Machine Learning model for Image Segmentation using U-Net and LinkNet Architecture

Image segmentation is a visual recognition task that can used to simply locate objects and backgrounds so that the image can represented as simple and easier to analyze. In this project, I trained the dataset with U-net and LinkNet models and compare the performance of the models. The dataset consists of around 67 input images and their respective segmentation masks.

MODEL-1

U-Net

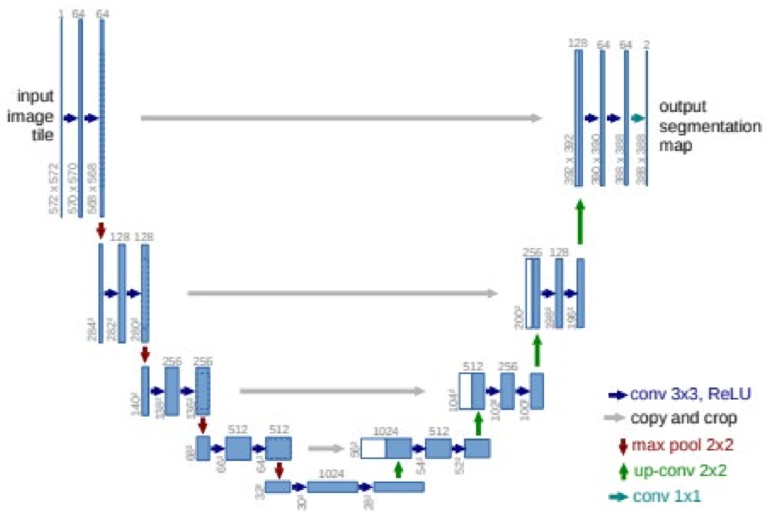
Many deep learning architectures have been proposed to solve various image processing challenges. LeNet, InceptionV3 MobileNet are some examples. The U-Net architecture was published in 2015 and is a network training strategy that relies on the use of data augmentation to use the available annotated samples efficiently. The U-Net architecture is shown in the figure below. It consists four encoder and decoder blocks. The backbone is the model to be used for the encoder part of the UNet. This lets us benefit from transfer learning by using pre-trained weights using Imagenet or ResNet.

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany
ronneber@informatik.uni-freiburg.de,
WWW home page: http://lmb.informatik.uni-freiburg.de/

2



Encoder:

```
# -- Encoder -- #
# Block encoder 1
inputs = Input(shape=input_size)
conv_enc_1 = Conv2D(64, 3, activation='relu', padding='same', kernel_initializer=initializer)(inputs)
conv_enc_1 = Conv2D(64, 3, activation = 'relu', padding='same', kernel_initializer=initializer)(conv_enc_1)

# Block encoder 2
max_pool_enc_2 = MaxPooling2D(pool_size=(2, 2))(conv_enc_1)
conv_enc_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(max_pool_enc_2)
conv_enc_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_enc_2)

# Block  encoder 3
max_pool_enc_3 = MaxPooling2D(pool_size=(2, 2))(conv_enc_2)
conv_enc_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(max_pool_enc_3)
conv_enc_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_enc_3)

# Block  encoder 4
max_pool_enc_4 = MaxPooling2D(pool_size=(2, 2))(conv_enc_3)
conv_enc_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(max_pool_enc_4)
conv_enc_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_enc_4)
# -- Encoder -- #

# ----------- #
maxpool = MaxPooling2D(pool_size=(2, 2))(conv_enc_4)
conv = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(maxpool)
conv = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv)
# ----------- #
```

Decoder

```
# -- Decoder -- #
# Block decoder 1
up_dec_1 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2)
merge_dec_1 = concatenate([conv_enc_4, up_dec_1], axis = 3)
conv_dec_1 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_1)
conv_dec_1 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_1)

# Block decoder 2
up_dec_2 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2)
merge_dec_2 = concatenate([conv_enc_3, up_dec_2], axis = 3)
conv_dec_2 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_2)
conv_dec_2 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_2)

# Block decoder 3
up_dec_3 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2)
merge_dec_3 = concatenate([conv_enc_2, up_dec_3], axis = 3)
conv_dec_3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_3)
conv_dec_3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_3)

# Block decoder 4
up_dec_4 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = initializer)(UpSampling2D(size = (2,2))
merge_dec_4 = concatenate([conv_enc_1, up_dec_4], axis = 3)
conv_dec_4 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(merge_dec_4)
conv_dec_4 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_4)
conv_dec_4 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = initializer)(conv_dec_4)
# -- Decoder -- #

output = Conv2D(N_CLASSES, 1, activation = 'softmax')(conv_dec_4)
```

Compling the Model

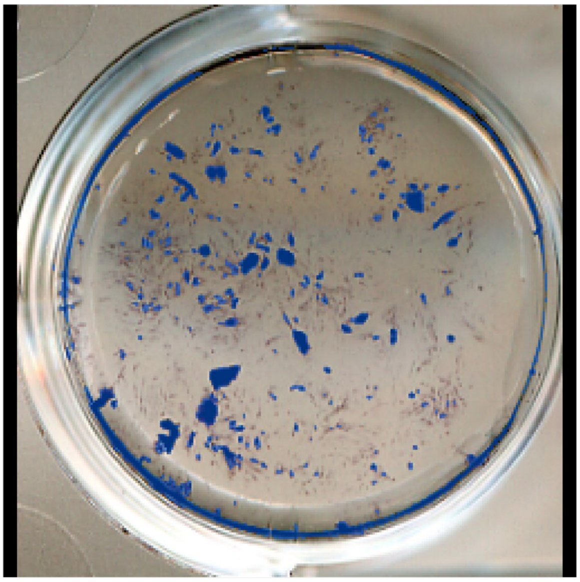
```
model = tf.keras.Model(inputs = inputs, outputs = output)
```

```
model.compile(optimizer = Adam(learning_rate=0.0001),
              loss = dice_loss,
              metrics=['accuracy'])
```

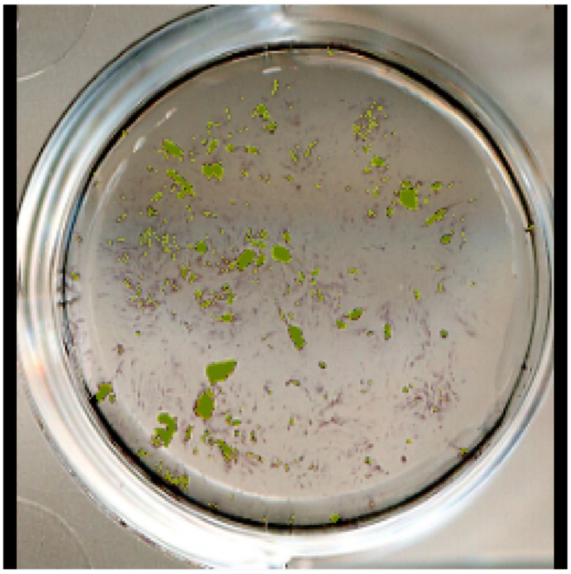
Training the Model for 50 epochs

```
Epoch 41/50
27/27 [==============================] - 211s 8s/step - loss: 0.9467 - accuracy: 0.9814 - val_loss: 0.9612 - val_accuracy: 0.
9696
Epoch 42/50
27/27 [==============================] - 211s 8s/step - loss: 0.9467 - accuracy: 0.9806 - val_loss: 0.9612 - val_accuracy: 0.
9714
Epoch 43/50
27/27 [==============================] - 209s 8s/step - loss: 0.9467 - accuracy: 0.9814 - val_loss: 0.9612 - val_accuracy: 0.
9755
Epoch 44/50
27/27 [==============================] - 211s 8s/step - loss: 0.9467 - accuracy: 0.9815 - val_loss: 0.9612 - val_accuracy: 0.
9743
Epoch 45/50
27/27 [==============================] - 211s 8s/step - loss: 0.9467 - accuracy: 0.9818 - val_loss: 0.9612 - val_accuracy: 0.
```

Predicted Mask



Ground Truth



Model 2 LinkNet

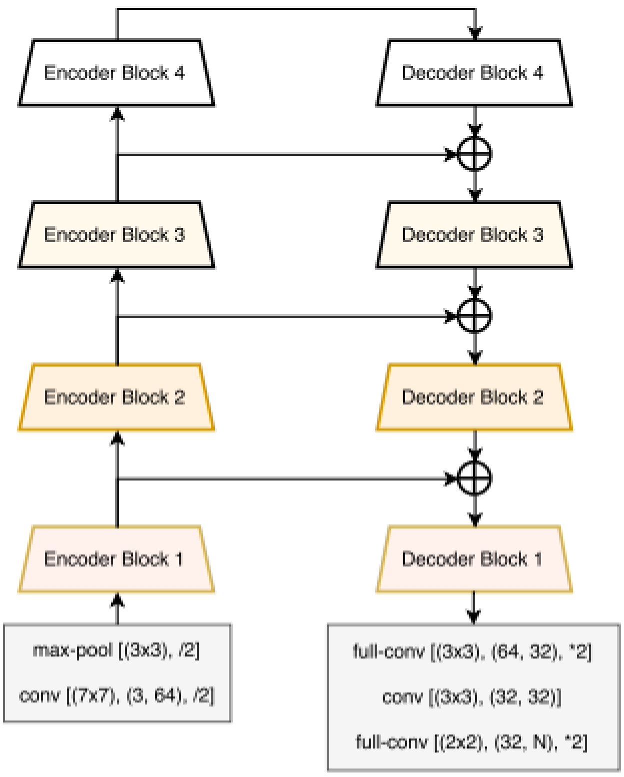
The LinkNet model was published in 2017 and it is a pixel wise segmentation for visual scene understanding and can be used for real-time application.

LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation

Abhishek Chaurasia
School of Electrical and Computer Engineering
Purdue University
West Lafayette, USA
Email: aabhish@purdue.edu

Eugenio Culurciello
Weldon School of Biomedical Engineering
Purdue University
West Lafayette, USA
Email: euge@purdue.edu

The Architecture



The initial block contains a convolution layer with a kernel size of ***7x7*** and a stride of ***2***; followed by a max-pool layer of window size ***2x2*** and stride of ***2***. Similarly, the final block performs full convolution taking feature maps from 64 to 32, followed by 2D-convolution. Finally, we use full-convolution as our classifier with a kernel size of ***2x2***.

Backbone

```
BACKBONE1 = 'resnet34'
import segmentation_models as sm

sm.set_framework('tf.keras')

sm.framework()
preprocess_input1 = sm.get_preprocessing(BACKBONE1)
```

The resnet34 network is used as a backbone here. The stored weights from the network are imported into the model as a starting reference and updated as we train.

Model Compile

```
activation='softmax'

model2 = sm.Linknet(BACKBONE1, encoder_weights='imagenet', classes=N_CLASSES, activation=activation)
```

```
Downloading data from https://github.com/qubvel/classification_models/releases/download/0.0.1/resnet34_imagenet_1000_no_top.h5
85524480/85521592 [==============================] - 4s 0us/step
85532672/85521592 [==============================] - 4s 0us/step
```

Defining a Custom loss function

```
: def dice_loss(y_true, y_pred):
      y_true = tf.cast(y_true, tf.float32)
      y_pred = tf.math.sigmoid(y_pred)
      numerator = 2 * tf.reduce_sum(y_true * y_pred)
      denominator = tf.reduce_sum(y_true + y_pred)
      return 1 - numerator / denominator
```

```
: ##model2.compile(optim, total_loss, metrics=metrics)

  model2.compile(optimizer = Adam(learning_rate=0.0001),
                 loss = dice_loss,
                 metrics=['accuracy'])
```

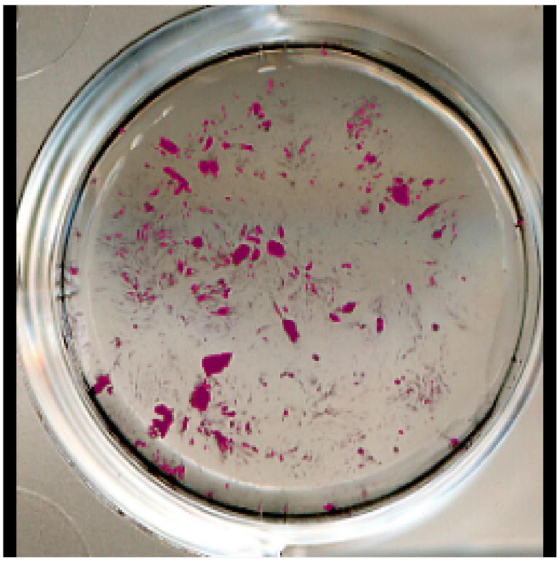
Training the LinkNet Model

```
model2_history = model2.fit(dataset['train'], epochs=EPOCHS,
                            callbacks=[saver],
                            steps_per_epoch=STEPS_PER_EPOCH,
                            validation_steps=VALIDATION_STEPS,
                            validation_data=dataset['val'])
```

```
Epoch 1/50
27/27 [==============================] - 39s 1s/step - loss: 0.9468 - accuracy: 0.9526 - val_loss: 0.9616 - val_accuracy: 0.9
763
Epoch 2/50
27/27 [==============================] - 38s 1s/step - loss: 0.9468 - accuracy: 0.9571 - val_loss: 0.9616 - val_accuracy: 0.9
763
Epoch 3/50
27/27 [==============================] - 38s 1s/step - loss: 0.9468 - accuracy: 0.9606 - val_loss: 0.9616 - val_accuracy: 0.9
763
```

Visualize the Prediction

Ground Truth



Conclusions and Comparisons

The Accuracy of the U-Net model after 50 epochs was 0.981467 and validation loss after 50 epochs was 0.9462.

The Accuracy of the LinkNet model after 50 epochs was 0.9792 and the validation loss after 50 epochs was 0.9612.

By comparing, we can say the both models were able to segment the image during training. Although the LinkNet model was consistently performing faster than the U-Net Model. The UNet model took around 8 seconds per step while training and Linknet took only 1-2 seconds per step.

The predicted segment masks from Unet model was closely resembling the Ground Truth while the LinkNet model's prediction masks were a little noisy. To conclude I would say each model has its own advantages and trade-offs. I would say if the problem needs to be accurate and consistent UNet will be safe bet. And if we need fast results and convergence we can use LinkNet Model.

References

1. LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation https://arxiv.org/pdf/1707.03718.pdf
2. U-Net: Convolutional Networks for Biomedical Image Segmentation https://arxiv.org/pdf/1505.04597.pdf
3. Segmentation Models https://github.com/qubvel/segmentation_models/blob/master/segmentation_models/models/linknet.py
4. Python for Microscopists https://github.com/bnsreenu/python_for_microscopists/blob/master/211_multiclass_Unet_vs_linknet.py