WHITE BLOOD CELLS SEGMENTATION

Domain Background

White blood cells (also known as WBC or leukocytes) help our body fight infections by attacking bacteria, viruses and other germs that invade the body. A count of leukocytes can help reveal several hidden and undiagnosed diseases. In a manual microscopic review of blood samples, pathologists minutely examine the count and morphology (i.e. size and shape) of white blood cells.

Red blood cells (or RBC, or erythrocytes) are the most common type of blood cells, and they outnumber WBCs by about 600:1. So, in an image of a blood sample, you will see mostly RBCs, with a few WBCs thrown in here and there.

The number of leukocytes in the blood is often an indicator of disease, and thus the WBC count is an important subset of the complete blood count. The normal white cell count is usually between 4 × 10^9/L and 11 × 10^9/L. In the US this is usually expressed as 4,000 to 11,000 white blood cells per microliter of blood. They make up approximately 1% of the total blood volume in a healthy adult, making them substantially less numerous than the RBCs at 40% to 45%. However, this 1% of the blood makes a large difference to health, because immunity depends on it. An increase in the number of leukocytes over the upper limits is called leukocytosis.

In manual process pathologists analyze the blood sample and count the WBC (white blood cells), but this is not accurately defined the correct count and related disease, they use pre-defined approach to determine the health of the person. But if we use Image segmentation using deep learning supervised algorithm model, it accurately demarcates the boundary of WBC even when they are touching each other and identify correct count. This will improve the accuracy and speed of testing and yield better results.

I hope by using state of the art deep learning model for this image segmentation task will improve the accuracy of results and correctly identify the health of a person.

Problem Statement

Develop an efficient Deep Learning model using CNN (Convolutional Neural Networks) to accurately demarcate the boundary of white blood cells in microscopic images of blood

Datasets and Inputs

SigTuple(https://sigtuple.com/) is AI based startup from India builds intelligent screening solutions to aid diagnosis through AI-powered analysis of visual medical data.

As part of hiring AI engineers for their team, they released a medical image data set on Hackerearth and conducted a hackathon. Please see the reference section on below for the data and hackathon links. I thought this is a good research problem and have a scope to select as a capstone for this Nano degree.

Structure of the data set:

```
├── Test_Data (61 files)
|     ├── 017532875DDF.jpg
|     ├── 029E137BB177.jpg
|     ├── 029E137BB179.jpg
|     ├── ...
└── Train_Data (328 files)
      ├── train-0.jpg
      ├── train-0-mask.jpg
      ├── train-100.jpg
      ├── train-100-mask.jpg
      ├── ...
```

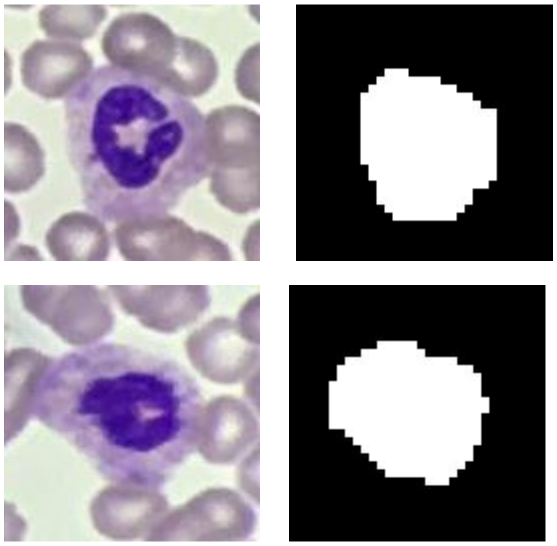
The training set consists of 164 (128X128) patches showing WBCs, and the area has been demarcated in a mask file. The cell at the center of these patches are WBCs, while those surrounding the WBC are RBCs. The files are named like train-0.jpg, train-1.jpg, ... The corresponding mask files are named train-0-mask.jpg, train-1-mask.jpg, ..., respectively. There are also around 5 larger images (and corresponding masks) of blood, showing one or more WBCs in the image.

The test set will consist of larger images of blood smears, from which we need to demarcate the WBC boundaries. My code has to both locate the WBCs and demarcate their boundary.

Solution Statement

For this problem I am going to use Instance based segmentation also called Simultaneous Detection and Segmentation(SDS) method for given Medical image data.

In this method we will use deep convolutional neural network model to classify each pixel to pre-defined classes. And segment each instance with specific class. In our case the classes are White Blood Cells and Red Blood Cells.



From the above pictures, the left side is train image with both WBCs (Blue region) and RBCs and right side is corresponding masked image with demarcated WBC boundaries. So, our model has to locate the WBCs and demarcate their boundary.

Convolutional neural networks will yield state of the art results on image processing like object detection, classification and segmentation.

For this problem I am using Convolutional neural networks for identifying WBCs boundary.

Convolutional neural network is a combination of convolutional map, filters, max pooling layers with activation and dropout functions.

For a given input image having dimensions (1,128,128,3) -> (number_of_samples, img_rows, img_cols, color_channels)

- First we need to define filters to run through entire image for capturing different features of image like edges, shapes etc.
- each filter slide through entire image with activation function like ReLU (rectified linear unit) produce a convolutional map
- Again we can apply same filters to better capture different features and produce an another convolutional map
- Although the role of the convolutional layer is to detect local conjunctions of features from the previous layer, the role of the pooling layer is to merge semantically similar features into one. Because the relative positions of the features forming a motif can vary somewhat, reliably detecting the motif can be done by coarse-graining the position of each feature. A typical pooling unit computes the maximum of a local patch of units in one feature map (or in a few feature maps). Neighboring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and

distortions. Two or three stages of convolution, non-linearity and pooling are stacked, followed by more convolutional and fully-connected layers.

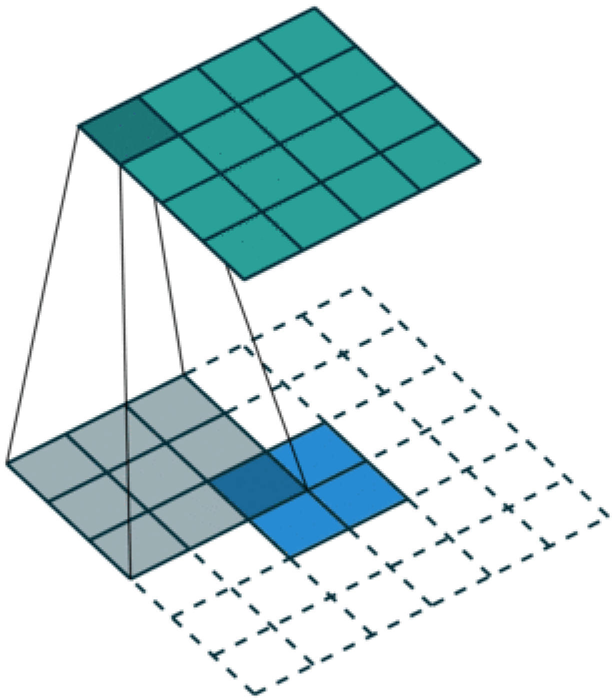
If we apply convolutional and pooling operations sequentially on the given image, the image size will decrease deep down and it will not return correct masked area of WBCs. So in order to keep the image as equal to the input shape. We are going to apply de-convolution on convolutional feature maps to restore the image to original image resolution and correctly identify the demarcated area of white blood cells.

Benchmark Model

There is large consent that successful training of deep neural net-works requires many thousand annotated training samples. But in our case thousands of training images are usually beyond reach in biomedical task.

I am using modified U-net model and training strategy that relies on the strong use of data augmentation to use the available annotated samples more efficiently. The architecture consists of a contracting path(Convolutions) to capture context and a symmetric expanding path(Deconvolution) that enables precise localization.

Deconvolution layer is a very unfortunate name and should rather be called a transposed convolutional layer.



Visually, for a transposed convolution with stride one and no padding, we just pad the original input (blue entries) with zeroes (white entries) (see above image).

For this problem who need to apply Instance based segmentation approach also called simultaneous detection and segmentation. If you use normal convolution approach we need more processing and training time for object detection and a separate method to segment each instance but if you use modified U-net model, you can run your input end to end at a time and there is no separate processing for identification and segmentation of instances.

We can apply dropout to last convolution to prevent overfitting. Most of the time dropout is applied to fully connected layers compared to convolutional layers. since the convolutional layers don't have a lot of parameters, overfitting is not a problem and therefore dropout would not have much effect. However, the additional gain in performance obtained by adding dropout in the convolutional layers (3.02% to 2.55%) is worth noting. Dropout in the lower layers still helps because it provides noisy inputs for the higher fully connected layers which prevents them from overfitting.

The modified model follows the same pattern of U-net but with adding drop-out and augmenting on rotation and flipping original images. So that the model able to see more data and well generalize and invariance to distortions in images.

The model consists of a contracting path (left side) and an expansive path (right side).

The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for down sampling. At each down sampling step, we double the number of feature channels.

Every step in the expansive path consists of an up sampling of the feature map followed by a 2x2 convolution (\up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution.

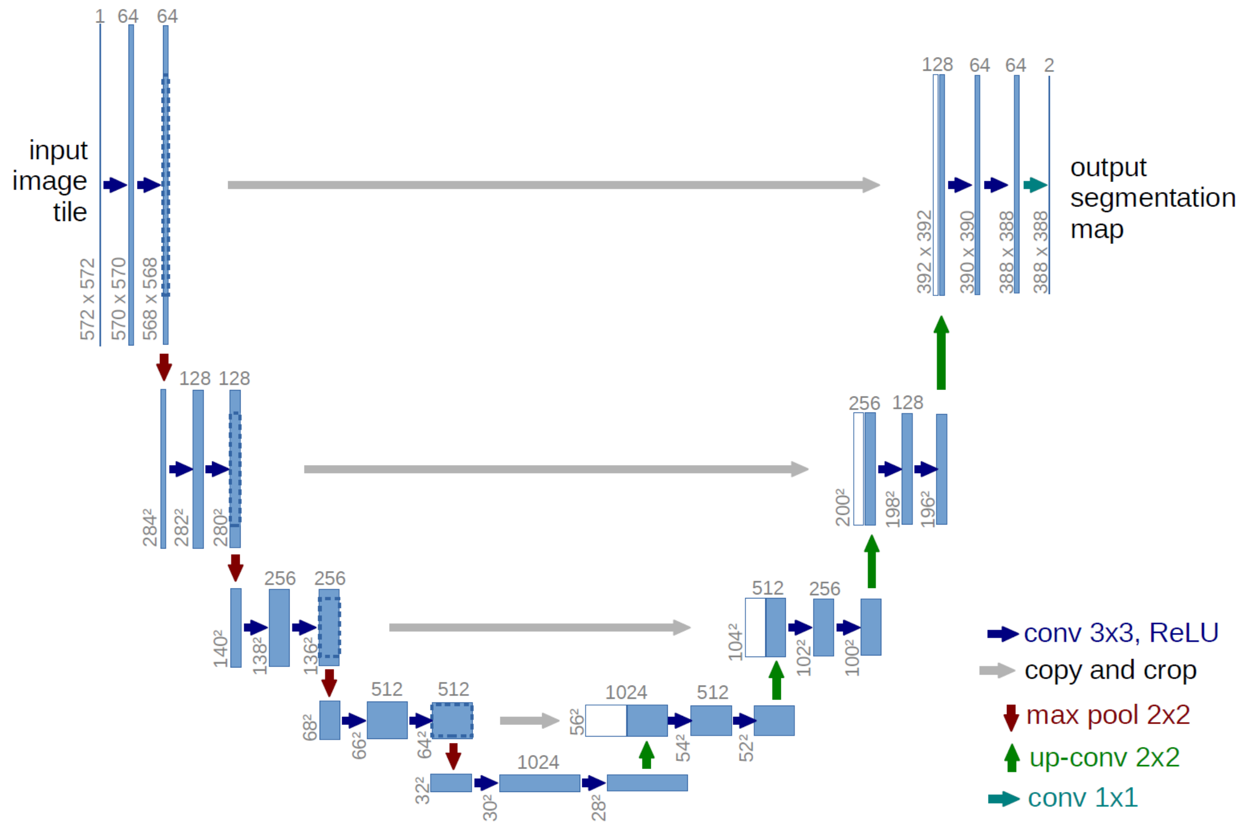
Before classifying final vector to desired classes, we can apply dropout to reduce overfitting.

Architecture inspired from U-net:

```
inputs = Input((img_rows, img_cols, 1))

  conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)

  conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)

  pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
```

```
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool3)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv5)

    up6 = concatenate([UpSampling2D(256, (2, 2), strides=(2, 2), padding='same')(conv5), conv4],
axis=3)
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(up6)
    conv6 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv6)

    up7 = concatenate([UpSampling2D(128, (2, 2), strides=(2, 2), padding='same')(conv6), conv3],
axis=3)
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(up7)
    conv7 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv7)

    up8 = concatenate([UpSampling2D(64, (2, 2), strides=(2, 2), padding='same')(conv7), conv2],
axis=3)
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(up8)
    conv8 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv8)


    up9 = concatenate([UpSampling2D(32, (2, 2), strides=(2, 2), padding='same')(conv8), conv1],
axis=3)

    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(up9)

    conv9 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv9)

    dropout1 = Dropout(0.25)(conv9)


    conv10 = Conv2D(1, (1, 1), activation='sigmoid')(dropout1)


    model = Model(inputs=[inputs], outputs=[conv10])


    return model
```

U-net architecture:



I would like to discuss few other supervised image classification algorithms with comparison with convolutional neural networks.

One of the best supervised classification method is Support Vector Machines (Generalization of support vector classifier). Support vector classifier is generalization version of max margin classifier.

Based on the given data and problem set we will go with either convolutional or SVM. If the data is not linearly separable and it is a general classification problem, then using SVM will give good results. SVM represent the data in high dimensional with kernel trick. The method will yield good results without overfitting the data set with less parameters and less memory constraint with support vectors. SVM will give state of the art results on general classification problems like classifying a given data point to any one of given binary or multi class labels. But it cannot extract the useful features from the given data like CNN. Even though by using k-fold cross validation with grid or randomized search, we still need to manually set the parameters (Kernel, C, Y values) to yield good results. But in case of CNN the model will automatically learn and identify key features in the image using convolution maps with stochastic gradient descent weight update algorithm.

SVM will yield global optimum. But with CNN or neural networks, there is a possibility of local minimum and would not get a big picture on complete data. We can use any one of momentum techniques to get over on local minima.

In our case we have demarcate the boundary of WBC from given image. We need to learn the features of image and understand what is mask and how mask is calculating when model is fitted with training data. CNN will automatically learn the different features and update their weights until gives a low error score.

We can fit linear SVM in place of final fully connected layers in CNN model instead of softmax classifier for multi label data. But with good understanding of data and features softmax out performs svm by giving probabilities to each data point for multi class.

So in conclusion based on the problem set and context of the problem we will go with different classification techniques like SVM, RandomForest, and Feed Forward Neural networks or CNN. In our case convolutional neural networks will give accurate results and correctly identify the boundary of white blood cells in images even though the cells are touching with each other for counting purpose.

The original U-net model have achieved 77.5% accuracy on DIC-HeLa data set from ISBI cell tracking challenge. I am taking this is as a benchmark and try to achieve more accuracy with modified U-net architecture.

Evaluation Metrics

We use Dice coefficient or F1 score for evaluation of the model on predicted masks with ground truth values.

The Dice score is often used to quantify the performance of image segmentation methods. There you annotate some ground truth region in your image and then make an automated algorithm to do it. You validate the algorithm by calculating the Dice score, which is a measure of how similar the objects are. So it is the size of the overlap of the two segmentations divided by the total size of the two objects. Using the same terms as describing accuracy, the Dice score is:

Dice score=number of true positives/number of positives + number of false positives

So the number of true positives, is the number that your method finds, the number of positives is the total number of positives that can be found and the number of false positives is the number of points that are negative that your method classifies as positive.

The Dice score is not only a measure of how many positives you find, but it also penalizes for the false positives that the method finds, similar to precision. so it is more similar to precision than accuracy. The only difference is the denominator, where you have the total number of positives instead of only the positives that the method finds. So the Dice score is also penalizing for the positives that your algorithm/method could not find.

In the case of image segmentation, let's say that you have a mask with ground truth, let's call the mask A like you suggest. So the mask has values 1 in the pixels where there is something you are trying to find and else zero. Now you have an algorithm to generate image/mask B, which also has to be a binary image, i.e. you create a mask for your segmentation. Then we have the following:

Number of positives is the total number of pixels that have intensity 1 in image A

Number of true positives is the total number of pixels which have the value 1 in both A and B. So it the intersection of the regions of ones in A and B. It is the same as using the AND operator on A and B.

Number of false positives is the number of pixels which appear as 1 in B but zero in A.

Project Design

The input images and their corresponding segmentation maps are used to train the network with the stochastic gradient descent implementation of Keras.

The solution consists of three steps:

1. Data Augmentation
   Data augmentation is essential to teach the network the desired invariance and robustness properties, when only few training samples are available. In case of microscopically images we primarily need shift and rotation invariance as well as robustness to deformations and gray value variations. Especially random elastic deformations of the training samples seem to be the key concept to train a segmentation network with very few annotated images.
   We can use Keras ImageDataGenerator api for augmentation of the image data.

2. Train a Model
   We can use Keras Conv2D api to create deep convolutional U-net model and train the model on our image data. We can use Aws Gpu for speed up the training time.
   For Ex:
   U_net.compile(optimizer=Adam(lr=1e-5), loss=dice_coef_loss, metrics=[dice_coef])

3. Testing on test set
   We can test the trained model on our test data and output the masked images. And evaluate the model using the dice coefficient.

References:

Domain background: https://en.wikipedia.org/wiki/White_blood_cell

Data sets and Inputs:

Hackathon: https://www.hackerearth.com/challenge/competitive/sigtuple-ai-challenge/machine-learning/wbc-segmentation/

Data set link: https://s3-ap-southeast-1.amazonaws.com/he-public-data/contests/SigTuple_data.tar

Benchmark Model: https://arxiv.org/abs/1505.04597

Evaluation Metrics: https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient

Data Augmentation and Technologies: https://keras.io/

Cloud computing and GPU: https://aws.amazon.com/