6s 333ms/step - loss: -282.0533 - accuracy: 0.8636 - val_loss: -212.4370 - val_accuracy: 0.8590
Epoch 9/50
18/18 [==============================] - 6s 331ms/step - loss: -282.0314 - accuracy: 0.8642 - val_loss: -218.7217 - val_accuracy: 0.8378
Epoch 10/50
18/18 [==============================] - 6s 329ms/step - loss: -282.0393 - accuracy: 0.8625 - val_loss: -217.0926 - val_accuracy: 0.8483
Epoch 11/50
18/18 [==============================] - 6s 327ms/step - loss: -282.0523 - accuracy: 0.8636 - val_loss: -212.2261 - val_accuracy: 0.8561
Epoch 12/50
18/18 [==============================] - 6s 331ms/step - loss: -282.0337 - accuracy: 0.8625 - val_loss: -219.1259 - val_accuracy: 0.8353
Epoch 13/50
18/18 [==============================] - 6s 323ms/step - loss: -282.0563 - accuracy: 0.8650 - val_loss: -213.7791 - val_accuracy: 0.8568
Epoch 14/50
18/18 [==============================] - 6s 323ms/step - loss: -282.0578 - accuracy: 0.8632 - val_loss: -218.8116 - val_accuracy: 0.8317
```

ACCURACY GROWTH OF THE MODEL AFTER TRAINING

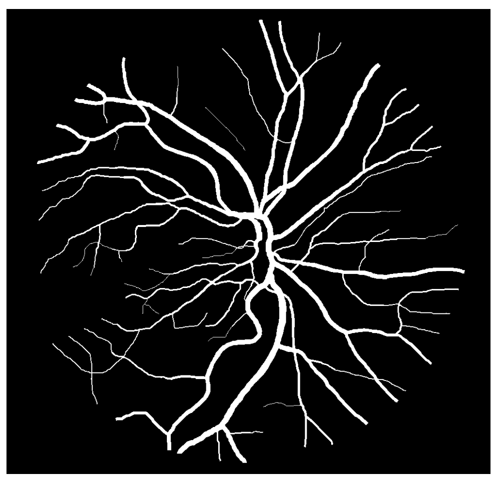


OUTPUT FROM THE MODEL:

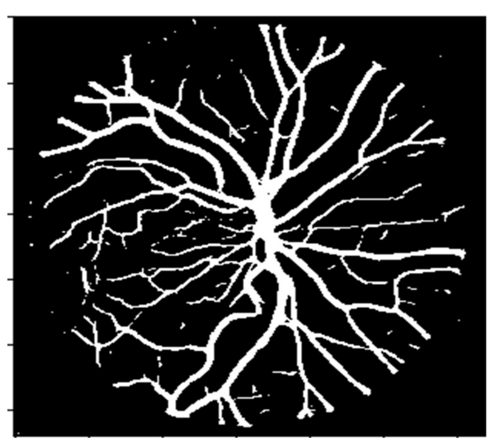
Input image:



Desired OUtput:



Output from the model (86% ACCURACY APPROX):



GITHUB REPOSITORY LINK:
https://github.com/aayushOz11/R2UNET-image-segmentaiton.git

You will find the python code of the model in the main.py file.
The flask implementation of the model can be found in the app.py file.