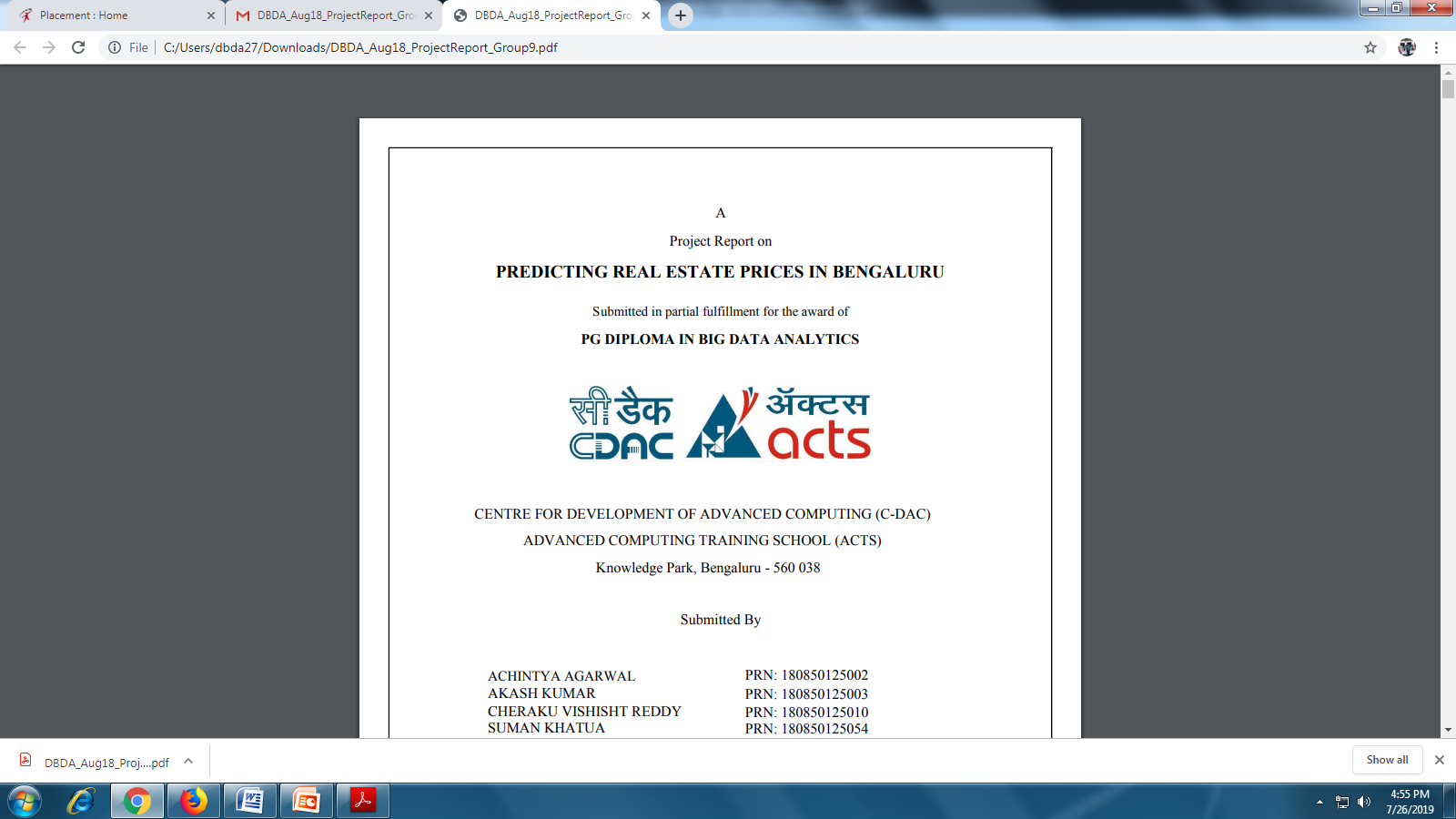
A

**Project Report on**

**WHITE BLOOD CELLS SEGMENTATION**

Submitted in partial fulfillment for the award of

**PG DIPLOMA IN BIG DATA ANALYTICS**

****

**Centre for Development of Advanced Computing**

**Knowledge Park**

Bangalore - 560 038  
Karnataka (India)

Submitted By

CH.M. PHANI DUTT PRN: 190250125011

G.NAGARAJU PRN: 190250125016 NEELANSHU PRN: 190250125033

SREELAL.B PRN: 190250125052

Guided by

Dr R.C.Saritha

Joint Director, C-DAC, KP,

Bangalore centre

C-DAC LOGO

**CERTIFICATE**

**WHITE BLOOD CELLS SEGMENTATION**

Submitted By

CH.M. PHANI DUTT PRN: 190250125011

G.NAGARAJU PRN: 190250125016 NEELANSHU PRN: 190250125033

SREELAL B PRN: 190250125052

This is to certify that, the project report entitled is the record of bonafide work carried out by them in partial fulfillment of the requirement for the award of **PG Diploma in Big Data Analytics** prescribed by **Centre for Development of Advanced Computing (C-DAC).**

Dr R.C.Saritha Ms. Priyanka Sharma Ms. M.Savithri

(Project Guide) (Project Co-ordinator) (Course Co-ordinator)

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to all the people without whom this project would have been highly impossible.

We would like to devote our first vote of thanks to our guide **Dr R.C.Saritha** for his constant support and encouragement. She has a great hand in the firm foundation of this project. We are deeply in debt for her valuable suggestions, scholarly guidance and constructive criticisms along with constant encouragement at each and every step for successful completion of the project.

We would also like to thank our Project Co-ordinator **Ms. Priyanka Sharma** for inspiring us towards completion of this project.

Last but not the least we would like to thank all those who assisted us directly or indirectly for their valuable time and help.

Ch.M.Phani Dutt

G.Nagaraju

Neelanshu

Sreelal B

**ABSTRACT**

The traditional approach to count white blood cells is to take a sample of blood which is placed under a microscope and a pathologist manually counts the number of cells in each frame. The total count is then extrapolated by assuming that the distribution is uniform across the entire blood sample and multiplying up.

Blood cell segmentation is a critical innovation for automatic differential blood counting, classification and analysis in clinical examination. In color blood cell images segmentation and recognition are two essential issues in the field of biomedical cell morphology. This paper approaches methods to segment the blood cells from microscopic thin blood images. This data is the premise to perform higher level tasks for example, automatic differential blood counting, and detection of different diseases.

There is a necessity of an automated, real-time, computer vision-based cell segmentation system. We developed an efficient model for segmenting White Blood Cells (WBCs) from input images of blood cells. Input images are captured by connecting digital camera to microscope. Captured images are enhanced and segmented. Modified UNet architecture was used to train the model using CNN.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Description** | **Page No.** |
| 1 | Introduction | 6-8 |
| 2 | Literature Survey | 9 |
| 3 | Dataset | 10 |
| 4 | Methodology | 11-17 |
| 5 | Project Code | 18-27 |
| 6 | Result | 28 |
| 7 | Future Scope | 29 |
| 8 | Conclusion | 30 |
| 9 | Bibliography | 31 |

**Introduction**:

White Blood cell composition reveals important diagnostic information about the patients as well as patient follow-up. One of the important counts required by pathologists is differential blood count in which an expert counts 100 white blood cells on the smear at hand and computes the percentage of occurrence of each type of cell counted. The result gives important information about patient’s health status and plays an important role in diagnosis. Counting each classes of the white blood cell manually is a tedious, time consuming and qualitative process.

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. See Figure 1.



Figure 1. White blood cells in blood sample

We have developed deep learning model for this image segmentation task which will improve the accuracy of results and correctly identify the health of a person.

**Problem Statement**

The goal is to develop an efficient Deep Learning model using CNN (Convolution Neural Networks) to accurately demarcate the boundary of white blood cells in microscopic images of blood.

**Technique**

From the given data set of images, we have to 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. 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.

Convolution neural networks will yield results on image processing like object detection, classification and segmentation. Convolution neural network is a combination of convolution map, filters, max pooling layers with activation and dropout functions. First we need to define filters to run through entire image for capturing different features of image like edges, shapes etc. Each filter slides through entire image with activation function like ReLU(rectified linear unit) produce a convolutional map. The activation function can act as a non-linear decision function to allow the input to pass through the neuron or not. If it passes it will return the value of the input otherwise return 0 in case of ReLU. Again we can apply same filters to better capture different features and produce another convolutional map.

A typical **maximum pooling** (Max Pooling layer) 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 invariance to small shifts and distortions. Deep neural nets with a large number of parameters are very powerful machine learning systems. However, over fitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with over fitting by combining the predictions of many different large neural nets at test time.

**Dropout** is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co adapting too much. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single un-thinned network that has smaller weights. This significantly reduces over fitting and gives major improvements over other regularization methods. Dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the art results on many benchmark data sets. Two or three stages of convolution, non-linearity and pooling with dropout are stacked, followed by more convolution and fully-connected layers. We have to run images through this stack of layers multiple times to capture the useful features like object detection and segmentation. Each run you need to update the weights of the network to reduce the loss to accurately identify or segmentation.

The process of updating weights with activation function and computing loss is called back propagation. The main goal is reach global minima (low error) with continuous updating of weights in each run. This is called optimization of the network for reaching low error rate. The rate at which each weight is updated is called learning rate. But there are possibilities you reach local minima instead of global minima. At that time, you need some momentum to cross local minima to reach global minima. For this we will use different type of momentum techniques like Adam etc. If we apply convolution 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. In order to keep the image as equal to the input shape, we are going to apply de-convolution on convolution feature maps to restore the image to original image resolution and correctly identify the demarcated area of white blood cells. We are using modified U-net model and training strategy to get efficient results.

**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 definition of true positive (TP), false positive (FP), and false negative (FN), it can be written as

If the dice score is high, the model is correctly segmenting given images with right classes.

**Literature Survey:**

In the Paper (S.S.Savkare, 2015), the authors proposed a model to detect the blood cells using the microscopic images using the traditional method of using K-means clustering. This paper uses the median filter as noise reduction filter. And the method was implemented on thin blood images which consisting of better stained images. But for images with low contrast the K-means clustering does not work well. The images are taken directly from a laboratory and were preprocessed and the accuracy has been measured using the K-means clustering.

In the paper (\*P.Yampri, 2006), the authors used Eigen cell and Parametric feature detection which was the morphological features of the nucleus. They have used principal component analysis and classified the white blood cell images into 2 groups based on number of nucleus lobes which are one and greater than 1. The proposed model shows some accuracy with their tested images but in some cases it has failed.

In the paper (M.Saritha, 2016), the authors detected the blood cancer using the Microscopic images from human blood samples. In which the cancer detection be based on the count of number of white blood cells present. The authors used geometrical features like Radius, perimeter of the nucleus as part of their feature extraction. And have calculated the accuracy on the test images.

**Dataset:**

The dataset for this project originates from the SigTuple AI Challenge. SigTuple released this data set.

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 cells 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 consists of 61 larger images of blood smears of multiple dimensions, with three channels (R, G, B) from which we need to demarcate the WBC boundaries. Both training and test data set have three colored channel images. The color information is not necessary because we only need the gray scale images that can discriminate the white blood cells from the remaining cells (red blood cells). There are far more pixels in our dataset that belong to red blood cells than there are pixels that belong to white blood cells. To compensate this class imbalance, we have used dice score as loss metric to predict the segmented masks.

**Methodology**

**Flow Chart of Methodology:**

Considering the problem statement from Literature work

Datasets considered were available to public

Preprocessing of Datasets of Microscopic Images

Training of Model (CNN)(Implementation)

Tuning of hyper parameters of Model

Prediction of Results using the test data

Validation of the Model

From the flow chart first two steps are already discussed from the Introduction and Literature survey. The third step Data preprocessing is discussed below:

**Data Preprocessing:**

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as preprocessing. Fortunately, for this dataset, there are no invalid data we must deal with however there are 5 large images (corresponding masks) of blood. So we need to re-size these images to 128 x 128 without losing the pixel information from both train and mask data and store it in new folder. So that all the images (both train and mask) are in same dimension and pass to our deep learning algorithm for training. The original and the mask images are shown below:

As the data set available is having less number of images to train the Deep learning Models, we have used the method of Image Augmentation using the ImageDataGenerator from the keras library. For each and every image we made by using augmentation, we have created the corresponding mask image also using the skimage module by taking the region props of that particular nucleus cell.

Then the training of the Model and Implementation has been done and it is discussed below:

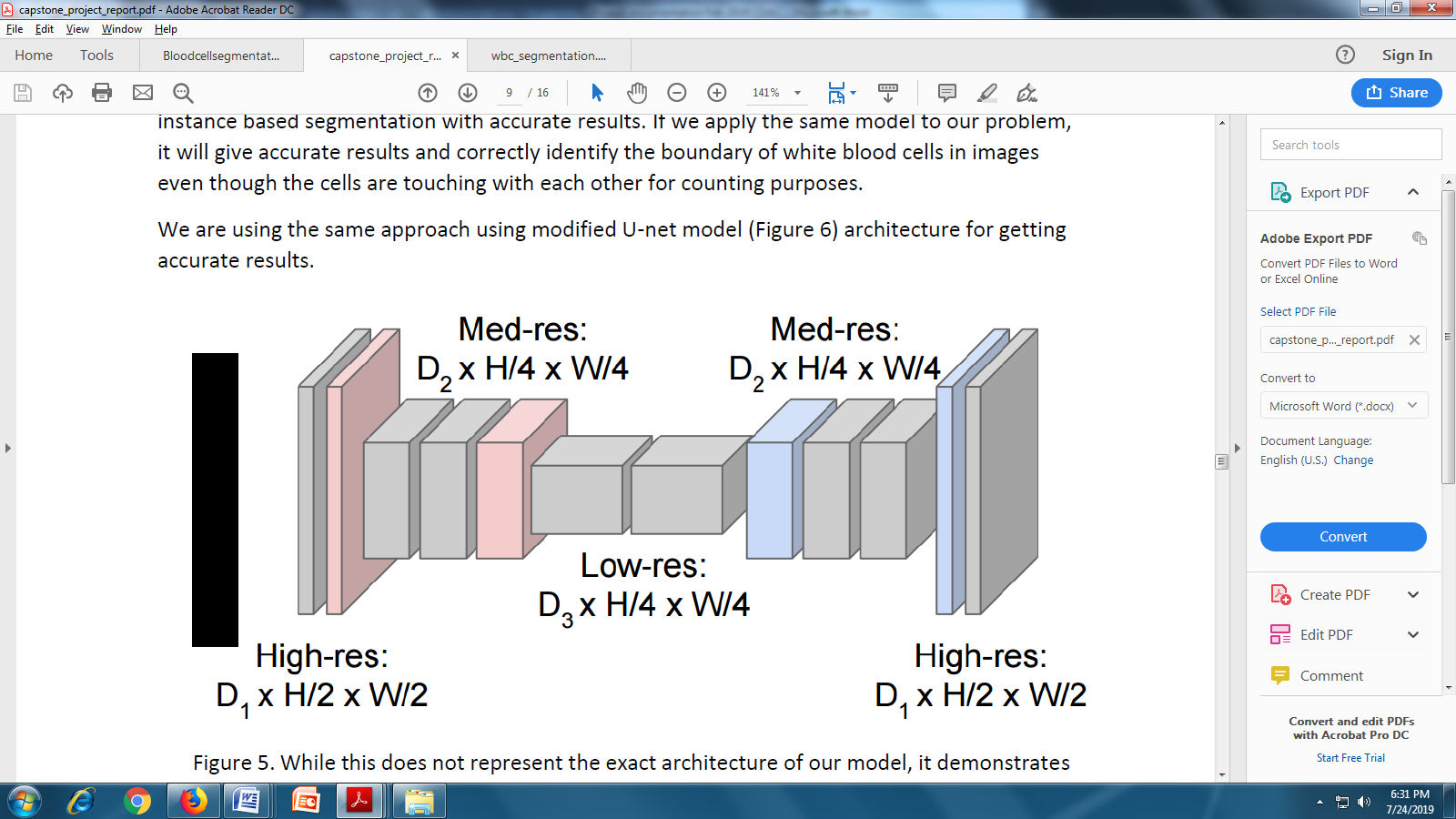
**Implementation:**

From the given training data set (SigTuple\_data/Train\_Data) the train images (train-0.jpg, train- 1.jpg, train-2.jpg.. etc.) can act as a training set and mask images (train-0-mask.jpg, train-1- mask.jpg, train-2-mask.jpg.. etc.) as dependent to fit to our train images. We can create these two sets as X\_train and Y\_train and pass these for training to our model. Here we are using tensorflow as a back end for our Keras deep learning model.

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape as (nb\_samples, rows, columns, channels) where nb\_samples correspond to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

For this problem that needs to apply Instance based segmentation approach also called simultaneous detection and segmentation. If we 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 (Olaf Ronneberger, 2015), you can run your input end to end at a time and there is no separate processing for identification and segmentation of instances.

We are using the same approach using modified U-net model (Figure 6) architecture for getting accurate results.



**Figure:5**

While this does not represent the exact architecture of our model, it demonstrates the principles of a down-sampling up-sampling convolution network. 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. See Figure 5.

Every step in the expansive path consists of an up sampling of the feature map followed by a 2x2 convolution (“up-sampling”) 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.

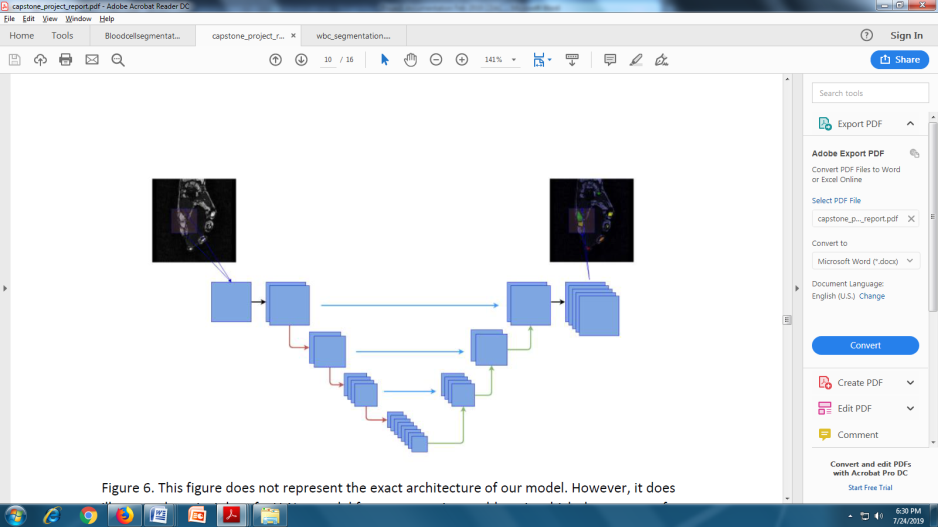


Figure: 6

This figure does not represent the exact architecture of our model. However, it does illustrate the principles of a U-Net model for segmentation problem, in which the outputs of early down-sampling layers are concatenated to those of later up-sampling layers.

We have trained this model with 50 epochs and have batch size of 8 and learning rate 1e-04(0.0001). During training after each conv and pooling layer we have applied dropout layer to reduce overfitting due to size of the data. 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 prevent them from overfitting. Dropout is a float value between 0 and 1. Fraction of the input units to drop. We have used 0.25 dropout value for our model to train the model for efficient results

Keras Model Architecture:

Layer (type) Output Shape Param # Connected to

========================================================================

============================

inputs (InputLayer) (None, 128, 128, 1) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1\_1 (Conv2D) (None, 128, 128, 32) 320 inputs[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1\_2 (Conv2D) (None, 128, 128, 32) 9248 conv1\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pool\_1 (MaxPooling2D) (None, 64, 64, 32) 0 conv1\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 64, 64, 32) 0 pool\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2\_1 (Conv2D) (None, 64, 64, 64) 18496 dropout\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2\_2 (Conv2D) (None, 64, 64, 64) 36928 conv2\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pool\_2 (MaxPooling2D) (None, 32, 32, 64) 0 conv2\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 32, 32, 64) 0 pool\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv3\_1 (Conv2D) (None, 32, 32, 128) 73856 dropout\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv3\_2 (Conv2D) (None, 32, 32, 128) 147584 conv3\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pool\_3 (MaxPooling2D) (None, 16, 16, 128) 0 conv3\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 16, 16, 128) 0 pool\_3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv4\_1 (Conv2D) (None, 16, 16, 256) 295168 dropout\_3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv4\_2 (Conv2D) (None, 16, 16, 256) 590080 conv4\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pool\_4 (MaxPooling2D) (None, 8, 8, 256) 0 conv4\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_4 (Dropout) (None, 8, 8, 256) 0 pool\_4[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv5\_1 (Conv2D) (None, 8, 8, 512) 1180160 dropout\_4[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv5\_2 (Conv2D) (None, 8, 8, 512) 2359808 conv5\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

upsample\_1 (UpSampling2D) (None, 16, 16, 512) 0 conv5\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concat\_1 (Concatenate) (None, 16, 16, 768) 0 upsample\_1[0][0]conv4\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv6\_1 (Conv2D) (None, 16, 16, 256) 1769728 concat\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv6\_2 (Conv2D) (None, 16, 16, 256) 590080 conv6\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_6 (Dropout) (None, 16, 16, 256) 0 conv6\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

upsample\_2 (UpSampling2D) (None, 32, 32, 256) 0 dropout\_6[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concat\_2 (Concatenate) (None, 32, 32, 384) 0 upsample\_2[0][0]conv3\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv7\_1 (Conv2D) (None, 32, 32, 128) 442496 concat\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv7\_2 (Conv2D) (None, 32, 32, 128) 147584 conv7\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_7 (Dropout) (None, 32, 32, 128) 0 conv7\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

upsample\_3 (UpSampling2D) (None, 64, 64, 128) 0 dropout\_7[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concat\_3 (Concatenate) (None, 64, 64, 192) 0 upsample\_3[0][0]conv2\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv8\_1 (Conv2D) (None, 64, 64, 64) 110656 concat\_3[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv8\_2 (Conv2D) (None, 64, 64, 64) 36928 conv8\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_8 (Dropout) (None, 64, 64, 64) 0 conv8\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

upsample\_4 (UpSampling2D) (None, 128, 128, 64) 0 dropout\_8[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concat\_4 (Concatenate) (None, 128, 128, 96) 0 upsample\_4[0][0]conv1\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv9\_1 (Conv2D) (None, 128, 128, 32) 27680 concat\_4[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv9\_2 (Conv2D) (None, 128, 128, 32) 9248 conv9\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_9 (Dropout) (None, 128, 128, 32) 0 conv9\_2[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

outputs (Conv2D) (None, 128, 128, 1) 33 dropout\_9[0][0]

==================================================================

============================

Total params: 7,846,081

Trainable params: 7,846,081

Non-trainable params: 0

**Tuning of Hyper Parameters:**

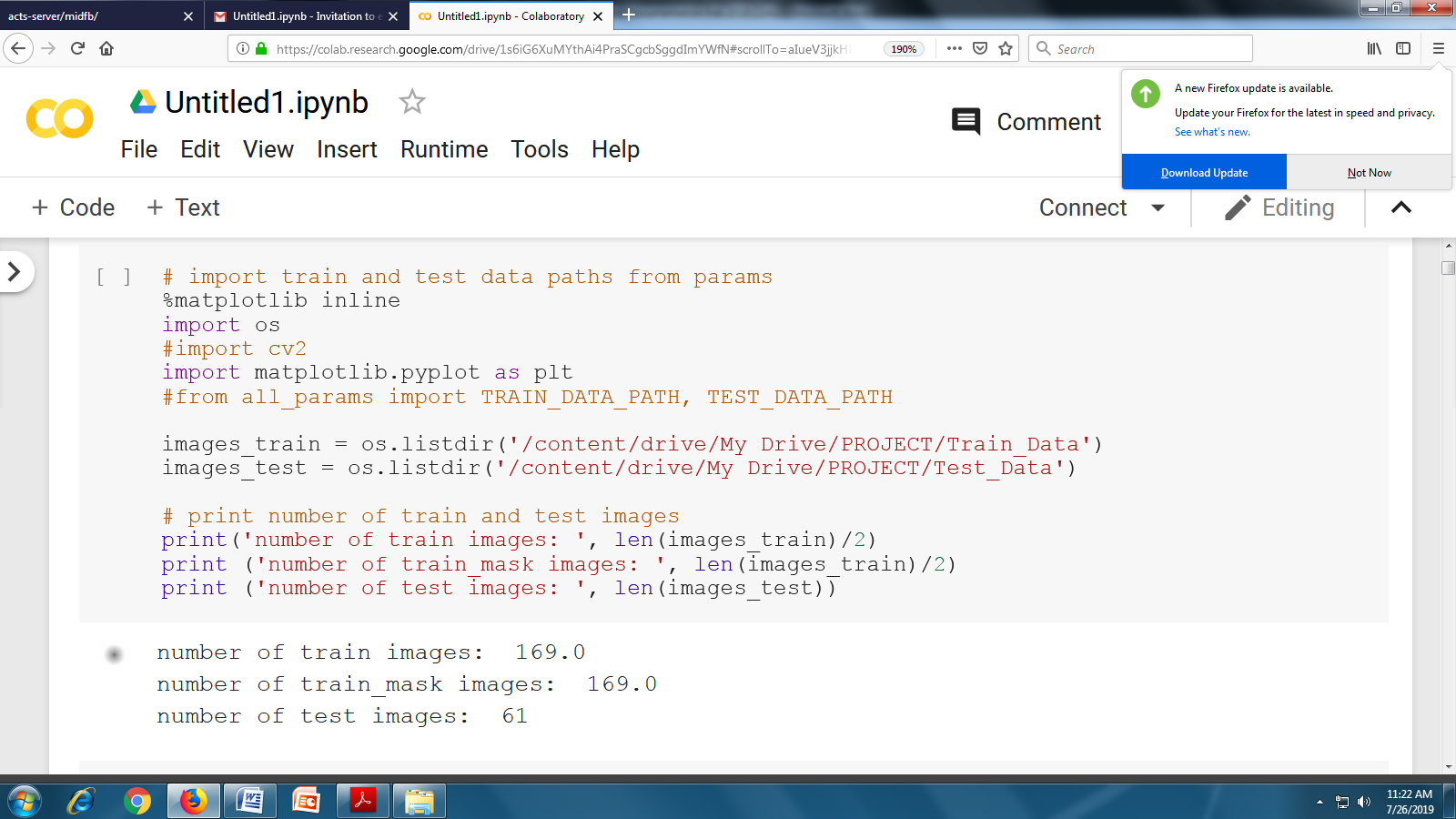
As the CNN model applied using the U-Net model achieved only 77% in one of the Image segmentation challenge mentioned in (Olaf Ronneberger, 2015). To achieve more accurate results on that particular model we have implemented tuning of hyper parameters like batch size etc.,. And the main parameter we have considered is the learning rate. So we have provided the Dynamic learning rate.

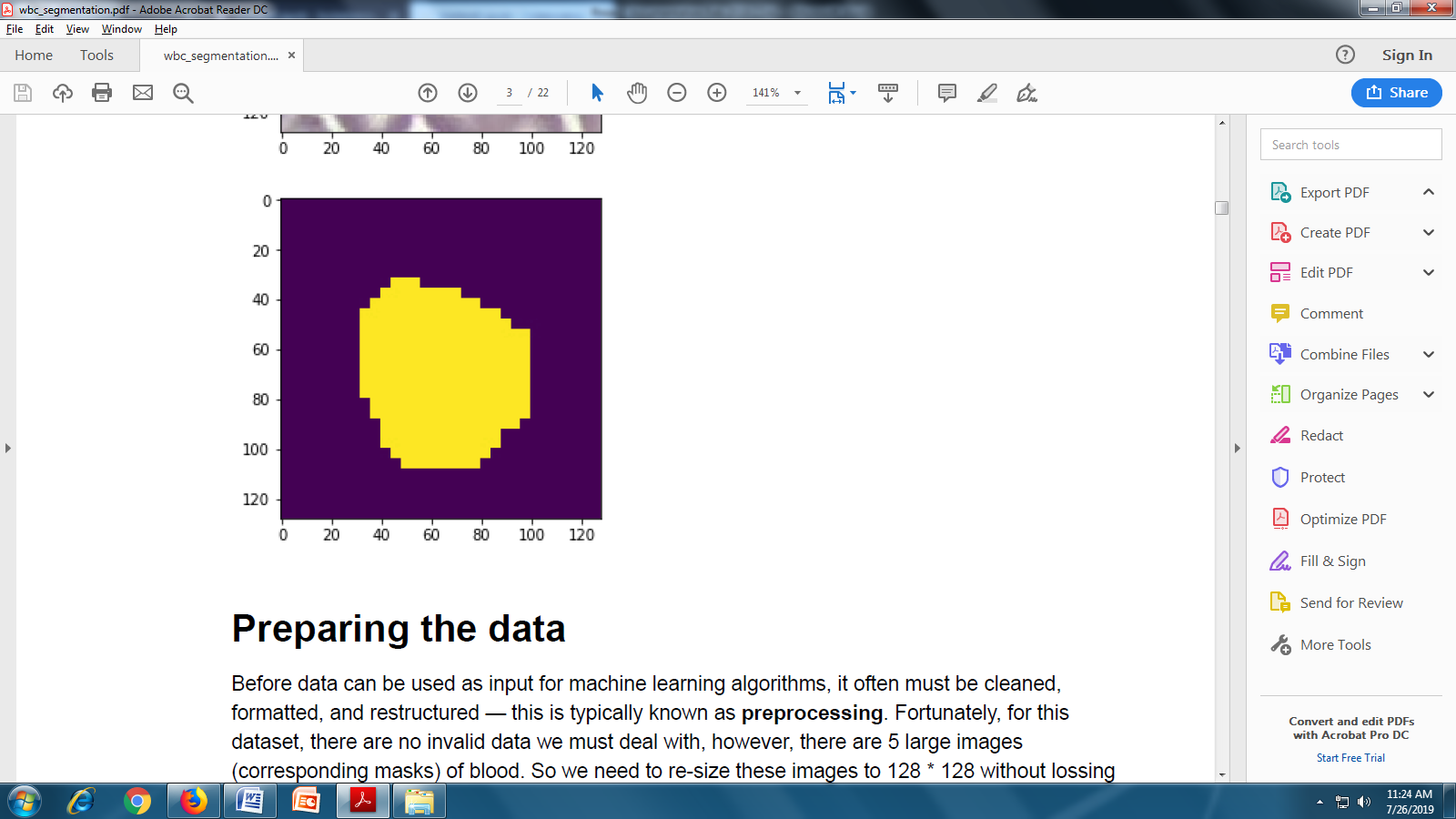
**Dynamic learning rate**: whenever the loss function stopped decreasing, a learning rate drop was added. We have used Keras call back function **ReduceLROnPlateau** for reducing learning rate. This callback monitors loss as a quantity and if no improvement is seen for 10 epochs, the learning rate is reduced by a factor by which the learning rate will be reduced. new\_lr = lr \* factor.

Adding dropout to a layer: dropout randomly drops weights in the layer it’s applied to during training and scales the weights so that the network keeps working during inference. The final Keras model was derived by training in an iterative fashion, adjusting the parameters (e.g. learning rate) has score of 94%.

**WHITE BLOOD CELLS SEGMENTATION**

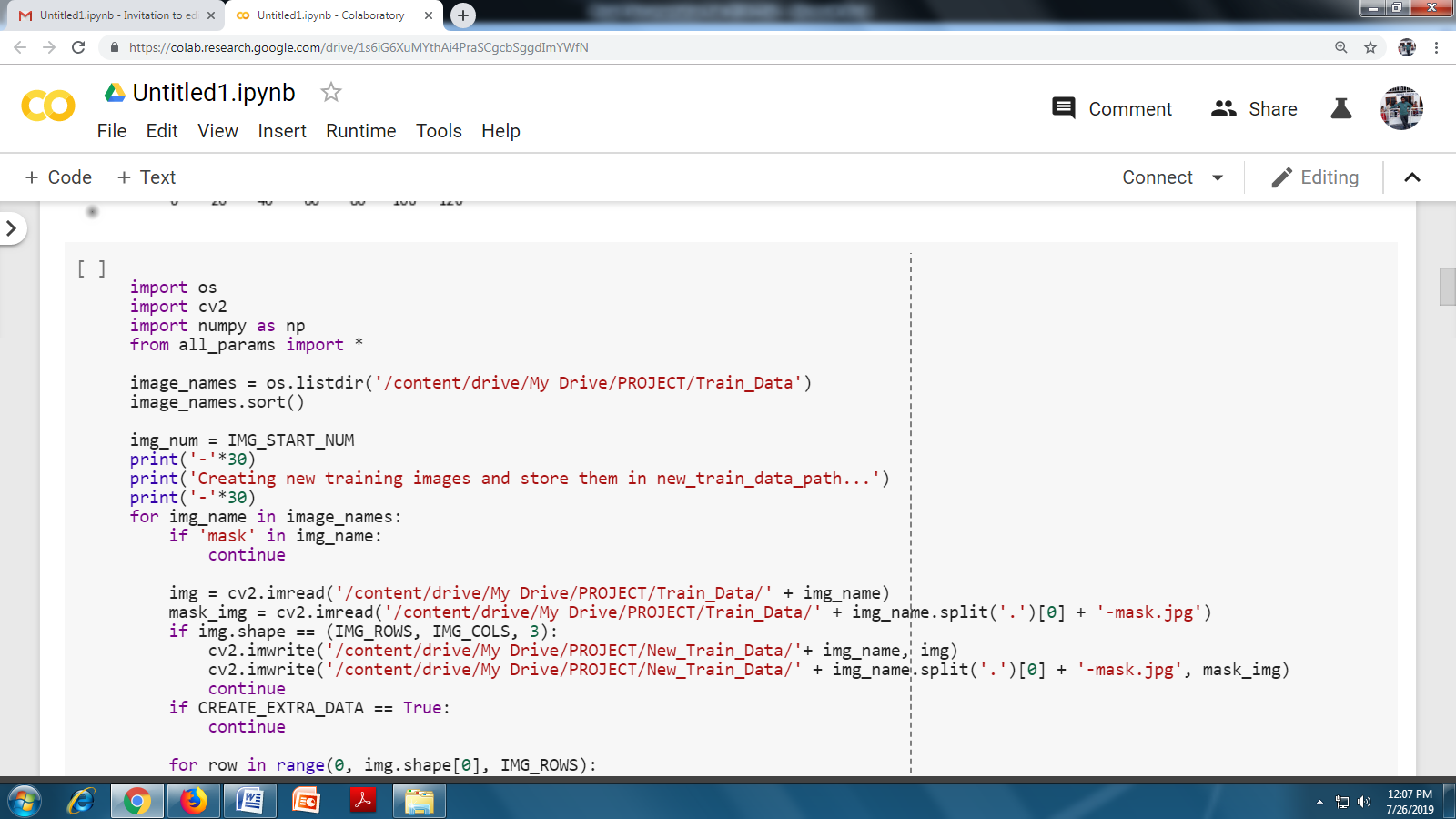
**Project Coding Part**

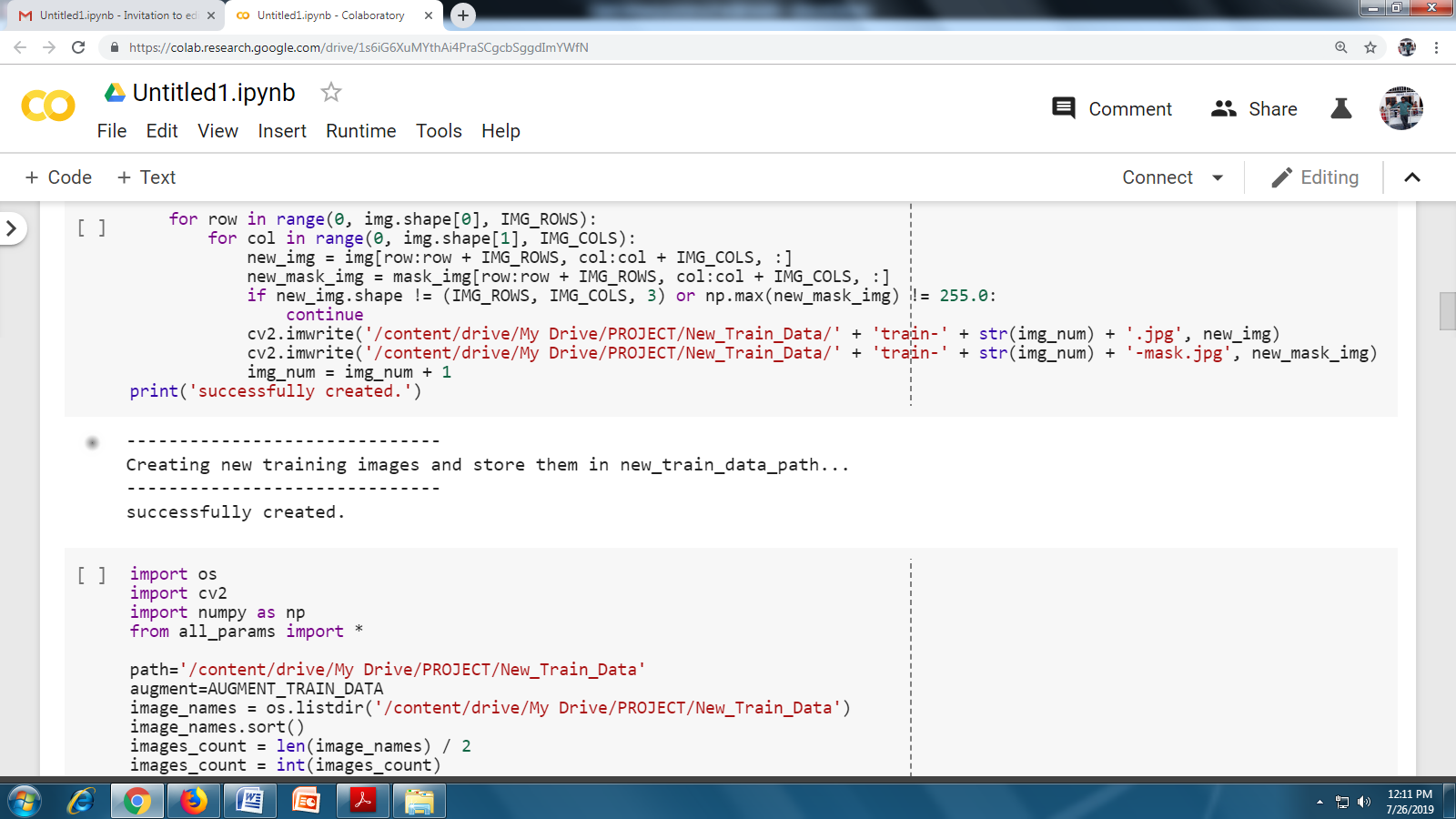


**Preparing the data**

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as preprocessing. Fortunately, for this dataset, there are no invalid data we must deal with; however, there are 5 large images (corresponding masks) of blood. So we need to re-size these images to 128 \* 128 without losing the pixel information from both train and mask data and store it in new folder called new\_train\_data. So that all the images (both train and mask) are in same dimension and pass to our deep learning algorithm for training



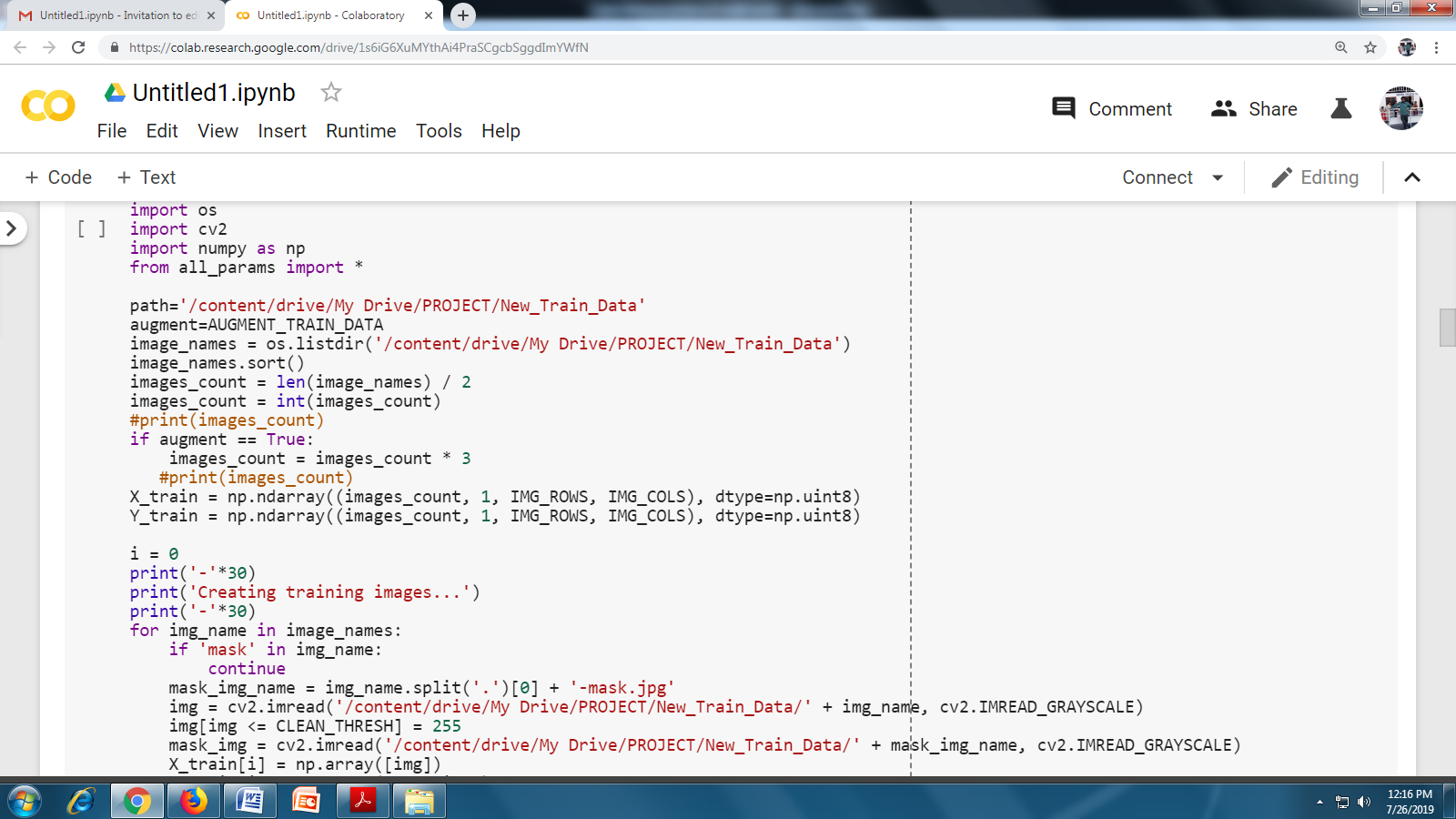


**Create train and test data sets**

From the given training data set(SigTuple\_data/Train\_Data) the train images(train-0.jpg, train-1.jpg,train-2.jpg.. etc) can act as a training set and mask images(train-0-mask.jpg, train-1-mask.jpg, train- 2-mask.jpg.. etc) as dependent to fit to our train images. We can create these two sets as X\_train

and Y\_train and pass these for training to our model. Here we are using tensorflow as a back end for our keras deep learning model.

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape where nb\_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.



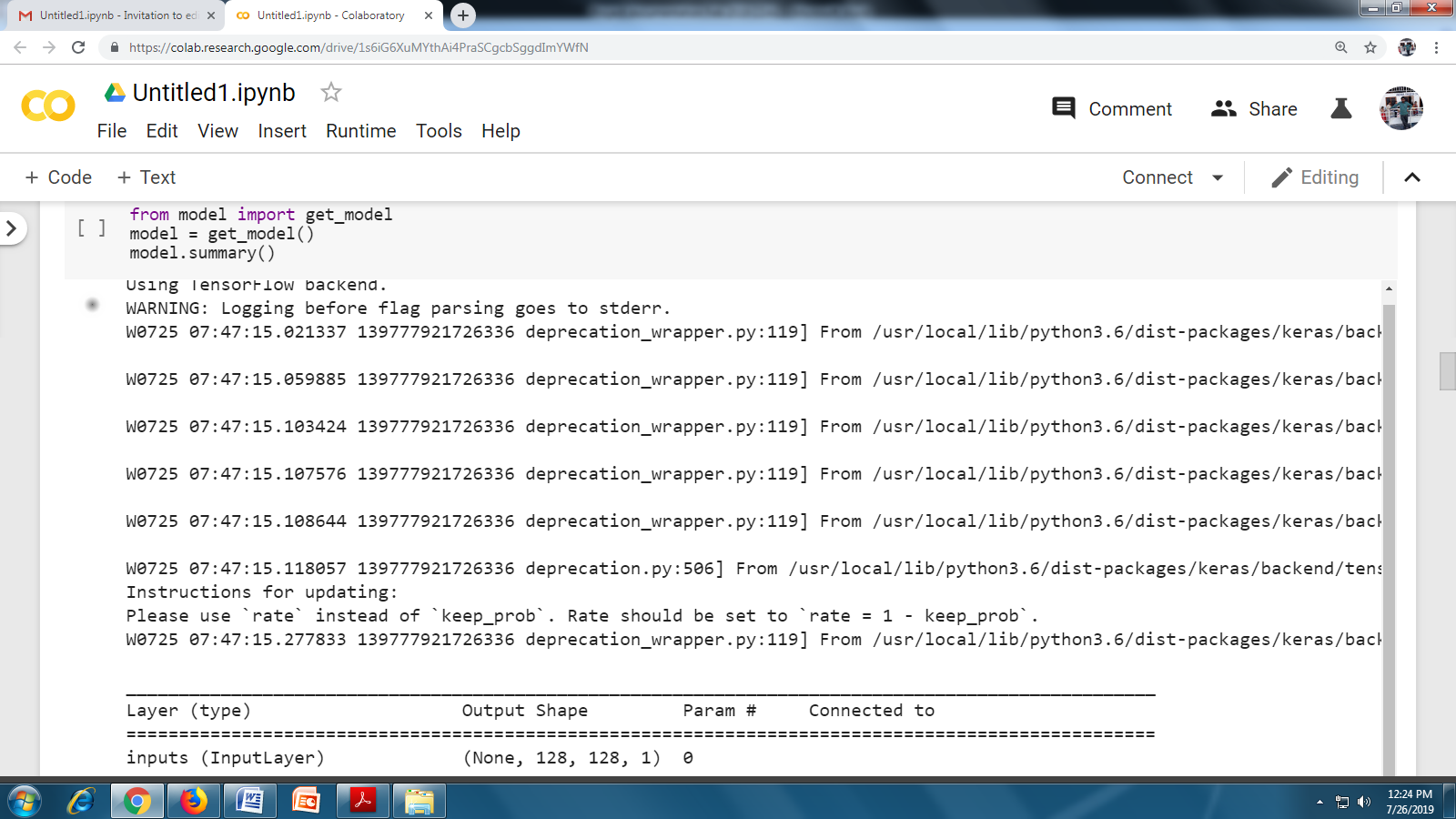


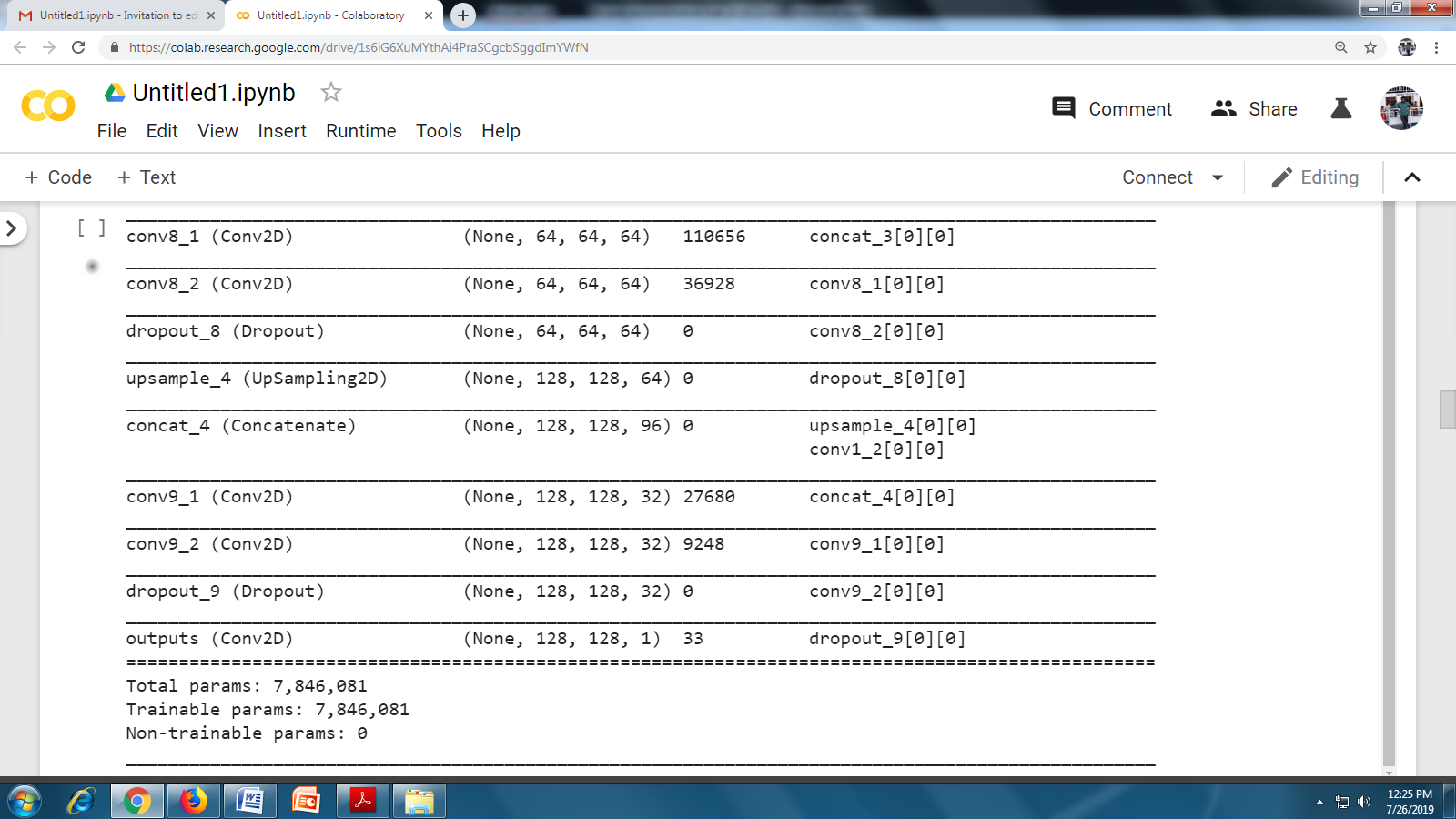
**Create a CNN (Convolutional Neural Networks) model**

For this problem that needs 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 (https://arxiv.org/abs/1505.04597), you can run your input end to end at a time and there is no separate processing for identification and segmentation of instances.

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.

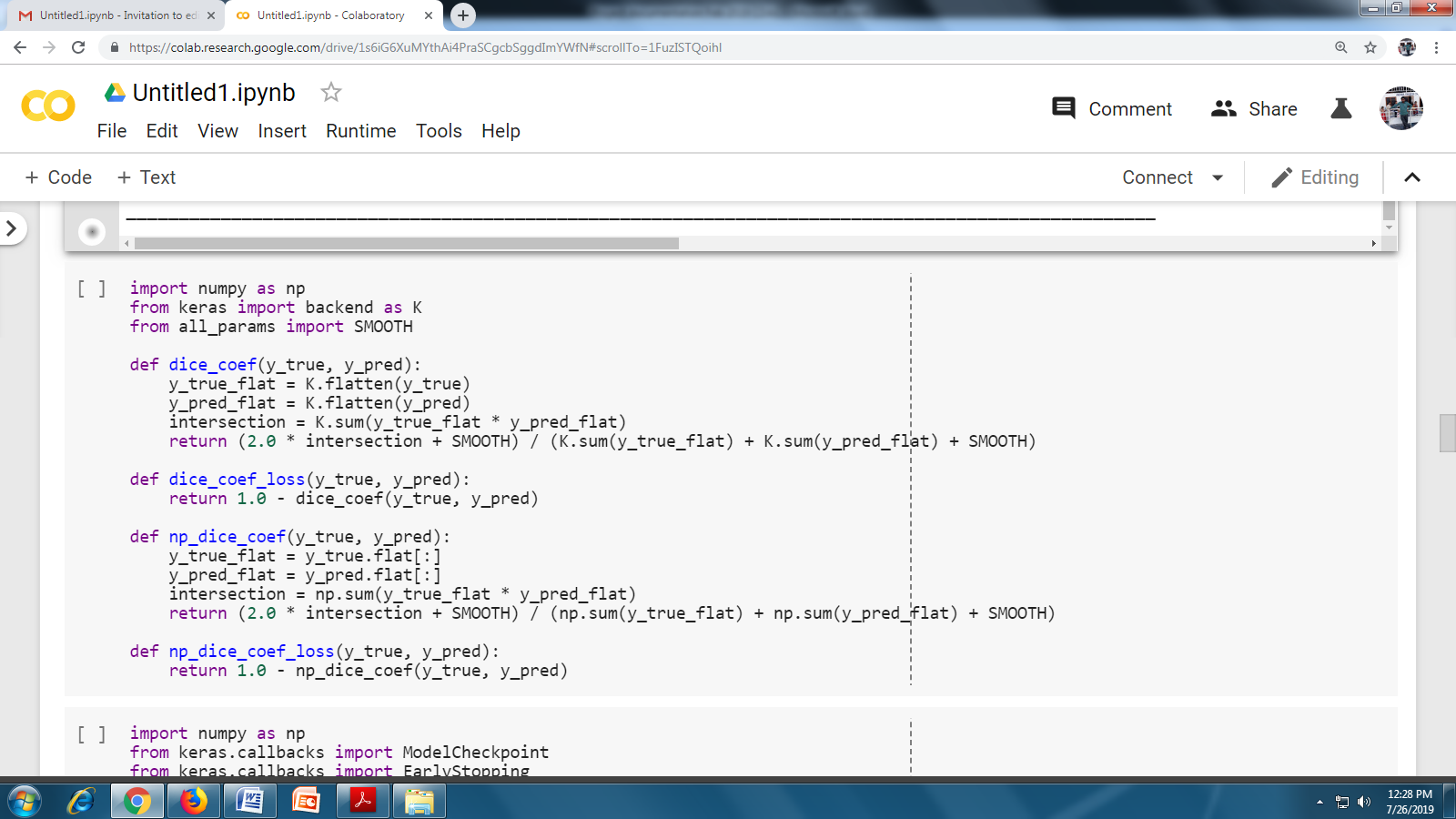
Most of the time dropout is applied to fully connected layers compared to convolutional layers. Since the convolutional layers don’t have a lot of parameters, overfitting is not a problem and therefore dropout would not have much effect. However, the additional gain in performance obtained by adding 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 prevent them from overfitting.





**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.



**Train the model:**

Train the model on X\_train, Y\_train data. We can create a separate folder checkpoints for best echo weights files. Finally the best model can be saved in WEIGHTS path file. To get the initial result, this U-net architecture was ported to Keras with tensor flow as back end; the result has score around 67%. This was improved upon by using the following techniques:

Dynamic learning rate: whenever the loss function stopped decreasing, a learning rate drop was added. I have used Keras call back function ReduceLROnPlateau for reducing learning rate. This callback monitors loss as a quantity and if no improvement is seen for 10 epochs, the learning rate is reduced by a factor by which the learning rate will be reduced.

new\_lr = lr \* factor

Adding dropout to a layer: dropout randomly drops weights in the layer it’s applied to during training and scales the weights so that the network keeps working during inference.

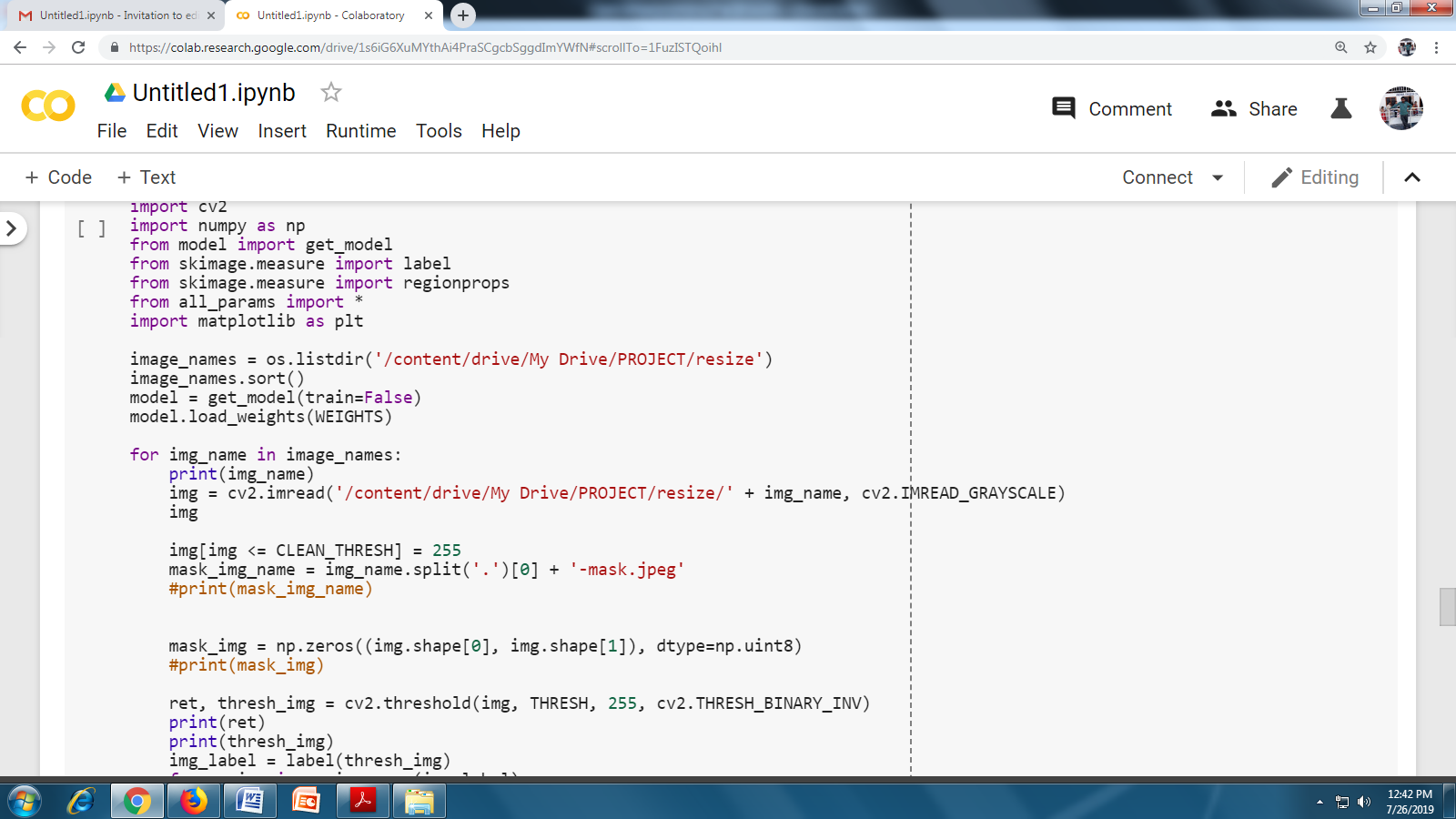
The final Keras model was derived by training in an iterative fashion, adjusting the parameters (e.g. learning rate) has score of 94%.

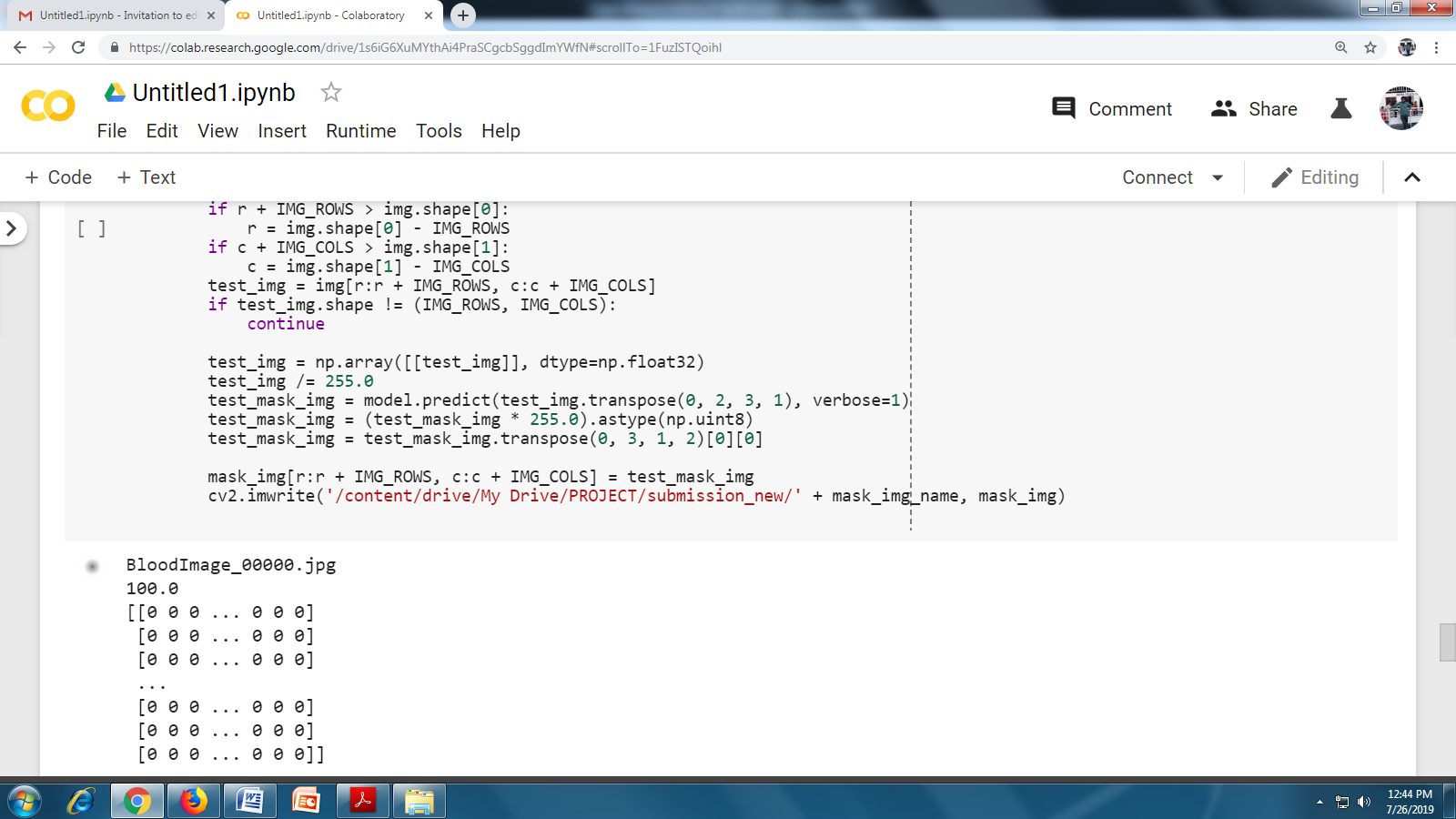




**Submission Data**

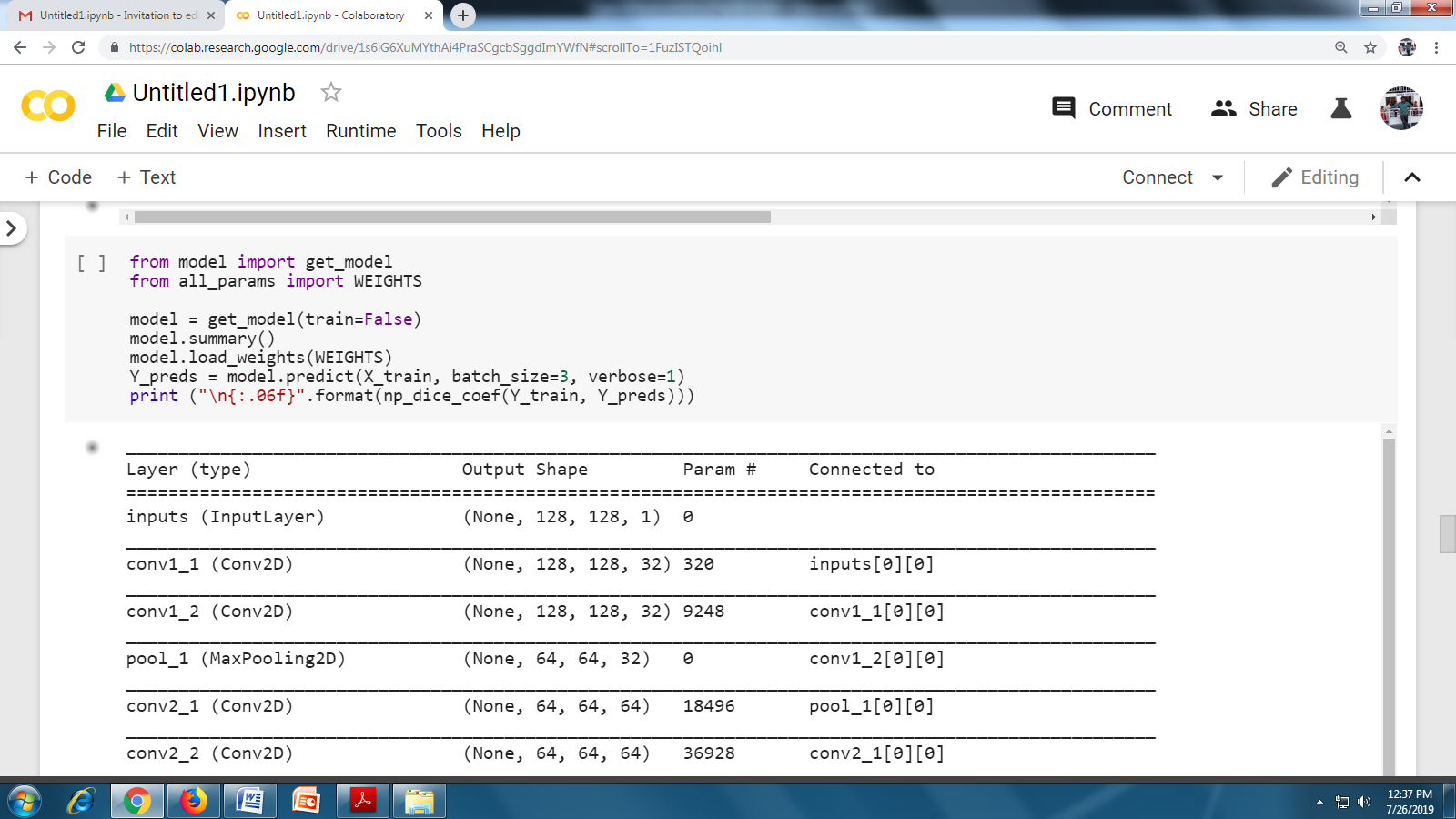
Now we have a model in place and trained on given train and mask images and predicted with 94% dice score. Now we have to generate masks from given test data

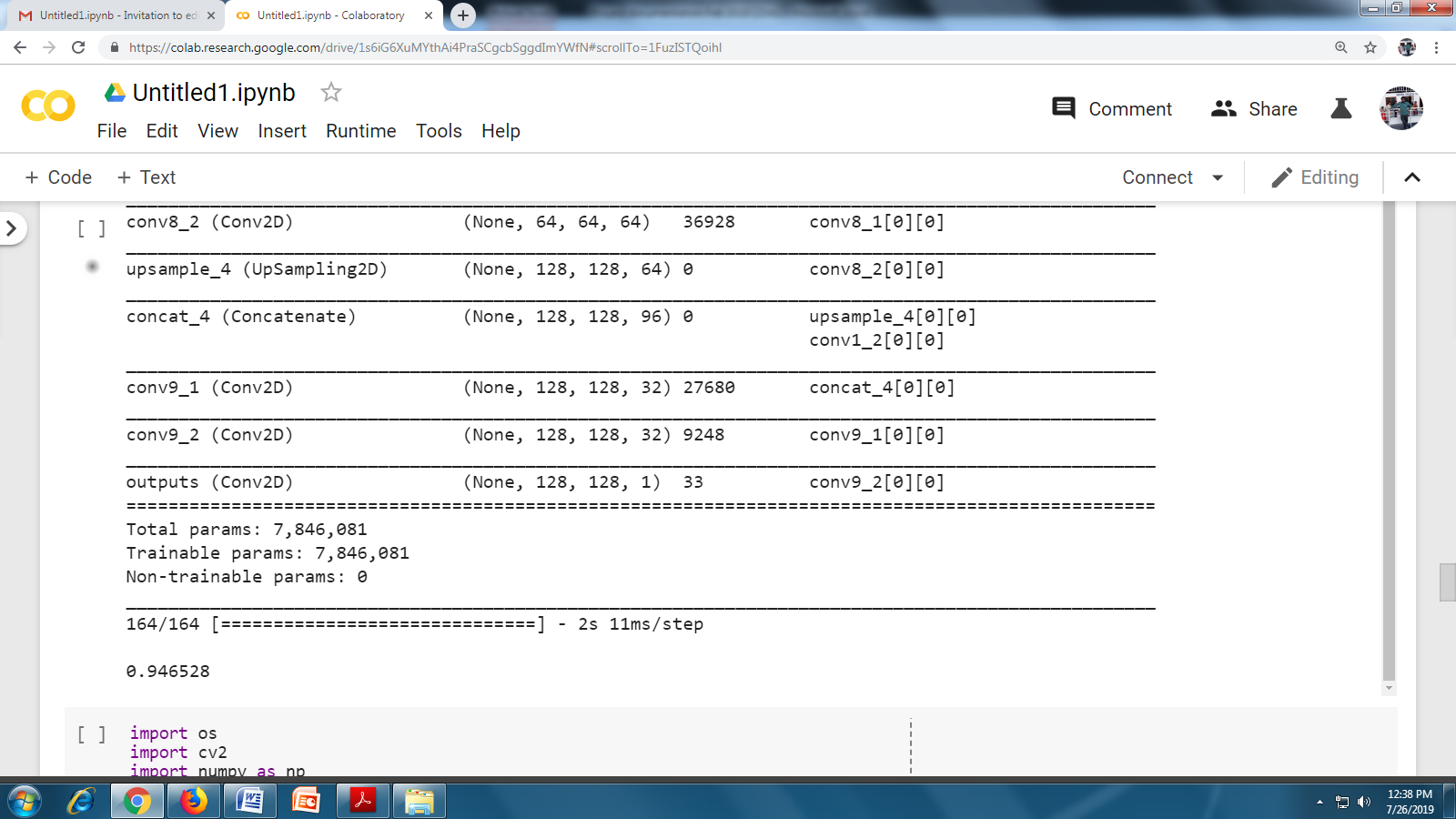




**Validation of the model:**

We need to validate our model using dice coefficient metrics and achieved dice score with modified Unet architecture.

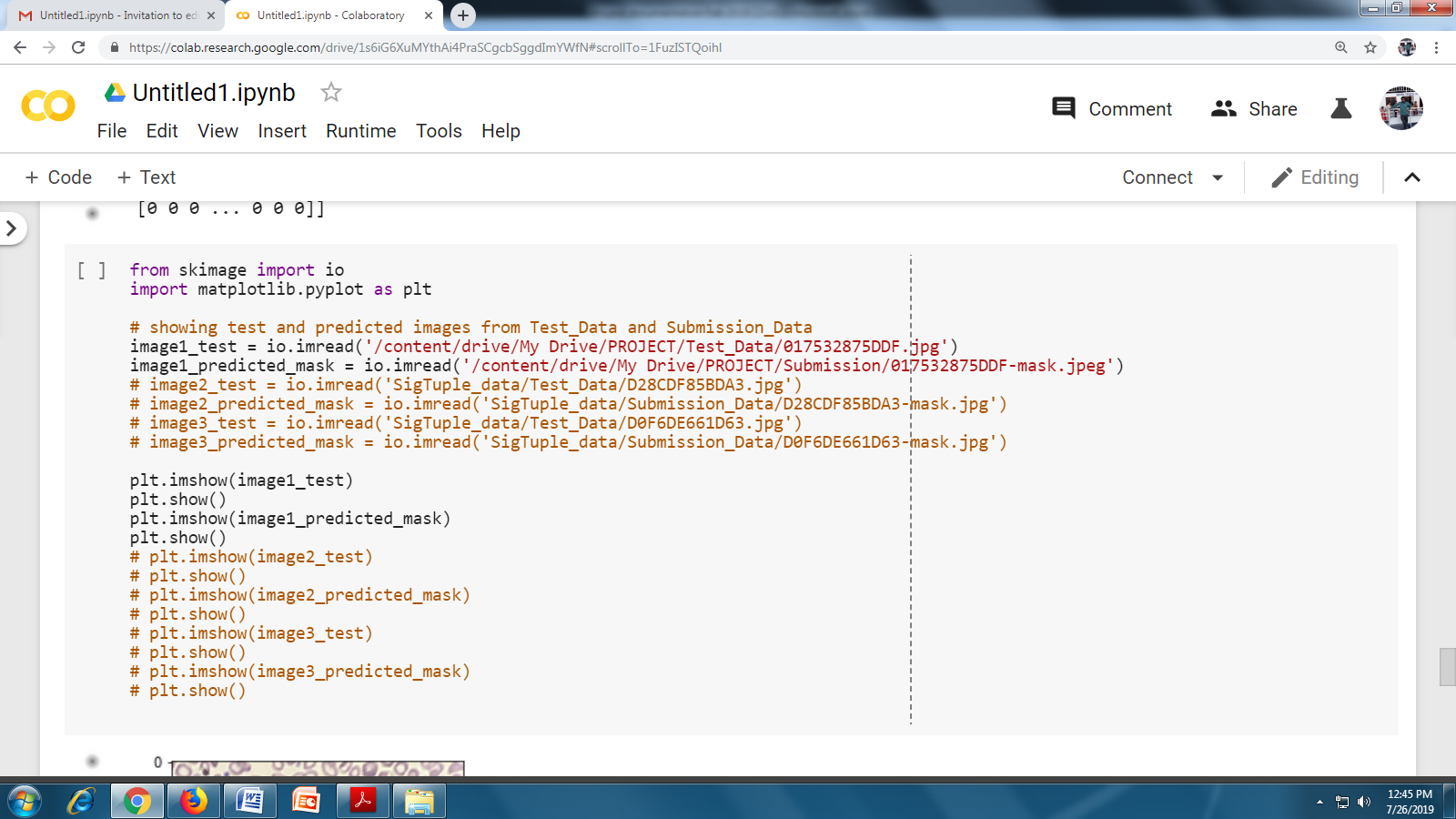


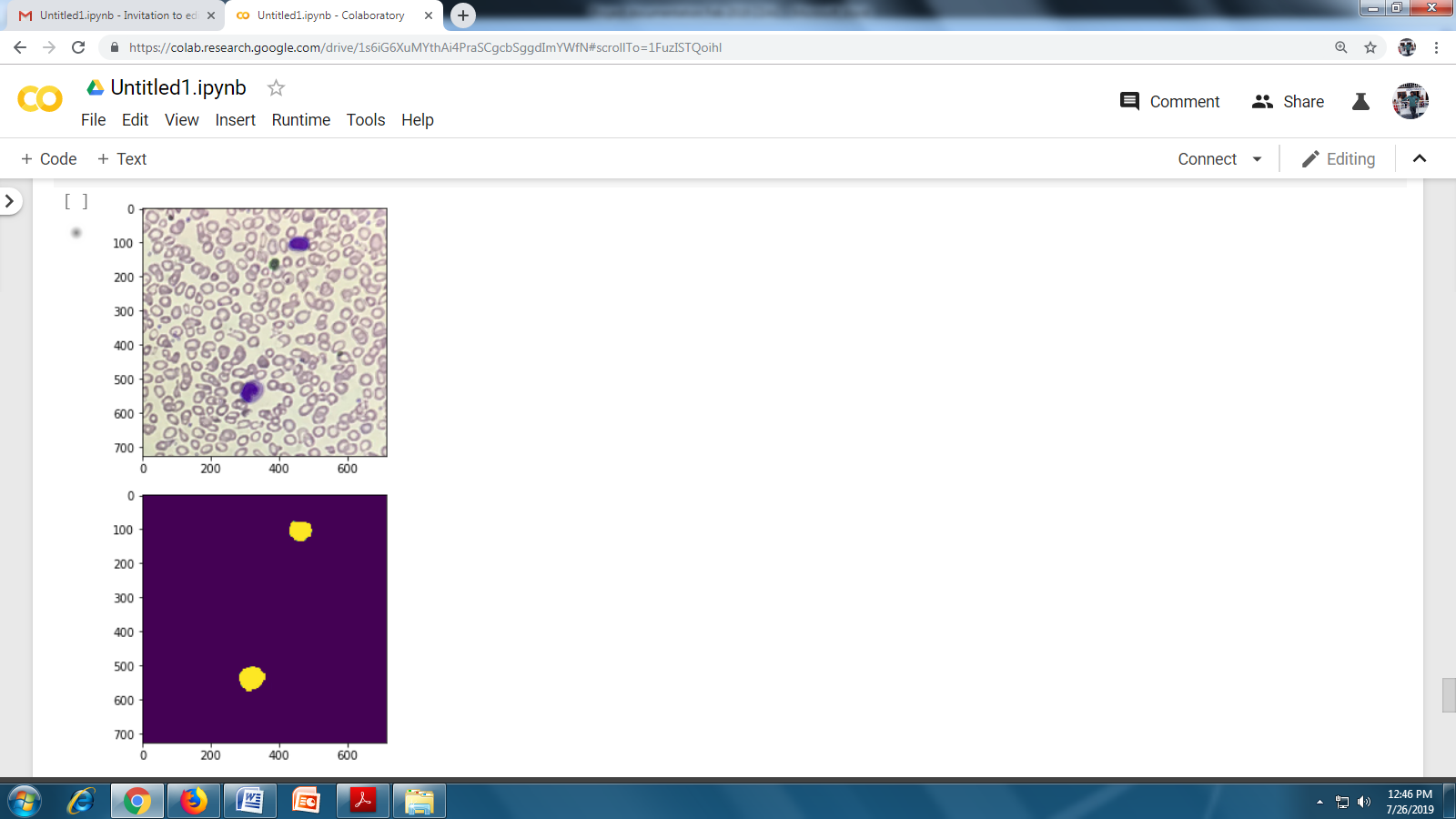


**Results:**

The model correctly demarcates the boundary of white blood cells irrespective of its size and count. As discussed above by adding dynamic learning rate and dropout the model improves a lot and achieved great results. We have used 0.25 as dropout rate, at any time ¾ neurons are active and contribute to learning the features. The higher the dropout, the less we would expect it to converge. So we pick this rate as good value and achieved good results.

All the images in test set are accurately demarcated with this model above 94%.





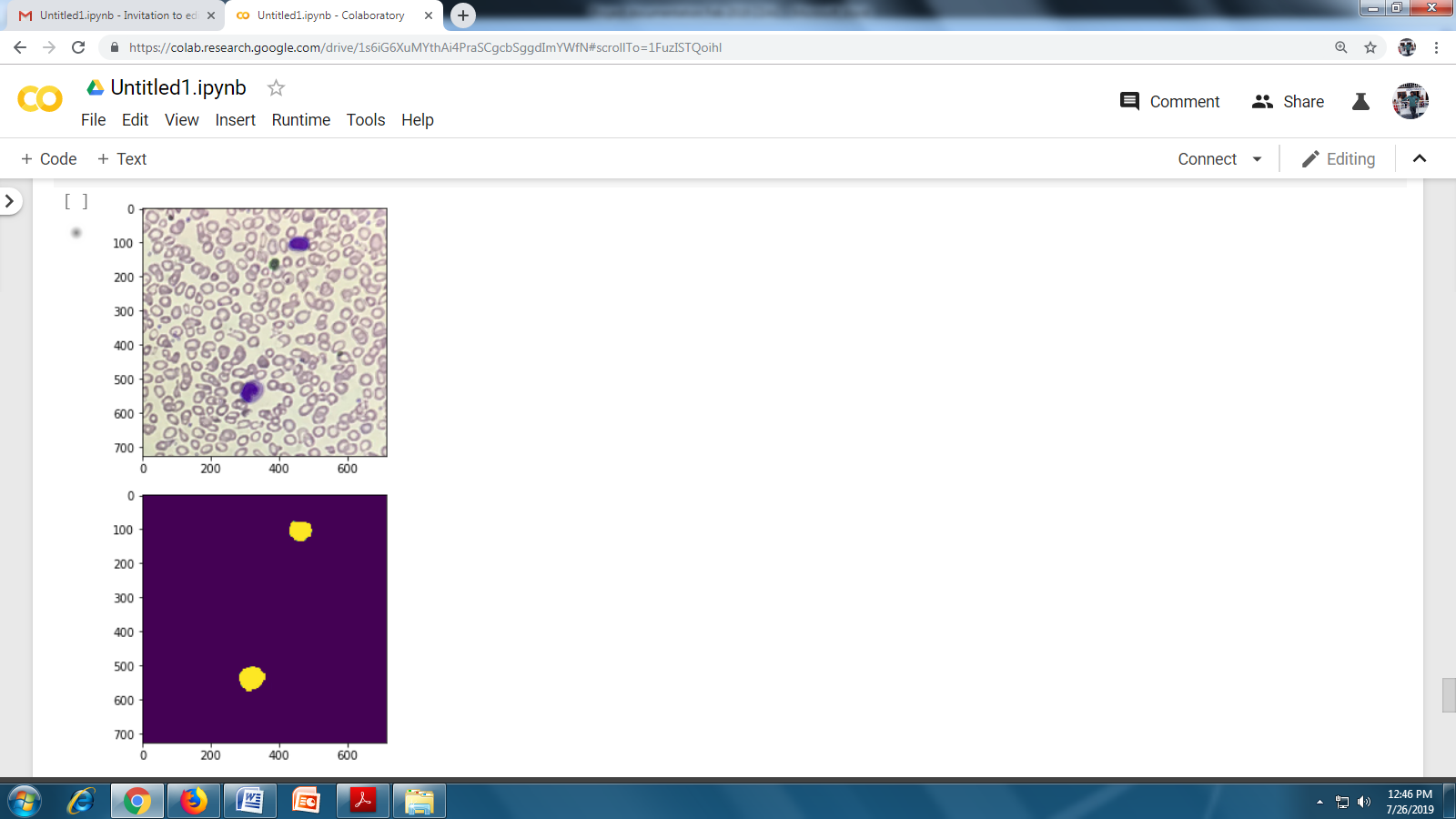
**Results**

**Model Evaluation and Validation**

We have used the given training set (both images and masks X\_train, Y\_train) to fit to our model and used dice coefficient as a metric to identify the accuracy of the model.

Finally, we applied trained model using dice score as metric to predict masks for X\_train and compare those masks with Y\_train. With y\_pred with y\_true.

The model achieved state of the art results with dice score of 94% and identified correct pixels of white blood cells.



**Justification:**

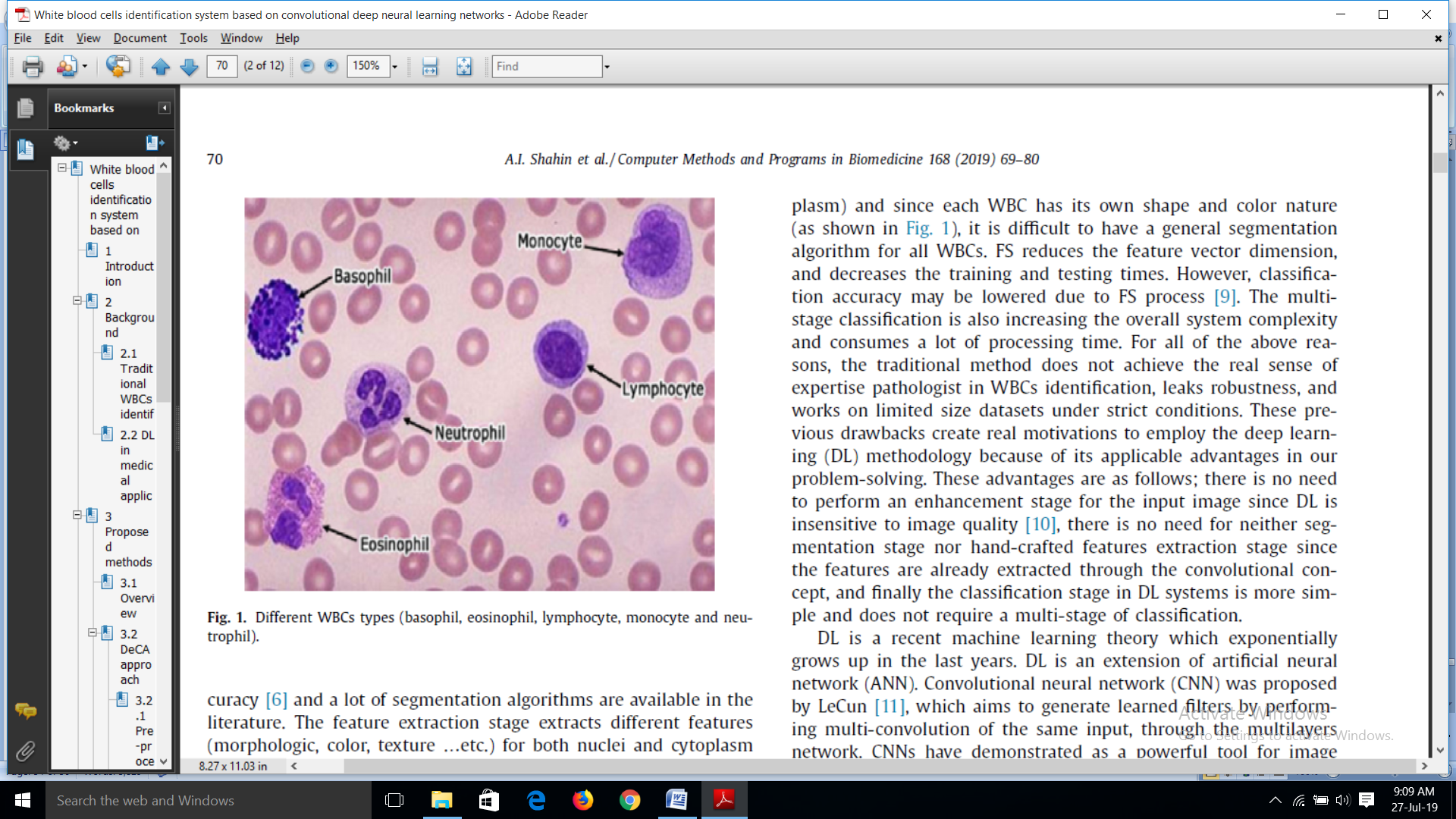
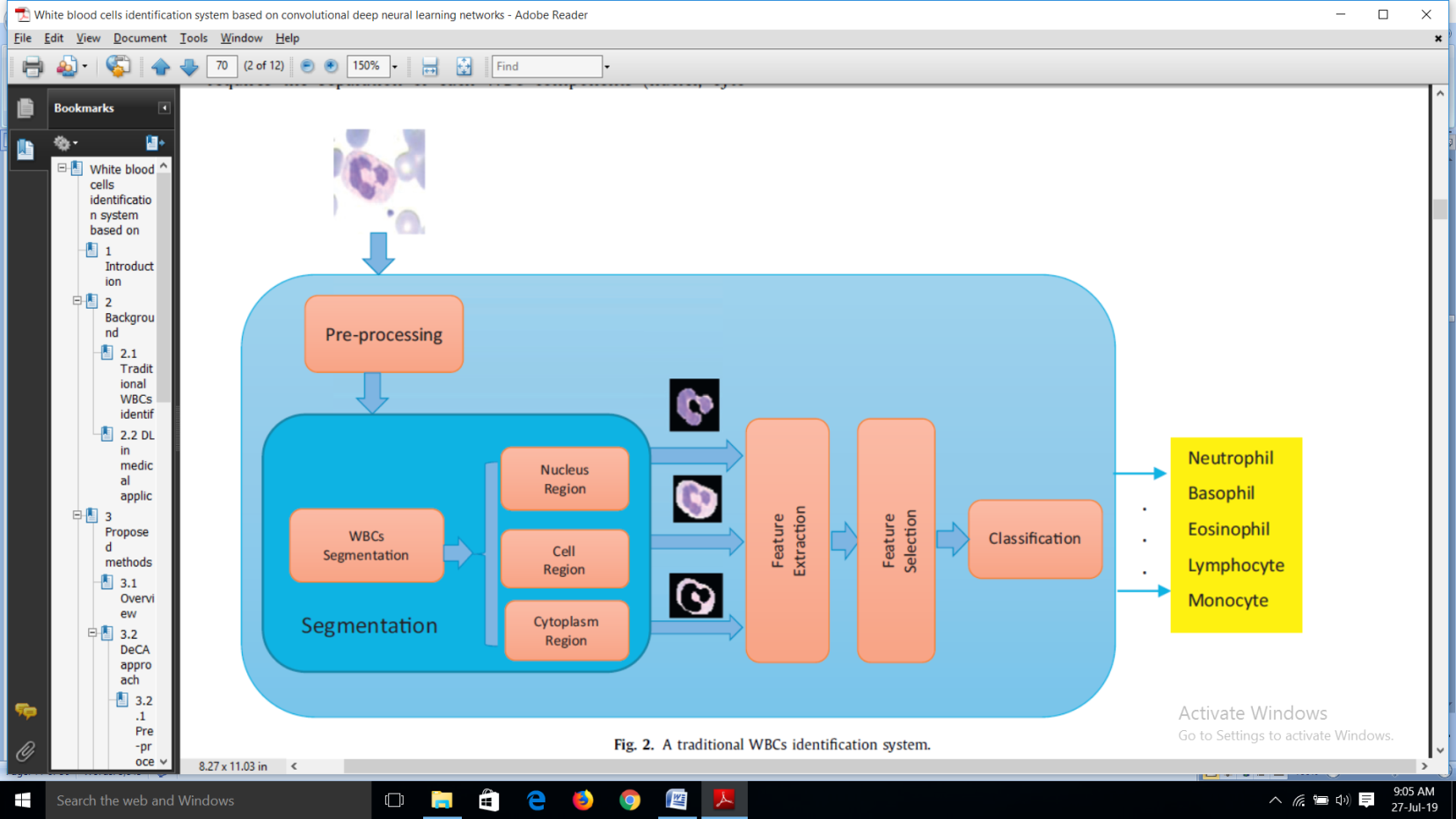
The model correctly demarcates the boundary of white blood cells irrespective of its size and count. As discussed above by adding dynamic learning rate and dropout the model improves a lot and achieved great results. We have used 0.25 as dropout rate, at any time ¾ neurons are active and contribute to learning the features. The higher the dropout, the less we would expect it to converge. So we pick this rate as good value and achieved good results.

All the images in test set are accurately demarcated with this model above 94%. And this dice score is far better compared to the bench mark model.

**Future Scope:**

After the supervised training of CNN on general images categories using state-of-the-art method, the pre-trained network has powerful features on generic vision tasks and specifically on image **category classification**.

The learned weights are adapted to shape, texture, complex features of the general images used. Then, the target dataset is passed through the pre-trained network with its previous weights and learned filters.

On Google colab machine each epoch of training time takes less than 5 sec and completed training in less than 5 min.

If we use multiple gpus and develop this model entirely on tensorflow back end, we will serve thousands of requests at a time and serve practical applications like real time white blood cells segmentation.

**Conclusion**

By using modified U-Net architecture model on original images and mask images and increasing the number of dropout layers as well as providing the dynamic learning rate, we have achieved the better performance accuracy of 94%.

# Bibliography

1)P.Yampri, \*. \*. (2006). White Blood Cell Classification based on the Combination of. *Research Center for Communication and Information Technology (ReCCIT)* , 4.

2)M.Saritha, P. (2016). Detection of Blood Cancer in Microscopic Images of Human Blood Samples: A Review . *International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) - 2016* , 5.

3)S.S.Savkare, S. (2015). Blood Cell Segmentation from Microscopic Blood. *2015 International Conference on Information Processing (ICIP)* , 4.

4)Olaf Ronneberger, P. F. (2015). U-Net: Convolutional Networks for Biomedical. *arXiv.*

5)Domain background: https://en.wikipedia.org/wiki/White\_blood\_cell

6) Data sets and Inputs:Hackathon:https://www.hackerearth.com/challenge/competitive/ sigtuple-aichallenge/ machine-learning/wbc-segmentation/

7)Data set link: https://s3-ap-southeast-1.amazonaws.com/he-publicdata/contests/ SigTuple\_data.tar

8)Evaluation Metrics: <https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93> Dice\_coefficient

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