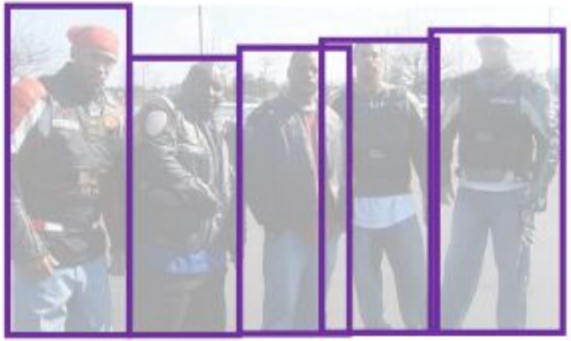
CS-584 Project - Group 17

Instance Segmentation: Cell Nuclei Detection

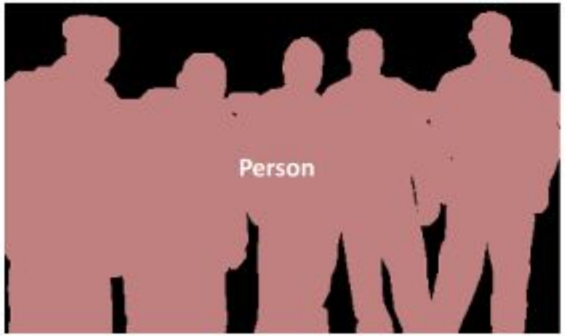
Shivam Kulkarni

April 29, 2021

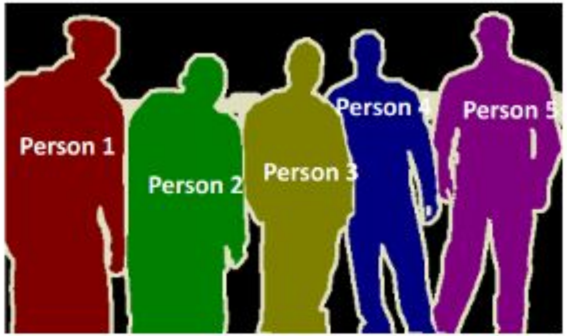
Instance Segmentation



Object Detection



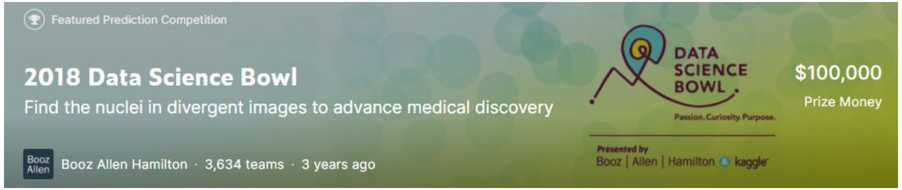
Semantic Segmentation



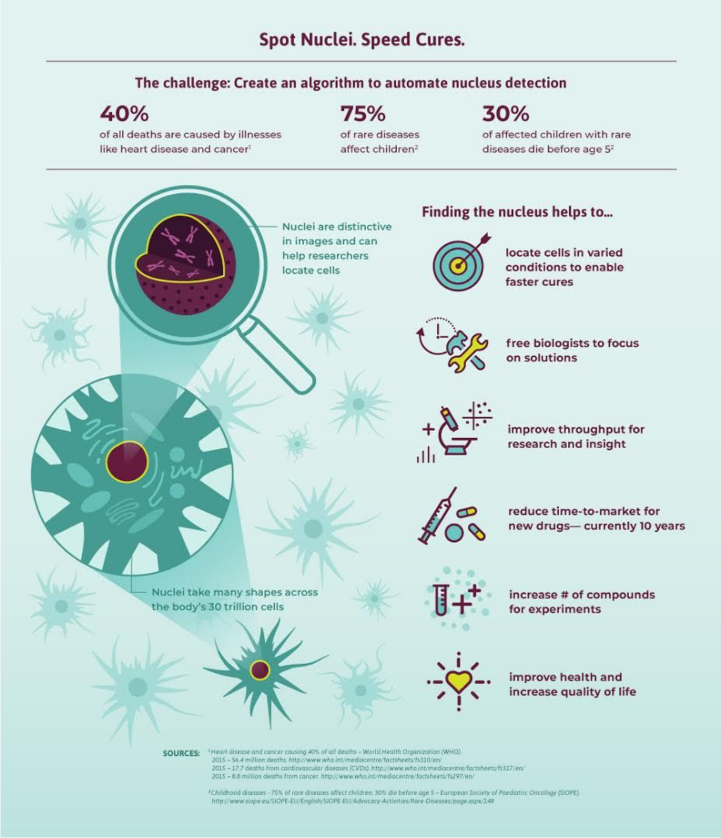
Instance Segmentation

Source: https://towardsdatascience.com

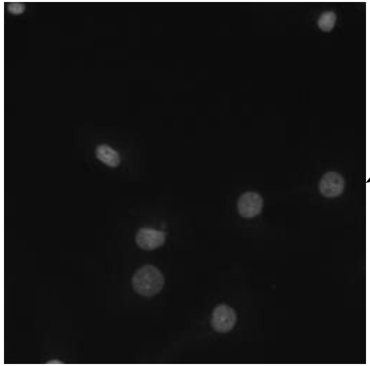
Background



- Identifying a cell's nuclei is often the first step in nuclear/cellular research
- Identifying nuclei allows researchers to identify individual cell in a sample which is important for the researchers to pinpoint the underlying biological process by measuring how cells react to various treatments
- Manually detecting nuclei is an arduous effort; researchers would rather spend time doing research that will lead to significant discoveries
- Manually detecting nuclei is a time-consuming and error-prone process

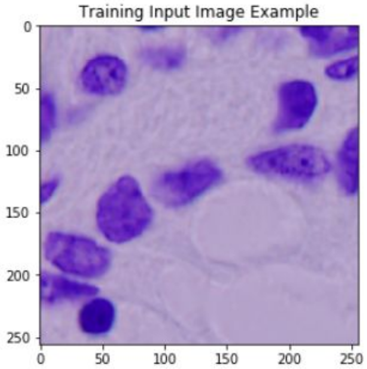


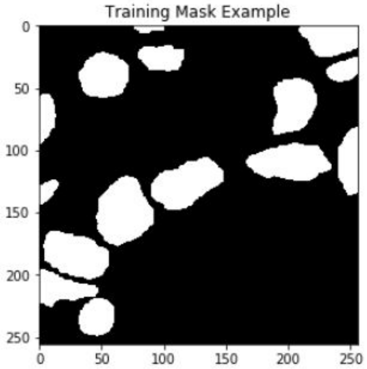
Training Samples



Input image

Corresponding masks of the input image





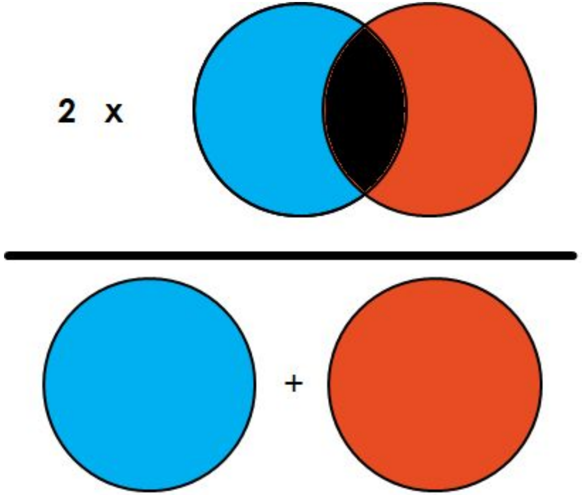
Data Preparation

- 670 training images, 65 test images
- Training examples can have multiple nuclei
- A mask is provided for every nucleus in the training example
  - Stacked all masks of training example onto each other to get y_train
- Some input images had 3 channels, some of them have 4 channels (RGB vs RGBA)
  - Dropped the fourth channel from the data
- Training images varied in sizes
  - Resized the images to consistent 256 x 256 image size
- Used 20% data for cross validation

```
X_train shape : (536, 256, 256, 3)
y_train shape : (536, 256, 256, 1)
X_val shape   : (134, 256, 256, 3)
y_val shape   : (134, 256, 256, 1)
X_test shape  : (65, 256, 256, 3)
```

Performance Metrics

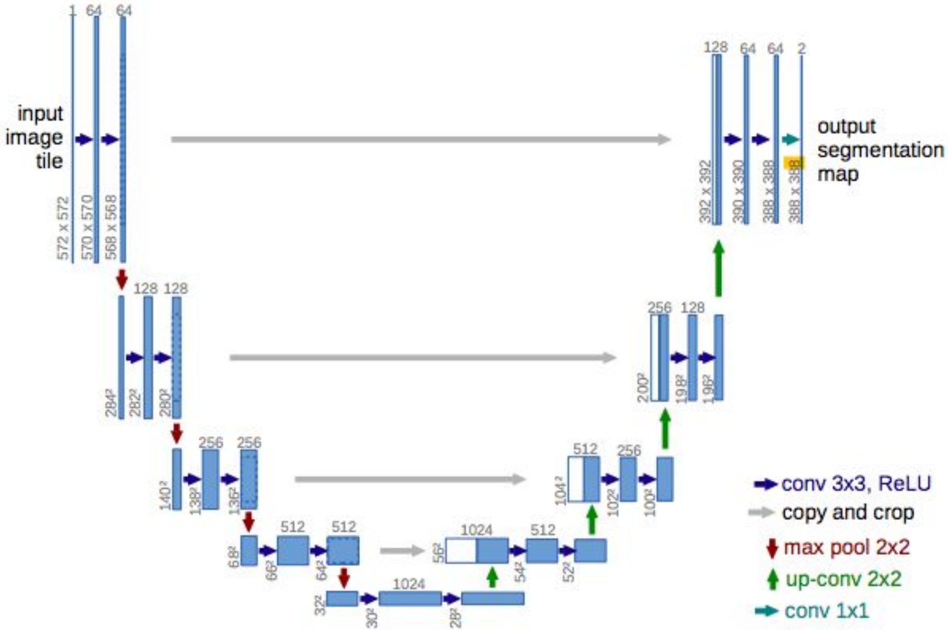
Dice Coefficient $$\frac{2 * |X \cap Y|}{|X| + |Y|}$$



Example:
We have two images having 100 pixels each
Total number of pixels for both images combined = 200
If area of overlap is 0
    Dice = 0/200 = 0
If area of overlap is 95%
    Dice 2*95/200 = 0.95

Source: https://towardsdatascience.com

U-Net Architecture



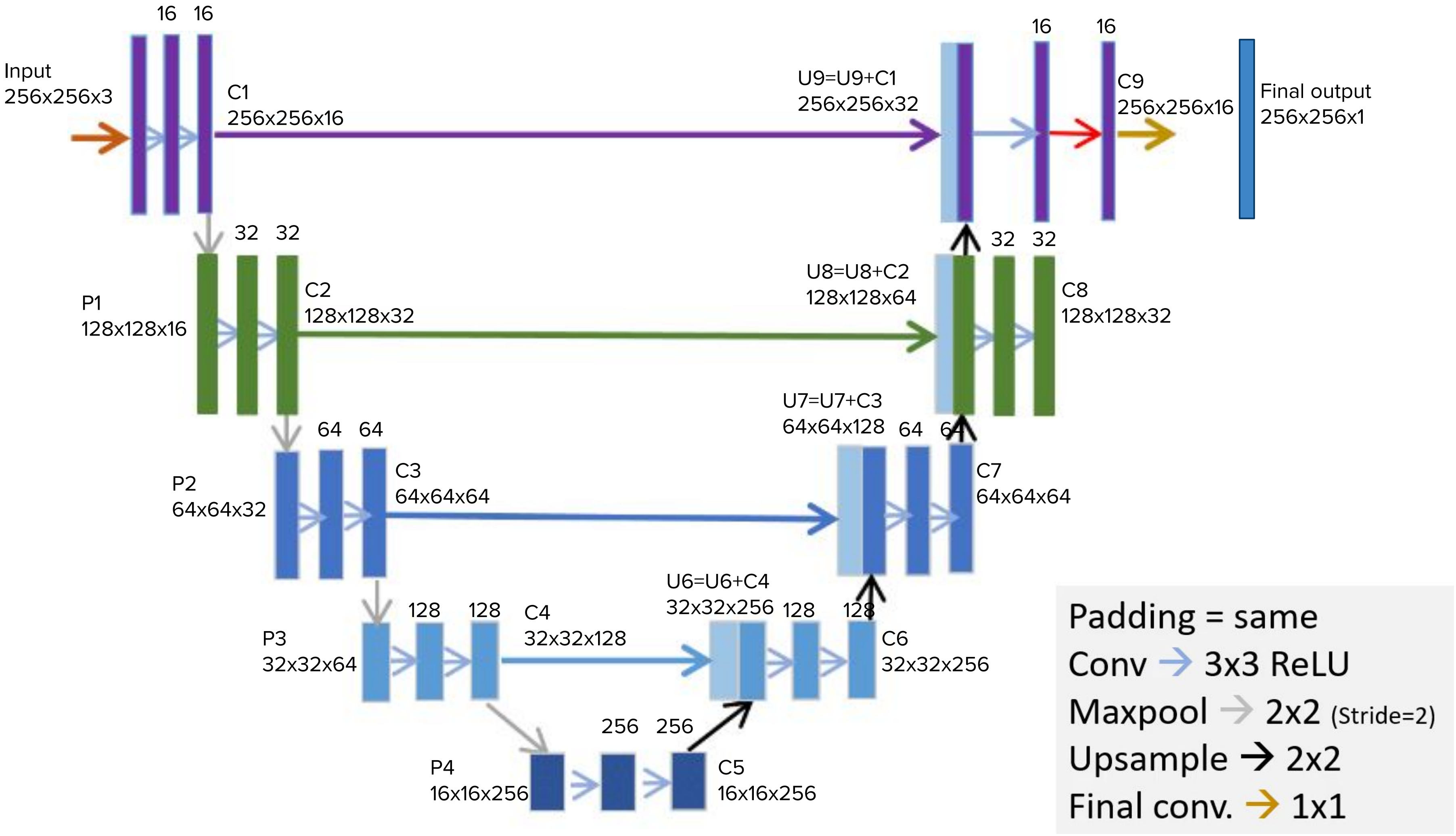
Source: https://arxiv.org/pdf/1505.04597.pdf

Pros:

- Made for bio-medical image segmentation
- Easy to implement using Keras

Cons:

- Does not perform instance segmentation
- Requires square input images (e.g. 256 x 256)
- May overfit on training images


16 16
Input
256x256x3
C1
256x256x16
U9=U9+C1
256x256x32
16 16
C9
256x256x16
Final output
256x256x1
32 32
P1
128x128x16
C2
128x128x32
U8=U8+C2
128x128x64
32 32
C8
128x128x32
64 64
P2
64x64x32
C3
64x64x64
U7=U7+C3
64x64x128
64 64
C7
64x64x64
128 128
P3
32x32x64
C4
32x32x128
U6=U6+C4
32x32x256
128 128
C6
32x32x256
256 256
P4
16x16x256
C5
16x16x256
Padding = same
Conv 3x3 ReLU
Maxpool 2x2 (Stride=2)
Upsample 2x2
Final conv. 1x1

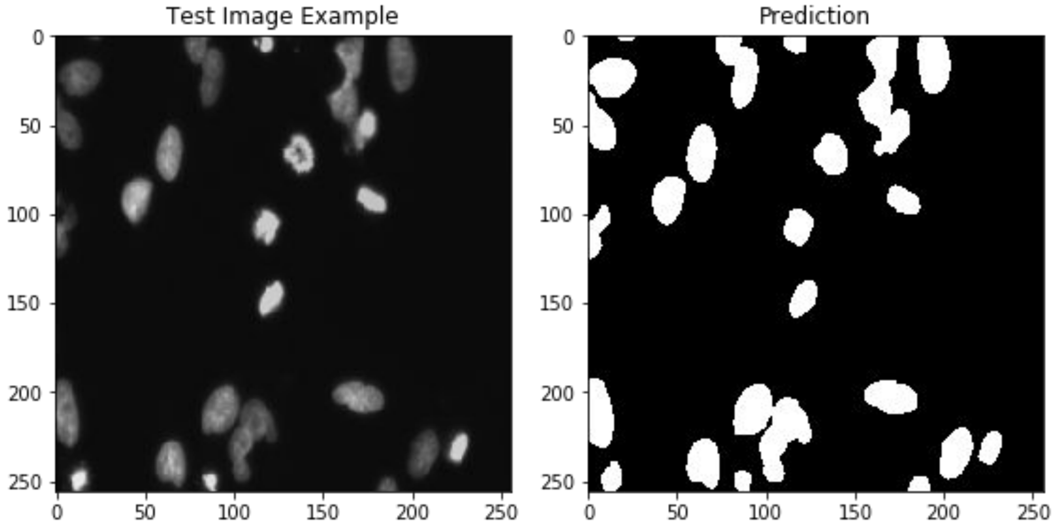
In [9]:

```python
def get_unet(IMG_WIDTH=256,IMG_HEIGHT=256,IMG_CHANNELS=3):
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    s = Lambda(lambda x: x / 255) (inputs)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (s)
    c1 = Dropout(0.1) (c1)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c1)
    p1 = MaxPooling2D((2, 2)) (c1)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (p1)
    c2 = Dropout(0.1) (c2)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c2)
    p2 = MaxPooling2D((2, 2)) (c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (p2)
    c3 = Dropout(0.2) (c3)
    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c3)
    p3 = MaxPooling2D((2, 2)) (c3)

    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (p3)
    c4 = Dropout(0.2) (c4)
    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c4)
    p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (p4)
    c5 = Dropout(0.3) (c5)
    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c5)

    u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (u6)
    c6 = Dropout(0.2) (c6)
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c6)

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

    u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (u8)
    c8 = Dropout(0.1) (c8)
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c8)

    u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (u9)
    c9 = Dropout(0.1) (c9)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same') (c9)

    outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)

    model = Model(inputs=[inputs], outputs=[outputs])
    model.compile(optimizer='adam',loss='binary_crossentropy', metrics=[dice_coef])
    return model
```

Results

Dice coefficient on training data : 0.86

Dice coefficient on validation data : 0.86



Potential Improvements

- Generate more synthetic data
- Make model more generalized
- Try Mask R-CNN as this was used by the winning team
- Try ensemble approach