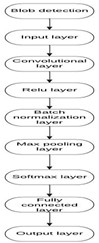
1. INTRODUCTION

The automated convolutional neural network is used to detect blood cells namely white blood cells, red blood cells, platelets and so on. CNN was proposed with novel blob detection methods which are called MSER (Maximally Stable Extremal Regions) and SURF (Speeded Up Robust Features), involved in pre-processing, which are employed to detect the white blood cells and extract the same from the given input image and a layer inclusion called Batch Normalization inside the hidden layers of the CNN which provides a faster training rate, helping the optimizer to perform well. The accuracy was proportionately increased to 1.41% compared to the CNN without Batch Norm layer. Overall sensitivity, accuracy, F1 score, precision, specificity, IOU (Intersection Over Union), AUC (Area Under the Curve) were the standard metrics chosen to analyse the performance of the CNN. Based on these results, the better performance was extracted using the CNN with Batch Norm layer and MSER (Maximally Stable Extremal Regions) for white blood cell (blob) detection to be specific the training time was reduced by 949 seconds and the accuracy was increased to 98.988%.

* 1. Project Overview

Data descriptors, such as classification and grouping, are used to characterize data items. Differentiating between dis- tinct kinds and subtypes of leukemia requires a thorough grasp of WBC characteristics. Although automatic and manual approaches are employed, WBC detection, segmentation, and classification techniques currently in use confront significant obstacles. WBC detection is done manually by pathologists, which is prone to human error and provides erroneous findings. This is a tough, time-consuming process that is susceptible to changes in pathologists’ classes. Only 77.4 percent of patients had pathologists who agreed on the diagnosis of leukemia. Other issues are connected with the complex structure of WBCs, such as uneven borders and WBCs and other blood components have textural similarities, which make Feature Extraction of WBC using Deep Learning problematic. White blood cells come in a variety of shapes, textures, colors, and intensities.

WBCs [1] are divided by subtypes, which include both be- nigh and cancerous cells. Furthermore, differences in staining and lighting make WBC detection more challenging. However, most laboratory-based automated WBC detection approaches rely image processing and pattern recognition approaches that are quantitative rather than qualitative. As a result, the creation of robust and generalized learning systems can be facilitated by the introduction of novel computer-aided techniques for precise WBC identification.



In this paper, CNN is used to perform white blood cell

detection using the given microscopic blood smear image. The objective is to use the blob detection methods which are called MSER [2] (Maximally Stable Extremal Regions) and SURF (Speeded Up Robust Features) in the preprocessing stage to detect the white blood cells and a layer named Batch Normalization is also included inside the hidden layers of CNN to speed up the training process and reduce the training time. Better results were obtained while using the MSER algorithm for blob detection and Batch Norm layer for CNN training. The layers of CNN are shown in Fig.1.

purpose

Different techniques have been used for brain tumors segmentation and detection. In general, MRI, CT, can be effective way methods to find out different types of diseases in the human body. Brain tumor will occur by the unnecessary growth of abnormal cells in the part of the human brain body in a messy method. Brain tumor detected by MRI imaging. Treatment according to its size, shape and location. Currently, many pictures treatment techniques have been presented to find and segment brain tumors of MRI. Brain tumors are developed as both benign and malignant tumors. Benign tumors do not affect other parts of the body but malignant tumors are cancer cells that can affect surrounding parts of the brain. The current, MRI has become a better technology for brain scanning from the medical side. The An MRI technique can take pictures of the brain. Therefore, the image processing and right segmentation is needed to detect tumors in the brain. The purpose of review of different methods of segmentation and classification of MRI images.

2. LITERATURE SURVEY

2.1 Existing problem

The identification and estimation of the number of white blood cells is one of the difficulties encountered in medical science. The abundance of red blood cells is the primary reason. The WBCs account for about 1% of the total blood volume in a healthy adult. Recognizing WBCs becomes more difficult as a result of the low proportion of WBCs in blood, making it harder to classify WBC subtypes.

Any change in the number of any classification's subtypes indicates that the body is experiencing a problem and responding to a particular pathogen. The type of the disease can determine the prognosis, which can assist in prescribing treatment for that particular condition. Under a light and electron microscope, studying blood-smeared slides is the traditional method for classifying WBC types. Before the blood cells are examined under a microscope, they are stained. Pathologists must examine the nucleus's shape and compare its size to that of RBCs for accurate identification. Since this is a

manual cycle it is inclined to mistake and is tedious. The greatest draw-

back with the manual classiﬁcation is the part of human mistake related with

mechanical checking of glass slide and the tradeoﬀ between the picture goal

furthermore, minuscule ﬁeld of view (FOV). An automated method for classifying white blood cells based on images of stained blood cells is required to overcome these drawbacks.

2.2 References

1. Meenakshi Banerjee, Punit Sharma, Amlin Chakrabarti, and Abhishek Bal (2018) Rough-Fuzzy C-Means and Shape-Based Properties for MRI Cancer cells Cell Segmentation and Analysis, Journal of King Saud University Computer and Information Sciences, vol.30, no.11. pp.1-18.

2. Adhi Lakshmi and Annadurai Arioli, "A Novel M-ACA-Based Cancer cells cell Segmentation and DAPP feature extraction with PPCSO-PCC-Based MRI Classification," Arabian Journal of Science and Engineering, vol.43, no.12, pp.7095-7111.Zotin Alexander2018: Svetlana Kirillova, Mikhail Kuratko, Yousif Hamad, and Konstantin Simonov Fuzzy C-means clustering-based edge detection in MRI cancer cells cell images, Procedia Computer Science, vol.126, no.3, pp.1261-1270

3. Survey on deep learning for radiotherapy by Philippe Meyer, Vincent Knoblet, Christophe Mazara, and Alex Aliment in 2018.Pages from Computers in Biology and Medicine126-146

4. Review of MRI-based Cancer cells cell Image Segmentation Using Deep Learning Methods, Ali Ism, Cam Turkoglu, and Malesich (2016), Procedia Computer Science, vol.102, no.1, pp.317-324.

5. "Edge detection in cancer cells cell images," by Anna Fabianski and Dominik Stankowski, 2008, International Conference on Perspective Technologies and Methods in MEMS Design, conference proceedings, pages60-62.

6. In "Improved Edge Detection Algorithm for Cancer cells cell Segmentation," Arsalan, Ikram Khan, and Sufyan Beg, MM (2015),58, no.16, pp.430-437.

7. Multifractal Texture Estimation for the Detection and Segmentation of Cancer cells, by Atid Islam, Syed MS Reza, and Khan M Iftekhar Uddin (2013), IEEE Transactions on Biomedical Engineering, vol.60, no.11, pp.3204-3215.

8. Chidambaram, T., and Perumal, K., published "Cancer cells cell segmentation using genetic algorithm and ANN techniques" in the conference proceedings of the 2017 IEEE International Conference on Power, Control, Signal, and Instrumentation Engineering (ICPCSI).970-982.

9. A watermarking-based medical image integrity control system and an image moment signature for tampering characterization was published in the IEEE Journal of Biomedical and Health Informatics in 2013, by Carteaux, GH, Huang, H. Shu, L. Luo, and Roux, C.17, no.6, pp.1057-1067.

10. Cancer cells cell Detection Using Artificial Neural Networks, published in the Journal of Science and Technology in 2012 by Dalia Mahmoud and Leather Mohamed13, no.2, pp.31-39.

11. Segmentation of cancer cells cell and enema along with healthy cancer cells tissues using wavelets and neural networks, Demir an, Toru, and Goler (2015), IEEE Journal of Biomedical and Health Informatics, vol.19, no.4, pp.1451-1458.

12. Mokhtar, HM, and El-Elegy, MT (2014) Cancer cell segmentation in cancer cells MRI employing a fuzzy method and class centre priors, EURASIP Journal on Image and Video Processing, vol.2014, publication no.21.

13. Cancer cells cell segmentation based on a hybrid clustering technique, Mean Abdel-Masoud, Mohammed Eulogy, and Rashid Al-Awadhi (2015), Egyptian Informatics Journal, vol.16, no.1, pp.71-81.

14. "An Automatic Learning-Based Framework for Robust Nucleus Segmentation," published in IEEE Transactions on Medical Imaging, vol.35, no.2, pp.550-566.

15. MA 2016 by Garima Singh and Ansari Utilizing K-means segmentation and a normalized histogram, the effective detection of cancer cells from MRIS was achieved. Proceedings of the India International Conference on Information Processing (IICIP), pages1-6.

16.Maria A. Zulu Aga, Rosalind Pratt, Premal A. Patel, Michael Earthen, Tom Doel, and Anna L. David are Guotie Wang, Wendi Li, and Anna L. David. Interactive Medical Image Segmentation Using Deep Learning with Image-Specific Fine Tuning" by Jan Depress, Sébastien Our Selin, and Tom Veratrin, IEEE Transactions on Medical Imaging, vol.37, no.7, pp.1562-1573.

17. Classification using deep learning neural networks for cancer cells, Future Computing and Informatics Journal, vol. Heba Mohsen, El-Sayed A EI-Dashan, El-Sayed M El-Hobart, and Abdel-Badeeh M. Salemd (2018)3, no.1, pp.68-71.

18. Classification using deep learning neural networks for cancer cells, by Heba Mohsen, El-Sayed A. El-Dahshan, El-Sayed M. El-Horbaty, and Abdel-Badech M. Salemd, Future Computing and Informatics Journal, vol.3, no.1, pp.68-71

19. Iftekharuddin, KM, Islam, MA, Zheng, J., Ogg, R.J., and Lanningham, F.MRI-based detection of cancer cells: Technique and Statistical Validation, conference proceedings from the 2006 Fortieth Asilomar Conference on Signals, Systems, and Computers, pages1983-1987.

20. A new method for cancer cells cell segmentation based on watershed and edge detection algorithms in the HSV color model, Ishita Maiti and Monisha Chakraborty (2012), National Conference on Computing and Communication Systems, conference proceedings, pages1-5.

21. Jainy Sachdeva, Vinod Kumar, Indra Gupta, Niranjan Khandelwal and Chirag Kamal Ahuja 2016, A bundle SFERCB-"Division, include extraction, decrease and order investigation by both SVM and ANN for disease cells, Applied Delicate Figuring, vol.47, no.10, pp.151-167.

22. Efficient Multilevel Cancer cells Cell Segmentation With Integrated Bayesian Model Classification, Jason J. Corso, Eitan Sharon, Shishir Dube, Suzie El-Saden, Usha Sinha, and Alan Yuille (2008), IEEE Transactions on Medical Imaging, vol. 27, no. 5, pages629-640.

23. Jayachandran Abnormality Segmentation and Classification of Multi-Class Cancer cells in MR Images Using Fuzzy Logic-Based Hybrid Kernel SVM, published in International Journal of Fuzzy Systems in 2015 by A and Kharmega Sundararaj, G17, no.3, pp.434-443.

24. Discriminative clustering and feature selection for cancer cells MRI segmentation was published in 2015 by Kong, Y., Deng, Y., and Q. Dai. Volume of IEEE Signal Processing Letters22, no.5, pp.573-577.

25. Lal.S and Chandra, M.Proficient calculation for contrast upgrade of regular pictures', Global Bedouin Diary of Data Innovation, vol.11, no.1, pp.95-102.

26. Guizhi Xu, Qingxin Yan, Lei Guo, Lei Zhao, Youxi Wu, and Ying LiOne-Class Immune Feature Weighted SVMS for MR Image Cancer cells cell Detection, IEEE Transactions on Magnetics, vol.47.no.10, pp.3849-3852.

Manisha, B. Radhakrishnan, and L. Padma Suresh published "Cancer cells cell region extraction using edge detection method in cancer cells MRI images" in the conference proceedings of the 2017 International Conference on Circuit Power and Computing Technologies (ICCPCT).1-5.

28. Max.Weighted Local Variance-Based Edge Detection and Its Application to Vascular Segmentation in Magnetic Resonance Angiography was published in 2007 by W K Law and Albert CS Chung in IEEE Transactions on Medical Imaging.26, no.9, pp.1224-1241.

29. A Learning-Based Similarity Fusion and Filtering Approach for Biomedical Image Retrieval Using SVM Classification and Relevance Feedback, IEEE Transactions on Information Technology in Biomedicine, 2011, by Md Mahmudur Rahman, Sameer K Antani, and George R Thoma15, no.4, pp.640-646.

30. Cancer cells cell Segmentation Based on Local Independent Projection-Based Classification, Meiyan Huang, Wei Yang, Yao Wu, Jun Jiang, Wufan Chen, and Qianjin Feng (2014), IEEE Transactions on Biomedical Engineering, vol.61, no.10, pp.2633-2645.

31. Ming-Ni Wu, Chia-Chen Lin and Jaw Chen Chang 2007, 'Disease cell Discovery Utilizing Variety Based K-Means Grouping Division. Conference papers from the Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007).vol.2, pp.245-250.

32. Mohammed Havaci and Alex Davi 2016, Malignant growth cell Division with Profound Brain organization, Clinical Picture examination, vol.35, no.1.pp.18-31

33. Namcirakpam Dhanachandra, Yambem Jina Chanu, and Khumanthem Manglem Image Segmentation Using K-means and Subtractive Clustering Algorithms," in Procedia Computer Science, vol.54, no.8, pp.764-771.

2.3 Problem Statement Definition

In medical science, one of the challenges faced is the identiﬁcation and determin-

ing the number of white blood cells. The major reason is the abundance of red

blood cells. In a healthy adult, the WBCs make approximately 1% of the total

blood volume. Due to this low proportion of the WBCs in blood, the recognition

of the WBCs become a challenge which further toughens the job of classifying

the subtypes of WBCs.

The change in number of any subtype of classiﬁcation means that there is a

problem in the body and the body isresponding to a type of pathogen. The fur-

there prognosis of the disease can be determined by the disease type and can help

in prescription of treatment for that speciﬁc ailment. The traditional method

of classifying the WBC types includes studying the blood smeared slides under

light and electron microscope. The blood cells are stained prior to studying them

under a microscope. For perfect identiﬁcation, the pathologists need to look

for the shape of nucleus and compare the size relative to RBCs. Since this is a

manual process it is prone to error and is time consuming. The biggest draw-

back with the manual classiﬁcation is the aspect of human error associated with

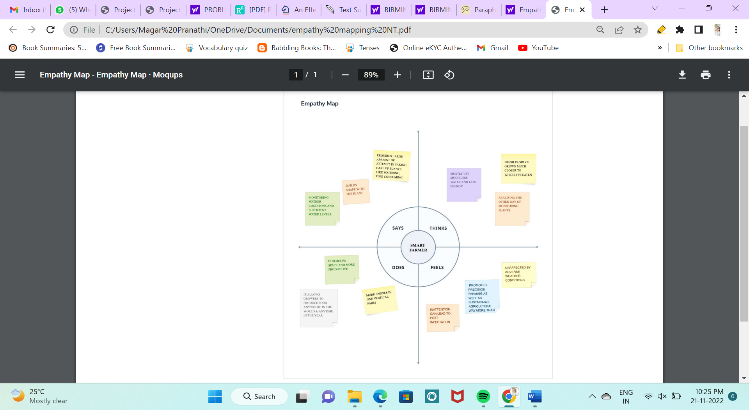
mechanical scanning of glass slide and the tradeoﬀ between the image resolution

and microscopic ﬁeld of view (FOV). To overcome these disadvantages, there is a need for an automated method to classify the white blood cells using an images

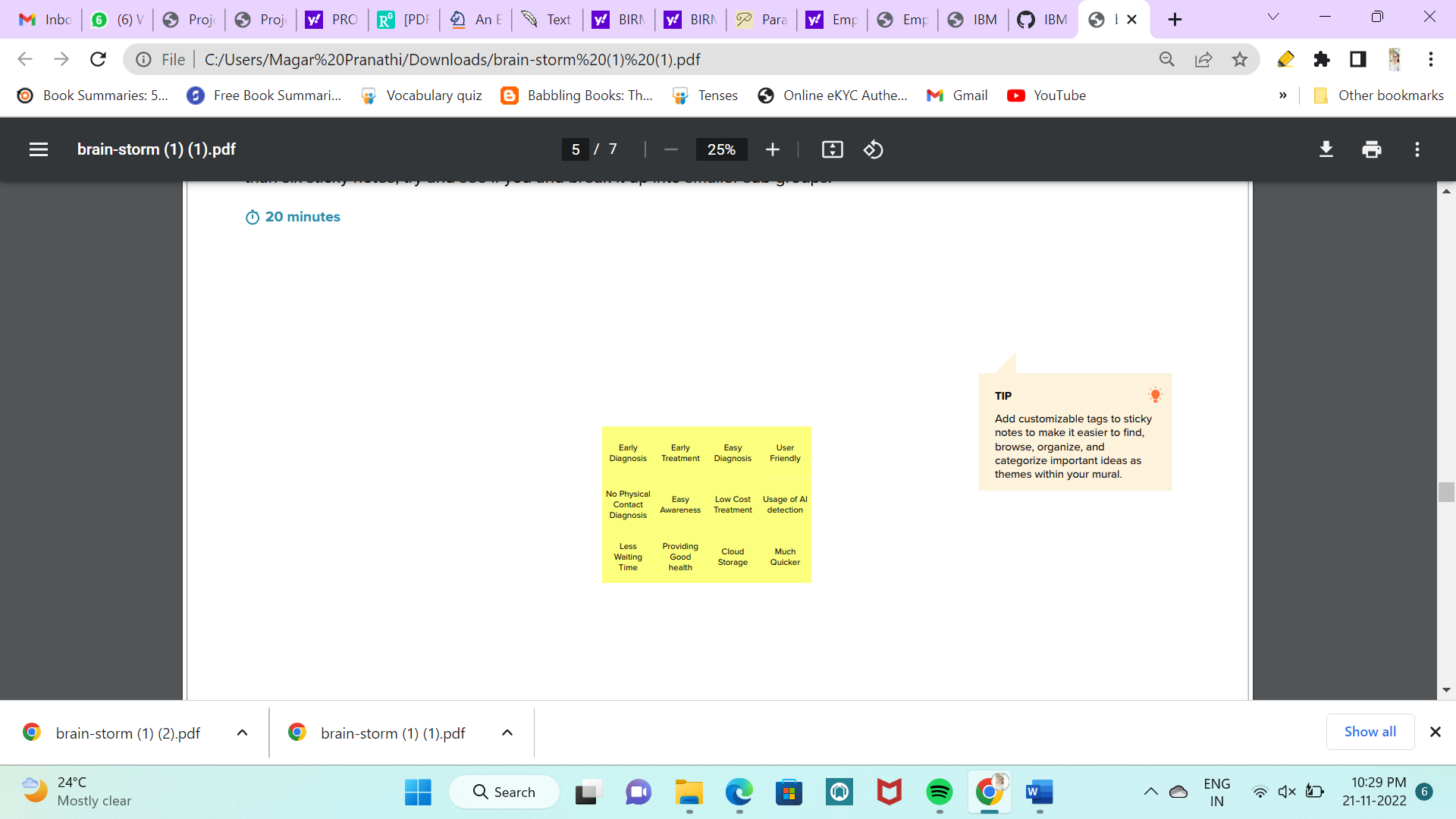
of stained blood cells.

1. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



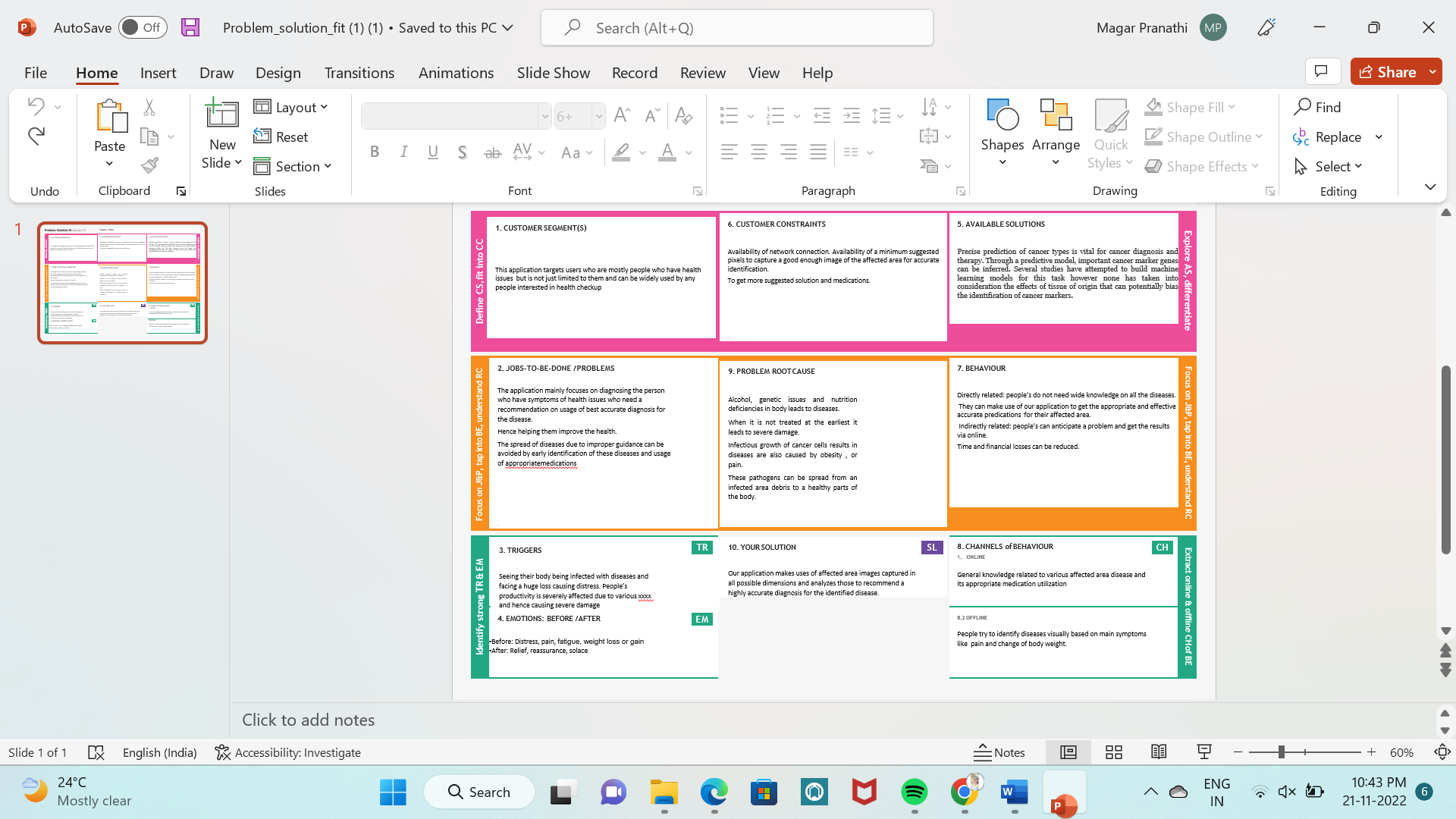
3.2 Ideation & Brainstorming



3.3 Proposed Solution

|  |  |  |
| --- | --- | --- |
| S.No. | PARAMETER | DESCRIPTION |
| 1 | Problem Statement (Problem to be solved) | In medical side, the difficulty of division and their nuclei from content of MRI image can be faced by the radiologists. At present, the segmentation from MRI image can be a challenging difficulty due to structure and location in human body when using a multimodal imaging data. The main challenges of tumor detection are less of accuracy to detect tumor area and to segment the tumor area. |
| 2 | Idea / Solution description | The image segmentation can be the essential complexity in tumor recognition in MRI image. But the tumor segmentation of separation task can be very essential to recognize the tumor and efficient diagnosis. |
| 3 | Novelty / Uniqueness | Quantitative and qualitative data about the beign tumor can be given by the proper segmentation process and it can be utilized to recognize what are the good treatments for patient and to get the better plan by doctor who treats the patient. Effective tumor detection can be happened when image analysis is going to be easy for understanding and segmenting |
| 4 | Social Impact / Customer Satisfaction | Dimensionality reduction that can indicate the important parts of the medical image as a feature vector. While large image sizes are used, this approach will be decreased feature demonstration can be needed to rapidly finish the processes like matching and retrieval of MRI image. From every MRI, four kinds of features can be extracted such as symmetry, intensity, texture and shape deformation depending on the some filtering methods. |
| 5 | Business Model (Revenue Model) | MRI is used to diagnose and analyze many diseases such as tumors, neurological diseases, epilepsy, etc. Usually, a system completely processed by hardware/computer helps automate this process to obtain accurate and fast results. |
| 6 | Scalability of the Solution | Numerous algorithms and methods have been presented for manual, semi and fully automated tumor segmentation due the complicated tumor segmentation process in MRI image. |

3.4 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

* Medical Image Analysis with :
* MATLAB
* With MATLAB you can:
* Visualize and explore 2D images and 3D volumes.
* Process very large multiresolution and high resolution images.
* Simplify medical image analysis tasks with built-in image segmentation algorithms
* Parse, Load, visualize, and process images.

1. PROJECT DESIGN

5.1 DATA FLOW

The user interacts with the UI (User Interface) and give the image as input.

• Then the input image is then pass to our flask application,

• And finally with the help of the model which we build we will classify the result and showcase it on the UI.

To accomplish this, we have to complete all the activities and tasks listed below:

• Data Collection.

• Collect the dataset or Create the dataset

• Data Preprocessing.

• Import the ImageDataGenerator library

• Configure ImageDataGenerator class

• ApplyImageDataGenerator functionality to Trainset and Testset

• Model Building

• Import the model building Libraries

• Initializing the model

• Adding Input Layer

• Adding Hidden Layer

• Adding Output Layer

• Configure the Learning Process

• Training and testing the model

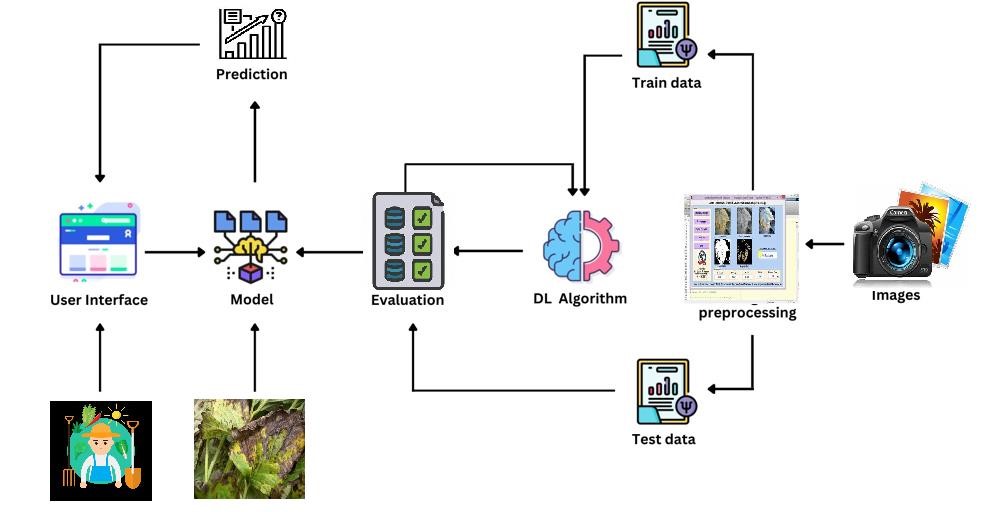
• Save the Model

• Application Building

• Create an HTML file

• Build Python Code

5.2 Solution & Technical Architecture



5.3 User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Stage** | **Awareness** | **Consideration** | **Decision** | **Service** | **Loyalty** |
| **Customer Actions** | View our add from websites social medias and hear about from friends. | Compare our performance with existing system and do research for improvement | Try this cancer detection application | Receive Product  Read product documentation. | Share Honest experience |
| **Touchpoints** | Modern Digital Platform,  Social media | Website,  word of mouth | Mobile application | Chat bot,  Email | Customer reviews sites |
| **Customer Experience** | Interested, Sceptical | Inquisitive, Excited | Excited, Exhilarated | Frustrated | Satisfied |
| **KPIs** | Take a survey on the number of people reached | Recent website  visitor’s | Application reach to the customer | Product reviews by the customer product success rate | Downfall rate,  customer satisfaction rate |
| **Business Goals** | Create awareness to the people who uses technology for cancer detection | Take steps to increase new website visitors | Increase application reach to the customer, online services | Minimized waiting time, increased service for customers | Maintain positive reviews,  increase  Retention rate |
| **Team Involved** | Marketing. | App development team | Online development,  customer service,  digital marketing | Product success and customer satisfaction | Business development team,  app development team |

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

|  |
| --- |
| from keras.preprocessing.image import img\_to\_array |
|  |  |
|  |  |
|  | class ImageToArrayPreprocessor: |
|  | def \_\_init\_\_(self, data\_format=None): |
|  | self.dataFormat = data\_format |
|  |  |
|  | def preprocess(self, image): |
|  | return img\_to\_array(image, data\_format=self.dataFormat) |
| 6.2 Sprint Delivery Schedule |  |

**from** keras.models **import** load\_model

**from** keras.preprocessing **import** image

**import** numpy **as** np

**import** cv2

**import** matplotlib.pyplot **as** plt

Using TensorFlow backend.

In [3]:

model **=** load\_model('weights.hdf5')

model**.**compile(loss**=**'binary\_crossentropy',

optimizer**=**'rmsprop',

metrics**=**['accuracy'])

WARNING:tensorflow:From /home/diwas/.conda/envs/tf/lib/python3.7/site-packages/tensorflow/python/framework/op\_def\_library.py:263: colocate\_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

WARNING:tensorflow:From /home/diwas/.conda/envs/tf/lib/python3.7/site-packages/keras/backend/tensorflow\_backend.py:3445: calling dropout (from tensorflow.python.ops.nn\_ops) with keep\_prob is deprecated and will be removed in a future version.

Instructions for updating:

Please use `rate` instead of `keep\_prob`. Rate should be set to `rate = 1 - keep\_prob`.

WARNING:tensorflow:From /home/diwas/.conda/envs/tf/lib/python3.7/site-packages/tensorflow/python/ops/math\_ops.py:3066: to\_int32 (from tensorflow.python.ops.math\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

In [4]:

img **=** image**.**load\_img('cancer/val/Cancer/\_2\_4392.jpeg', target\_size**=**(150, 150))

imgplot **=** plt**.**imshow(img)

x **=** image**.**img\_to\_array(img)

x **=** np**.**expand\_dims(x, axis**=**0)

images **=** np**.**vstack([x])

classes **=** model**.**predict\_classes(images, batch\_size**=**10)

**if** classes **==** [1]:

print("cancer")

**else**:

print("Normal")

cancer

7. CODING & SOLUTIONING

from keras.models import Model

from keras.layers import Input

from keras.layers import Conv2D

from keras.layers import MaxPool2D

from keras.layers import UpSampling2D

from keras.layers import Dropout

from keras.layers import concatenate

from all\_params import IMG\_ROWS, IMG\_COLS

def get\_model(input\_shape=(IMG\_ROWS, IMG\_COLS, 1), train=True):

layers = {}

layers['inputs'] = Input(shape=input\_shape, name='inputs')

layers['conv1\_1'] = Conv2D(32, (3, 3), padding='same', activation='relu', name='conv1\_1')(layers['inputs'])

layers['conv1\_2'] = Conv2D(32, (3, 3), padding='same', activation='relu', name='conv1\_2')(layers['conv1\_1'])

layers['pool\_1'] = MaxPool2D(pool\_size=(2, 2), name='pool\_1')(layers['conv1\_2'])

if train == True:

layers['dropout\_1'] = Dropout(0.25, name='dropout\_1')(layers['pool\_1'])

layers['conv2\_1'] = Conv2D(64, (3, 3), padding='same', activation='relu', name='conv2\_1')(layers['dropout\_1'])

else:

layers['conv2\_1'] = Conv2D(64, (3, 3), padding='same', activation='relu', name='conv2\_1')(layers['pool\_1'])

layers['conv2\_2'] = Conv2D(64, (3, 3), padding='same', activation='relu', name='conv2\_2')(layers['conv2\_1'])

layers['pool\_2'] = MaxPool2D(pool\_size=(2, 2), name='pool\_2')(layers['conv2\_2'])

if train == True:

layers['dropout\_2'] = Dropout(0.25, name='dropout\_2')(layers['pool\_2'])

layers['conv3\_1'] = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv3\_1')(layers['dropout\_2'])

else:

layers['conv3\_1'] = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv3\_1')(layers['pool\_2'])

layers['conv3\_2'] = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv3\_2')(layers['conv3\_1'])

layers['pool\_3'] = MaxPool2D(pool\_size=(2, 2), name='pool\_3')(layers['conv3\_2'])

if train == True:

layers['dropout\_3'] = Dropout(0.25, name='dropout\_3')(layers['pool\_3'])

layers['conv4\_1'] = Conv2D(256, (3, 3), padding='same', activation='relu', name='conv4\_1')(layers['dropout\_3'])

else:

layers['conv4\_1'] = Conv2D(256, (3, 3), padding='same', activation='relu', name='conv4\_1')(layers['pool\_3'])

layers['conv4\_2'] = Conv2D(256, (3, 3), padding='same', activation='relu', name='conv4\_2')(layers['conv4\_1'])

layers['pool\_4'] = MaxPool2D(pool\_size=(2, 2), name='pool\_4')(layers['conv4\_2'])

if train == True:

layers['dropout\_4'] = Dropout(0.25, name='dropout\_4')(layers['pool\_4'])

layers['conv5\_1'] = Conv2D(512, (3, 3), padding='same', activation='relu', name='conv5\_1')(layers['dropout\_4'])

else:

layers['conv5\_1'] = Conv2D(512, (3, 3), padding='same', activation='relu', name='conv5\_1')(layers['pool\_4'])

layers['conv5\_2'] = Conv2D(512, (3, 3), padding='same', activation='relu', name='conv5\_2')(layers['conv5\_1'])

layers['upsample\_1'] = UpSampling2D(size=(2, 2), name='upsample\_1')(layers['conv5\_2'])

layers['concat\_1'] = concatenate([layers['upsample\_1'], layers['conv4\_2']], name='concat\_1')

layers['conv6\_1'] = Conv2D(256, (3, 3), padding='same', activation='relu', name='conv6\_1')(layers['concat\_1'])

layers['conv6\_2'] = Conv2D(256, (3, 3), padding='same', activation='relu', name='conv6\_2')(layers['conv6\_1'])

if train == True:

layers['dropout\_6'] = Dropout(0.25, name='dropout\_6')(layers['conv6\_2'])

layers['upsample\_2'] = UpSampling2D(size=(2, 2), name='upsample\_2')(layers['dropout\_6'])

else:

layers['upsample\_2'] = UpSampling2D(size=(2, 2), name='upsample\_2')(layers['conv6\_2'])

layers['concat\_2'] = concatenate([layers['upsample\_2'], layers['conv3\_2']], name='concat\_2')

layers['conv7\_1'] = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv7\_1')(layers['concat\_2'])

layers['conv7\_2'] = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv7\_2')(layers['conv7\_1'])

if train == True:

layers['dropout\_7'] = Dropout(0.25, name='dropout\_7')(layers['conv7\_2'])

layers['upsample\_3'] = UpSampling2D(size=(2, 2), name='upsample\_3')(layers['dropout\_7'])

else:

layers['upsample\_3'] = UpSampling2D(size=(2, 2), name='upsample\_3')(layers['conv7\_2'])

layers['concat\_3'] = concatenate([layers['upsample\_3'], layers['conv2\_2']], name='concat\_3')

layers['conv8\_1'] = Conv2D(64, (3, 3), padding='same', activation='relu', name='conv8\_1')(layers['concat\_3'])

layers['conv8\_2'] = Conv2D(64, (3, 3), padding='same', activation='relu', name='conv8\_2')(layers['conv8\_1'])

if train == True:

layers['dropout\_8'] = Dropout(0.25, name='dropout\_8')(layers['conv8\_2'])

layers['upsample\_4'] = UpSampling2D(size=(2, 2), name='upsample\_4')(layers['dropout\_8'])

else:

layers['upsample\_4'] = UpSampling2D(size=(2, 2), name='upsample\_4')(layers['conv8\_2'])

layers['concat\_4'] = concatenate([layers['upsample\_4'], layers['conv1\_2']], name='concat\_4')

layers['conv9\_1'] = Conv2D(32, (3, 3), padding='same', activation='relu', name='conv9\_1')(layers['concat\_4'])

layers['conv9\_2'] = Conv2D(32, (3, 3), padding='same', activation='relu', name='conv9\_2')(layers['conv9\_1'])

if train == True:

layers['dropout\_9'] = Dropout(0.25, name='dropout\_9')(layers['conv9\_2'])

layers['outputs'] = Conv2D(1, (1, 1), activation='sigmoid', name='outputs')(layers['dropout\_9'])

else:

layers['outputs'] = Conv2D(1, (1, 1), activation='sigmoid', name='outputs')(layers['conv9\_2'])

model = Model(inputs=layers['inputs'], outputs=layers['outputs'])

return model

EXPLANATION

* The code then creates a new variable called conv1\_2 which will be used to create an input layer for our first convolutional layer, Conv1D.
* The name of this layer is 'conv1\_2'.
* Next, we create another variable called conv2\_2 which will be used to create an input layer for our second convolutional layer, Conv2D.
* The name of this layer is 'conv2\_2'.
* l e r n t o c o m p u t e v i s u a l f i g u r e s w h i c h w o u l d b e y ou r d a t a .
* T h e n , you can use these visual figures as inputs into your neural network and train it with backpropagation through time (BPTT).
* BPTT allows you to update weights in your neural network based on what you have learned from training data that has been fed into the system so far.
* You can also use BPTT to test out different networks or configurations without having to wait until
* The code is for a convolutional neural network that has 9 layers.
* The first layer is the input, which is an image of size (3, 3).
* The second layer is the convolutional layer, which has 128 neurons and 3 x 3 filters.
* The activation function for this layer is relu.
* The third layer is the feature map with 64 neurons and 3 x 3 filters.
* The activation function for this layer is also relu.
* The fourth layer has 32 neurons and 3 x 3 filters as well as an activation function of sigmoid.
* This means that it will take on a value between 0 and 1 depending on how much activity there was in each neuron in the previous layers.

8. TESTING

FRAME WORK

Researchers proposed a variety of image processing and object segmentation approaches for identifying and counting blood cells in blood sample pictures. Those approaches, on the other hand, are picture quality sensitive and rely on noise reduction approaches by pre-processing and emphasize essential stochastic gradient signals.

Furthermore, while image segmentation algorithms may identify diverse items in pictures, they cannot distinguish between objects of the same class. Those approaches cannot identify overlapping and fading cells in context.The proposed methodology uses the MSER (Maximally Stable Extremal Regions) and SURF [3] (Speeded Up Robust Features) for blob detection, detecting the white blood cells accurately and a Batch Normalization layer that is included inside the hidden layers of CNN to speed up the training and improve the accuracy in training and testing results

1. *Dataset Acquisition*

The original blood smear pictures for this proposed ap- proach were collected from the KAGGLE dataset to train and assess the suggested methodology.The images available in the dataset are colored images.

INPUT: Colored images of the size 400 x 400 pixels that are magnified 40 times using a microscope.

OUTPUT: White blood cell classified 240 x 320-pixel images.

1. *Blob detection*

In pictures, a blob [4] is anything that is regarded a huge item or anything bright in a dark backdrop. We may generalize it as a colony or a large object that is recognizable from its background. The algorithm should be able to create a mask

where, f(t) is the characteristic function of the region

* + Approximating the region

*m*\_*pq* = *xp yq f* (*x*, *y*)*dxdy*

¸

*q* = *Ap*

Σ2 = *A*Σ1 *AT*

Where, m is moment of the image p,q are rationalization points

x,y is the coordinates of the image

1. *CNN overview*

with the blob with the largest size as the WBC nucleus. It should also apply the mask to the original image and extract the Region of Interest (ROI). In this paper two such blob detection algorithms were implemented namely Maximally Stable Extremal Regions.

that are virtually identical across a large range of criteria. All Pixels that are less than or equal to a specific threshold are white, whereas those that are greater than or equal to that threshold are black. If we are shown a series of thresholded pictures, we will first see [5] a black image, followed by white dots corresponding to local intensity minima, which will get larger. These white specks will gradually blend together, resulting in a white picture. All extremal areas are the set of all linked components in the sequence. By fitting ellipses to

the areas, elliptical frames may be affixed to the MSERs.

The descriptions for such regions are maintained as features. All pixels within the MSER are either brighter (bright extremal regions) or darker (dark extremal regions) than all pixels on the MSER’s outer edge.

*2) Working:*

* + Perform a simple image luminance thresholding by sweeping the intensity threshold from white to black.
  + Segregate the connected components which are the ex- tremal regions.
  + Find when an extremal zone is stable for all thresholded images or when the relative growth of the square is at a local minimum.
  + The area above or below may correspond with the genuine area due to the image’s clear discrete character, where the area is still called stable maximally.
  + Border the region with an ellipse
  + Keep those regions descriptors as features

*3) Mathematical Description:*

* + Detecting extremal regions

*f* (*t*) = |*I* (*t*) − *I*0 |

1 ¸ *t* |*I* (*t*) − *I*0 |*dt* (1)

9. RESULTS

9.1 PERFORMANCE ANALYSIS AND RESULTS

The input images are colored (RGB) that are having a resolution of 400 x 400 pixels that are taken by zooming the blood smear 40 times using a microscope. The white blood cells (WBC) are bluish purple. A total of 18375 images were taken from the KAGGLE dataset from which 14700 were used for training the CNN and the rest 3675 were used for testing and validation. There are 14700 images used for training and they are given to the network as 100 epochs with a batch size of 115.

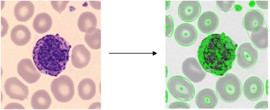


Fig. 3. Blob detection using MSER

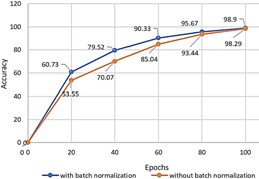
The above Fig. 3 represents the output of MSER blob detection algorithm. From the figure its clear that the MSER detects all the blobs that are extremally stable, out of which the WBC, the biggest blob is extracted.



Fig. 4. Region of Interest

In the above results, the Region of Interest (RoI) is extracted by masking the background in the original image. Thus, Fig. 4 shows the WBC (RoI) that is extracted from the detected blobs of the MSER algorithm.

The graph Fig.5 shows the performance of CNN with respect to accuracy as a function of number of epochs. It is clear that CNN with batch normalization [11] [12] [13] has better accuracy with less number of epochs compared to CNN without batch normalization. After 100 epochs, accuracy with batch normalization is 98.9 where as without batch normaliza- tion is 98.29. The above graph shows that the performance of CNN with respect to number of epochs as a function of test loss. It is clear that CNN with batch normalization has lower test loss compared to CNN without batch normalization. For 100 epochs, the test loss with and without batch normalization is obtained approximatly 0.38 and 0.59 respectively. From the results shown in Fig.6, it is clear that CNN with batch normalization has lesser test loss than CNN without batch normalization.



By using the trained network the fifteen different types of WBCs are tested seperately and there respective accuracy also displayed in the results. Fig.7 shows the results of the accuracy level between 98.50% to 100%.

A confusion matrix is used to determine if a categorization model is correct. There could be four possible outcomes, they are True Negative (TN), False-Negative (FN), False Positive (FP), and True Positive (TP). These outcomes will define the different performance measures, such as overall. accuracy, sensitivity, specificity, precision, F1 score, AUC, and IOU. Fig. 8 shows the Confusion Matrix for 15 different types of WBCs.

TABLE I

PERFORMANCE METRICS OF THE CLASSIFIER

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Blood cell | Overall accuracy | Precision | Recall | Specificity | F1 score | AUC | IOU |
| 1.Basophil | 99.646 | 40 | 16.666 | 99.918 | 23.529 | 28.333 | 13.333 |
| 2.Erythroblast | 99.619 | 50 | 21.428 | 99.918 | 29.999 | 35.714 | 17.647 |
| 3.Eosinophil | 99.265 | 83.333 | 74.626 | 99.722 | 78.740 | 78.980 | 64.935 |
| 4.Lymphocyte (typical) | 96.979 | 91.984 | 93.333 | 97.914 | 92.653 | 92.658 | 86.313 |
| 5.Monocyte | 96.734 | 84.554 | 84.114 | 98.207 | 84.334 | 84.334 | 72.911 |
| 6.Myeloblast | 96.244 | 87.063 | 92.438 | 97.059 | 89.670 | 89.751 | 81.275 |
| 7.Neutrophil (segmented) | 98.176 | 99.512 | 98.684 | 97.716 | 98.094 | 98.098 | 96.261 |
| Maximum classification metrics value | 99.646 | 99.512 | 98.684 | 99.918 | 98.094 | 98.098 | 96.261 |

TABLE II

COMPARISON OF EXISTING PAPERWORK[1] WITH PROPOSED WORK IN TERMS OF CLASSIFIER’S PERFORMANCE METRICS.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Methods | Overall accuracy | Precision | Recall | Specificity | F1 score | AUC | IOU |
| Existing method [1] | 97.34 | 99.41 | 95.84 | 94.89 | 97.59 | 97.56 | 95.30 |
| Proposed work | 99.64 | 99.51 | 98.68 | 99.91 | 98.09 | 98.09 | 96.26 |

10. ADVANTAGES AND DISADVANTAGES

CNN's are really effective for image classification as the concept of dimensionality reduction suits the huge number of parameters in an image. This write-up barely scratched the surface of CNNs here but provides a basic intuition on the above-stated fact.

the disadvantages of CNN models are:

* Classification of Images with different Positions
* Adversarial examples
* Coordinate Frame
* Other minor disadvantages like performance

These disadvantages lead to other models/ ideas like Capsule neural network. We have explained the points in depth.

11. CONCLUSION

This study presents an effective method for segmenting cancer cells using a new deep learning technique called kernel-based CNN with M-SVM and combined BWT and KSVM. The MRI image has been pre-processed using a combination of the LOG and CLAHE filtering methods during this procedure. SGLDM was then used to extract the features. M-SVM has categorized the MRI image of the cancer cells as either normal or abnormal based on the specific characteristics of the cancer cells cell. Finally, the proposed deep learning method successfully segmented the cancer cells from the MRI image to provide a treatment recommendation for the patient. For the purpose of cancer cells segmentation and classification, the CNN and M-SVM classifier have been utilized in this segmentation. The proposed deep learning algorithm can precisely segment cancer cells.

Using the K-SVM method and the feature vectors and cancer cells region, the BWT method was used to classify the cancer cells cell. The combined method of BWT and K-SVM was used to detect and classify cancer cells from cancer cells MRI images. The experimental results of these proposed approaches show that they will aid in the precise and timely identification of the cancer cells cell. As a result, these techniques can be utilized to detect cancer cells at an early stage, thereby preventing patient deaths.

The following is a summary of the contributions made to the existing body of knowledge in this thesis.

* The resulting method can be used to improve the segmentation and classification of cancer cells in order to detect them earlier, thereby preventing the patient's death. As a result, a brand-new and improved deep learning algorithm as well as a combination of BWT and K-SVM were made available for highly accurate cancer cells cell segmentation from an MRI image.
* By removing noises and adjusting the contrast of a given MRI image, the combined LOG and CLAHE method was used to improve image quality.
* A kernel-based CNN with M-SVM deep learning algorithm has been proposed to efficiently and automatically segment a cancer cells cell from an MRI image using a training set.
* In the combined BWT and K-SVM method, statistical features from the MRI image are extracted using the Berkeley Wavelet Transform, and the results are passed on to the Kernel Support Vector Machine to effectively classify the cancer cells cell as benign or malignant.
* The ongoing MATLAB software and a superfast computer are used to evaluate and simulate the proposed methods' performance. K-means Clustering, SVM, and conventional CNN were compared to the proposed methods in terms of accuracy, error rate, and time complexity.
* According to the comparison, the proposed deep learning method outperformed conventional segmentation methods for cancer cells cell segmentation by more than 90%.
* When compared to the methods that are currently in use, these proposed approaches required less time and complexity to segment and classify the cancer cells cell in MRI images.
* The improved deep learning algorithm proposed has a lower error rate for segmenting cancer cells than K-means clustering, SVM, and conventional CNN.
* • The proposed method combined BWT and K-SVM to create an effective cancer cells cell classification that outperformed conventional image classification methods with a maximum recognition rate of 95%.
* • The cancer cells cell was detected and classified as benign or malignant using the proposed segmentation and classification methods with high accuracy, low time complexity, and low error rate.

12. Future Work

* To implement these proposed segmentation methods in real-time cancer cells cell detection system.
* To extend these proposed methods for coloured images also
* To use these proposed segmentation methods in large dataset to evaluate the robustness of methods.
* This research work can be improved by applying the arithmetical transforms and mapped encoding techniques.

13. APPENDIX

SOURCE CODE:

**import** pandas **as** pd

**import** cv2

**import** numpy **as** np

**import** os

**from** random **import** shuffle

**from** tqdm **import** tqdm

**import** scipy

**import** skimage

**from** skimage.transform **import** resize

**from** sklearn.model\_selection **import** train\_test\_split

print(os**.**listdir("cancer/"))

['train', 'test', 'val', '.DS\_Store']

In [4]:

print(os**.**listdir("cancer/train"))

['Cancer', 'Normal']

In [5]:

TRAIN\_DIR **=** "cancer/train/"

TEST\_DIR **=** "cancer/test/"

In [6]:

*#Preprocessing*

**def** get\_label(Dir):

**for** nextdir **in** os**.**listdir(Dir):

**if** **not** nextdir**.**startswith('.'):

**if** nextdir **in** ['NORMAL']:

label **=** 0

**elif** nextdir **in** ['CANCER']:

label **=** 1

**else**:

label **=** 2

**return** nextdir, label

In [7]:

**def** preprocessing\_data(Dir):

X **=** []

y **=** []

**for** nextdir **in** os**.**listdir(Dir):

nextdir, label **=** get\_label(Dir)

temp **=** Dir **+** nextdir

**for** image\_filename **in** tqdm(os**.**listdir(temp)):

path **=** os**.**path**.**join(temp **+** '/' , image\_filename)

img **=** cv2**.**imread(path,cv2**.**IMREAD\_GRAYSCALE)

**if** img **is** **not** **None**:

img **=** skimage**.**transform**.**resize(img, (150, 150, 3))

img **=** np**.**asarray(img)

X**.**append(img)

y**.**append(label)

X **=** np**.**asarray(X)

y **=** np**.**asarray(y)

**return** X,y

In [7]:

*#X\_train, y\_train = preprocessing\_data(TRAIN\_DIR)*

In [8]:

**def** get\_data(Dir):

X **=** []

y **=** []

**for** nextDir **in** os**.**listdir(Dir):

**if** **not** nextDir**.**startswith('.'):

**if** nextDir **in** ['NORMAL']:

label **=** 0

**elif** nextDir **in** ['CANCER']:

label **=** 1

**else**:

label **=** 2

temp **=** Dir **+** nextDir

**for** file **in** tqdm(os**.**listdir(temp)):

img **=** cv2**.**imread(temp **+** '/' **+** file)

**if** img **is** **not** **None**:

img **=** skimage**.**transform**.**resize(img, (150, 150, 3))

*#img\_file = scipy.misc.imresize(arr=img\_file, size=(150, 150, 3))*

img **=** np**.**asarray(img)

X**.**append(img)

y**.**append(label)

X **=** np**.**asarray(X)

y **=** np**.**asarray(y)

**return** X,y

In [10]:

X\_train, y\_train **=** get\_data(TRAIN\_DIR)

100%|██████████| 2478/2478 [00:27<00:00, 23.64it/s]

100%|██████████| 2483/2483 [00:39<00:00, 63.38it/s]

In [11]:

X\_test , y\_test **=** get\_data(TEST\_DIR)

100%|██████████| 620/620 [00:17<00:00, 35.22it/s]

100%|██████████| 620/620 [00:15<00:00, 40.49it/s]

In [12]:

print(X\_train**.**shape,'\n',X\_test**.**shape)

(4961, 150, 150, 3)

(1240, 150, 150, 3)

In [13]:

print(y\_train**.**shape,'\n',y\_test**.**shape)

(4961,)

(1240,)

In [12]:

**from** keras.utils.np\_utils **import** to\_categorical

y\_train **=** to\_categorical(y\_train, 2)

y\_test **=** to\_categorical(y\_test, 2)

Using TensorFlow backend.

In [13]:

print(y\_train**.**shape,'\n',y\_test**.**shape)

(4961,)

(1240,)

In [14]:

Pimages **=** os**.**listdir(TRAIN\_DIR **+** "CANCER")

Nimages **=** os**.**listdir(TRAIN\_DIR **+** "NORMAL")

In [15]:

**import** matplotlib.pyplot **as** plt

**def** plotter(i):

imagep1 **=** cv2**.**imread(TRAIN\_DIR**+**"CANCER/"**+**Pimages[i])

imagep1 **=** skimage**.**transform**.**resize(imagep1, (150, 150, 3) , mode **=** 'reflect')

imagen1 **=** cv2**.**imread(TRAIN\_DIR**+**"NORMAL/"**+**Nimages[i])

imagen1 **=** skimage**.**transform**.**resize(imagen1, (150, 150, 3))

pair **=** np**.**concatenate((imagen1, imagep1), axis**=**1)

print("(Left) - No CANCER Vs (Right) - CANCER")

print("-----------------------------------------------------------------------------------------------------------------------------------")

plt**.**figure(figsize**=**(10,5))

plt**.**imshow(pair)

plt**.**show()

**for** i **in** range(0,5):

plotter(i)

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

In [16]:

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn **import** metrics

**from** sklearn.metrics **import** accuracy\_score

*#function*

**def** train\_test\_rmse(x,y):

x **=** Iris\_data[x]

y **=** Iris\_data[y]

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x, y, test\_size **=** 0.2,random\_state**=**123)

linreg **=** LinearRegression()

linreg**.**fit(X\_train, y\_train)

y\_pred **=** linreg**.**predict(X\_test)

print(accuracy\_score(y\_test, y\_pred)) *# or you can save it in variable and return it*

**return** np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_pred))

In [17]:

**import** seaborn **as** sns

count **=** y\_train**.**sum(axis **=** 0)

sns**.**countplot(x **=** count)

Out[17]:

****

In [18]:

**from** keras.callbacks **import** ReduceLROnPlateau , ModelCheckpoint

lr\_reduce **=** ReduceLROnPlateau(monitor**=**'val\_acc', factor**=**0.1, epsilon**=**0.0001, patience**=**1, verbose**=**1)

/home/neuzan/Programs/anaconda3/envs/DeepL/lib/python3.6/site-packages/keras/callbacks.py:1065: UserWarning: `epsilon` argument is deprecated and will be removed, use `min\_delta` instead.

warnings.warn('`epsilon` argument is deprecated and '

In [19]:

filepath**=**"weights.hdf5"

checkpoint **=** ModelCheckpoint(filepath, monitor**=**'val\_acc', verbose**=**1, save\_best\_only**=True**, mode**=**'max')

In [20]:

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense , Activation

**from** keras.layers **import** Dropout

**from** keras.layers **import** Flatten

**from** keras.constraints **import** maxnorm

**from** keras.optimizers **import** SGD , RMSprop

**from** keras.layers **import** Conv2D , BatchNormalization

**from** keras.layers **import** MaxPooling2D

**from** keras.utils **import** np\_utils

**from** keras **import** backend **as** K

K**.**set\_image\_dim\_ordering('th')

**from** sklearn.model\_selection **import** GridSearchCV

**from** keras.wrappers.scikit\_learn **import** KerasClassifier

In [21]:

*#X\_train=X\_train.reshape(5216,3,150,150)*

*#X\_test=X\_test.reshape(624,3,150,150)*

In [22]:

model **=** Sequential()

model**.**add(Conv2D(16, (3, 3), activation**=**'relu', padding**=**"same", input\_shape**=**(150,150,3)))

model**.**add(Conv2D(16, (3, 3), padding**=**"same", activation**=**'relu'))

model**.**add(Conv2D(32, (3, 3), activation**=**'relu', padding**=**"same"))

model**.**add(Conv2D(32, (3, 3), padding**=**"same", activation**=**'relu'))

model**.**add(Conv2D(64, (3, 3), activation**=**'relu', padding**=**"same"))

model**.**add(Conv2D(64, (3, 3), padding**=**"same", activation**=**'relu'))

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model**.**add(Flatten())

model**.**add(Dense(64, activation**=**'relu'))

model**.**add(Dropout(0.2))

model**.**add(Dense(2 , activation**=**'sigmoid'))

model**.**compile(loss**=**'binary\_crossentropy',

optimizer**=**RMSprop(lr**=**0.00005),

metrics**=**['accuracy'])

print(model**.**summary())

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

Layer (type) Output Shape Param #

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

conv2d\_1 (Conv2D) (None, 16, 150, 3) 21616

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

conv2d\_2 (Conv2D) (None, 16, 150, 3) 2320

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

conv2d\_3 (Conv2D) (None, 32, 150, 3) 4640

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

conv2d\_4 (Conv2D) (None, 32, 150, 3) 9248

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

conv2d\_5 (Conv2D) (None, 64, 150, 3) 18496

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

conv2d\_6 (Conv2D) (None, 64, 150, 3) 36928

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

max\_pooling2d\_1 (MaxPooling2 (None, 64, 75, 1) 0

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

flatten\_1 (Flatten) (None, 4800) 0

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

dense\_1 (Dense) (None, 64) 307264

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

dropout\_1 (Dropout) (None, 64) 0

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

dense\_2 (Dense) (None, 2) 130

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

Total params: 400,642

Trainable params: 400,642

Non-trainable params: 0

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

None

In [23]:

batch\_size **=** 256

epochs **=** 10

In [24]:

history **=** model**.**fit(X\_train, y\_train, validation\_data **=** (X\_test , y\_test) ,callbacks**=**[lr\_reduce,checkpoint] ,

epochs**=**epochs)

Train on 5216 samples, validate on 624 samples

Epoch 1/10

5216/5216 [==============================] - 12s 2ms/step - loss: 0.5063 - acc: 0.7597 - val\_loss: 0.4808 - val\_acc: 0.7780

Epoch 00001: val\_acc improved from -inf to 0.77804, saving model to weights.hdf5

Epoch 2/10

5216/5216 [==============================] - 6s 1ms/step - loss: 0.2925 - acc: 0.8792 - val\_loss: 0.6008 - val\_acc: 0.7252

Epoch 00002: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.

Epoch 00002: val\_acc did not improve from 0.77804

Epoch 3/10

5216/5216 [==============================] - 6s 1ms/step - loss: 0.2312 - acc: 0.9042 - val\_loss: 0.5019 - val\_acc: 0.7780

Epoch 00003: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-07.

Epoch 00003: val\_acc did not improve from 0.77804

Epoch 4/10

5216/5216 [==============================] - 6s 1ms/step - loss: 0.2249 - acc: 0.9077 - val\_loss: 0.4912 - val\_acc: 0.7821

Epoch 00004: val\_acc improved from 0.77804 to 0.78205, saving model to weights.hdf5

Epoch 5/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2243 - acc: 0.9097 - val\_loss: 0.4968 - val\_acc: 0.7796

Epoch 00005: ReduceLROnPlateau reducing learning rate to 4.999999987376214e-08.

Epoch 00005: val\_acc did not improve from 0.78205

Epoch 6/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2251 - acc: 0.9078 - val\_loss: 0.4975 - val\_acc: 0.7796

Epoch 00006: ReduceLROnPlateau reducing learning rate to 5.000000058430488e-09.

Epoch 00006: val\_acc did not improve from 0.78205

Epoch 7/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2224 - acc: 0.9094 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00007: ReduceLROnPlateau reducing learning rate to 4.999999969612646e-10.

Epoch 00007: val\_acc did not improve from 0.78205

Epoch 8/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2255 - acc: 0.9078 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00008: ReduceLROnPlateau reducing learning rate to 4.999999858590343e-11.

Epoch 00008: val\_acc did not improve from 0.78205

Epoch 9/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2243 - acc: 0.9085 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00009: ReduceLROnPlateau reducing learning rate to 4.999999719812465e-12.

Epoch 00009: val\_acc did not improve from 0.78205

Epoch 10/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2242 - acc: 0.9088 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00010: ReduceLROnPlateau reducing learning rate to 4.999999546340118e-13.

Epoch 00010: val\_acc did not improve from 0.78205

In [34]:

model**.**save('mymodel.h5')

In [27]:

**import** matplotlib.pyplot **as** plt

**from** keras.models **import** load\_model

plt**.**plot(history**.**history['acc'])

plt**.**plot(history**.**history['val\_acc'])

plt**.**title('model accuracy')

plt**.**ylabel('accuracy')

plt**.**xlabel('epoch')

plt**.**legend(['train', 'test'], loc**=**'upper left')

plt**.**show()

*# summarize history for loss*

plt**.**plot(history**.**history['loss'])

plt**.**plot(history**.**history['val\_loss'])

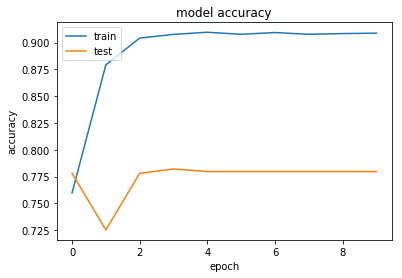
plt**.**title('model loss')

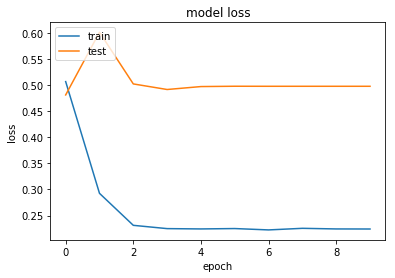
plt**.**ylabel('loss')

plt**.**xlabel('epoch')

plt**.**legend(['train', 'test'], loc**=**'upper left')

plt**.**show()

****

****

In [28]:

**from** sklearn.metrics **import** confusion\_matrix

pred **=** model**.**predict(X\_test)

pred **=** np**.**argmax(pred,axis **=** 1)

y\_true **=** np**.**argmax(y\_test,axis **=** 1)

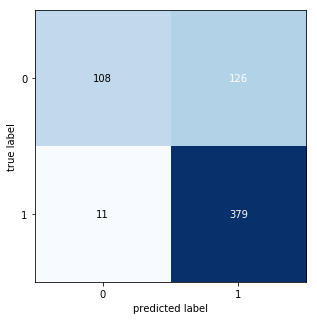
In [29]:

CM **=** confusion\_matrix(y\_true, pred)

**from** mlxtend.plotting **import** plot\_confusion\_matrix

fig, ax **=** plot\_confusion\_matrix(conf\_mat**=**CM , figsize**=**(5, 5))

plt**.**show()

****

In [16]:

*#PRECISION = (TP/(TP+FP))*

379**/**(379**+**126)

Out[16]:

0.7504950495049505

In [17]:

*#RECALL = (TP/(TP+FN))*

379 **/** (379 **+** 11)

Out[17]:

0.9717948717948718

In [18]:

*#ACCURACY = (TP+TN)/(TP+TN+FP+FN)*

(379**+**108)**/**(379**+**108**+**126**+**11)

Out[18]:

0.780448717948718

In [ ]:

**import** cv2

**import** numpy **as** np

**import** os

**from** random **import** shuffle

**from** tqdm **import** tqdm

**import** scipy

**import** skimage

**from** skimage.transform **import** resize

**from** sklearn.model\_selection **import** train\_test\_split

print(os**.**listdir("cancer/"))

['train', 'test', 'val', '.DS\_Store']

In [4]:

print(os**.**listdir("cancer/train"))

['Cancer', 'Normal']

In [5]:

TRAIN\_DIR **=** "cancer/train/"

TEST\_DIR **=** "cancer/test/"

In [6]:

*#Preprocessing*

**def** get\_label(Dir):

**for** nextdir **in** os**.**listdir(Dir):

**if** **not** nextdir**.**startswith('.'):

**if** nextdir **in** ['NORMAL']:

label **=** 0

**elif** nextdir **in** ['CANCER']:

label **=** 1

**else**:

label **=** 2

**return** nextdir, label

In [7]:

**def** preprocessing\_data(Dir):

X **=** []

y **=** []

**for** nextdir **in** os**.**listdir(Dir):

nextdir, label **=** get\_label(Dir)

temp **=** Dir **+** nextdir

**for** image\_filename **in** tqdm(os**.**listdir(temp)):

path **=** os**.**path**.**join(temp **+** '/' , image\_filename)

img **=** cv2**.**imread(path,cv2**.**IMREAD\_GRAYSCALE)

**if** img **is** **not** **None**:

img **=** skimage**.**transform**.**resize(img, (150, 150, 3))

img **=** np**.**asarray(img)

X**.**append(img)

y**.**append(label)

X **=** np**.**asarray(X)

y **=** np**.**asarray(y)

**return** X,y

In [7]:

*#X\_train, y\_train = preprocessing\_data(TRAIN\_DIR)*

In [8]:

**def** get\_data(Dir):

X **=** []

y **=** []

**for** nextDir **in** os**.**listdir(Dir):

**if** **not** nextDir**.**startswith('.'):

**if** nextDir **in** ['NORMAL']:

label **=** 0

**elif** nextDir **in** ['CANCER']:

label **=** 1

**else**:

label **=** 2

temp **=** Dir **+** nextDir

**for** file **in** tqdm(os**.**listdir(temp)):

img **=** cv2**.**imread(temp **+** '/' **+** file)

**if** img **is** **not** **None**:

img **=** skimage**.**transform**.**resize(img, (150, 150, 3))

*#img\_file = scipy.misc.imresize(arr=img\_file, size=(150, 150, 3))*

img **=** np**.**asarray(img)

X**.**append(img)

y**.**append(label)

X **=** np**.**asarray(X)

y **=** np**.**asarray(y)

**return** X,y

In [10]:

X\_train, y\_train **=** get\_data(TRAIN\_DIR)

100%|██████████| 2478/2478 [00:27<00:00, 23.64it/s]

100%|██████████| 2483/2483 [00:39<00:00, 63.38it/s]

In [11]:

X\_test , y\_test **=** get\_data(TEST\_DIR)

100%|██████████| 620/620 [00:17<00:00, 35.22it/s]

100%|██████████| 620/620 [00:15<00:00, 40.49it/s]

In [12]:

print(X\_train**.**shape,'\n',X\_test**.**shape)

(4961, 150, 150, 3)

(1240, 150, 150, 3)

In [13]:

print(y\_train**.**shape,'\n',y\_test**.**shape)

(4961,)

(1240,)

In [12]:

**from** keras.utils.np\_utils **import** to\_categorical

y\_train **=** to\_categorical(y\_train, 2)

y\_test **=** to\_categorical(y\_test, 2)

Using TensorFlow backend.

In [13]:

print(y\_train**.**shape,'\n',y\_test**.**shape)

(4961,)

(1240,)

In [14]:

Pimages **=** os**.**listdir(TRAIN\_DIR **+** "CANCER")

Nimages **=** os**.**listdir(TRAIN\_DIR **+** "NORMAL")

In [15]:

**import** matplotlib.pyplot **as** plt

**def** plotter(i):

imagep1 **=** cv2**.**imread(TRAIN\_DIR**+**"CANCER/"**+**Pimages[i])

imagep1 **=** skimage**.**transform**.**resize(imagep1, (150, 150, 3) , mode **=** 'reflect')

imagen1 **=** cv2**.**imread(TRAIN\_DIR**+**"NORMAL/"**+**Nimages[i])

imagen1 **=** skimage**.**transform**.**resize(imagen1, (150, 150, 3))

pair **=** np**.**concatenate((imagen1, imagep1), axis**=**1)

print("(Left) - No CANCER Vs (Right) - CANCER")

print("-----------------------------------------------------------------------------------------------------------------------------------")

plt**.**figure(figsize**=**(10,5))

plt**.**imshow(pair)

plt**.**show()

**for** i **in** range(0,5):

plotter(i)

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

(Left) - No CANCER Vs (Right) - CANCER

-----------------------------------------------------------------------------------------------------------------------------------

In [16]:

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn **import** metrics

**from** sklearn.metrics **import** accuracy\_score

*#function*

**def** train\_test\_rmse(x,y):

x **=** Iris\_data[x]

y **=** Iris\_data[y]

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x, y, test\_size **=** 0.2,random\_state**=**123)

linreg **=** LinearRegression()

linreg**.**fit(X\_train, y\_train)

y\_pred **=** linreg**.**predict(X\_test)

print(accuracy\_score(y\_test, y\_pred)) *# or you can save it in variable and return it*

**return** np**.**sqrt(metrics**.**mean\_squared\_error(y\_test, y\_pred))

In [17]:

**import** seaborn **as** sns

count **=** y\_train**.**sum(axis **=** 0)

sns**.**countplot(x **=** count)

Out[17]:



In [18]:

**from** keras.callbacks **import** ReduceLROnPlateau , ModelCheckpoint

lr\_reduce **=** ReduceLROnPlateau(monitor**=**'val\_acc', factor**=**0.1, epsilon**=**0.0001, patience**=**1, verbose**=**1)

/home/neuzan/Programs/anaconda3/envs/DeepL/lib/python3.6/site-packages/keras/callbacks.py:1065: UserWarning: `epsilon` argument is deprecated and will be removed, use `min\_delta` instead.

warnings.warn('`epsilon` argument is deprecated and '

In [19]:

filepath**=**"weights.hdf5"

checkpoint **=** ModelCheckpoint(filepath, monitor**=**'val\_acc', verbose**=**1, save\_best\_only**=True**, mode**=**'max')

In [20]:

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense , Activation

**from** keras.layers **import** Dropout

**from** keras.layers **import** Flatten

**from** keras.constraints **import** maxnorm

**from** keras.optimizers **import** SGD , RMSprop

**from** keras.layers **import** Conv2D , BatchNormalization

**from** keras.layers **import** MaxPooling2D

**from** keras.utils **import** np\_utils

**from** keras **import** backend **as** K

K**.**set\_image\_dim\_ordering('th')

**from** sklearn.model\_selection **import** GridSearchCV

**from** keras.wrappers.scikit\_learn **import** KerasClassifier

In [21]:

*#X\_train=X\_train.reshape(5216,3,150,150)*

*#X\_test=X\_test.reshape(624,3,150,150)*

In [22]:

model **=** Sequential()

model**.**add(Conv2D(16, (3, 3), activation**=**'relu', padding**=**"same", input\_shape**=**(150,150,3)))

model**.**add(Conv2D(16, (3, 3), padding**=**"same", activation**=**'relu'))

model**.**add(Conv2D(32, (3, 3), activation**=**'relu', padding**=**"same"))

model**.**add(Conv2D(32, (3, 3), padding**=**"same", activation**=**'relu'))

model**.**add(Conv2D(64, (3, 3), activation**=**'relu', padding**=**"same"))

model**.**add(Conv2D(64, (3, 3), padding**=**"same", activation**=**'relu'))

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model**.**add(Flatten())

model**.**add(Dense(64, activation**=**'relu'))

model**.**add(Dropout(0.2))

model**.**add(Dense(2 , activation**=**'sigmoid'))

model**.**compile(loss**=**'binary\_crossentropy',

optimizer**=**RMSprop(lr**=**0.00005),

metrics**=**['accuracy'])

print(model**.**summary())

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

Layer (type) Output Shape Param #

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

conv2d\_1 (Conv2D) (None, 16, 150, 3) 21616

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

conv2d\_2 (Conv2D) (None, 16, 150, 3) 2320

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

conv2d\_3 (Conv2D) (None, 32, 150, 3) 4640

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

conv2d\_4 (Conv2D) (None, 32, 150, 3) 9248

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

conv2d\_5 (Conv2D) (None, 64, 150, 3) 18496

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

conv2d\_6 (Conv2D) (None, 64, 150, 3) 36928

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

max\_pooling2d\_1 (MaxPooling2 (None, 64, 75, 1) 0

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

flatten\_1 (Flatten) (None, 4800) 0

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

dense\_1 (Dense) (None, 64) 307264

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

dropout\_1 (Dropout) (None, 64) 0

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

dense\_2 (Dense) (None, 2) 130

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

Total params: 400,642

Trainable params: 400,642

Non-trainable params: 0

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

None

In [23]:

batch\_size **=** 256

epochs **=** 10

In [24]:

history **=** model**.**fit(X\_train, y\_train, validation\_data **=** (X\_test , y\_test) ,callbacks**=**[lr\_reduce,checkpoint] ,

epochs**=**epochs)

Train on 5216 samples, validate on 624 samples

Epoch 1/10

5216/5216 [==============================] - 12s 2ms/step - loss: 0.5063 - acc: 0.7597 - val\_loss: 0.4808 - val\_acc: 0.7780

Epoch 00001: val\_acc improved from -inf to 0.77804, saving model to weights.hdf5

Epoch 2/10

5216/5216 [==============================] - 6s 1ms/step - loss: 0.2925 - acc: 0.8792 - val\_loss: 0.6008 - val\_acc: 0.7252

Epoch 00002: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.

Epoch 00002: val\_acc did not improve from 0.77804

Epoch 3/10

5216/5216 [==============================] - 6s 1ms/step - loss: 0.2312 - acc: 0.9042 - val\_loss: 0.5019 - val\_acc: 0.7780

Epoch 00003: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-07.

Epoch 00003: val\_acc did not improve from 0.77804

Epoch 4/10

5216/5216 [==============================] - 6s 1ms/step - loss: 0.2249 - acc: 0.9077 - val\_loss: 0.4912 - val\_acc: 0.7821

Epoch 00004: val\_acc improved from 0.77804 to 0.78205, saving model to weights.hdf5

Epoch 5/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2243 - acc: 0.9097 - val\_loss: 0.4968 - val\_acc: 0.7796

Epoch 00005: ReduceLROnPlateau reducing learning rate to 4.999999987376214e-08.

Epoch 00005: val\_acc did not improve from 0.78205

Epoch 6/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2251 - acc: 0.9078 - val\_loss: 0.4975 - val\_acc: 0.7796

Epoch 00006: ReduceLROnPlateau reducing learning rate to 5.000000058430488e-09.

Epoch 00006: val\_acc did not improve from 0.78205

Epoch 7/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2224 - acc: 0.9094 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00007: ReduceLROnPlateau reducing learning rate to 4.999999969612646e-10.

Epoch 00007: val\_acc did not improve from 0.78205

Epoch 8/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2255 - acc: 0.9078 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00008: ReduceLROnPlateau reducing learning rate to 4.999999858590343e-11.

Epoch 00008: val\_acc did not improve from 0.78205

Epoch 9/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2243 - acc: 0.9085 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00009: ReduceLROnPlateau reducing learning rate to 4.999999719812465e-12.

Epoch 00009: val\_acc did not improve from 0.78205

Epoch 10/10

5216/5216 [==============================] - 7s 1ms/step - loss: 0.2242 - acc: 0.9088 - val\_loss: 0.4974 - val\_acc: 0.7796

Epoch 00010: ReduceLROnPlateau reducing learning rate to 4.999999546340118e-13.

Epoch 00010: val\_acc did not improve from 0.78205

In [34]:

model**.**save('mymodel.h5')

In [27]:

**import** matplotlib.pyplot **as** plt

**from** keras.models **import** load\_model

plt**.**plot(history**.**history['acc'])

plt**.**plot(history**.**history['val\_acc'])

plt**.**title('model accuracy')

plt**.**ylabel('accuracy')

plt**.**xlabel('epoch')

plt**.**legend(['train', 'test'], loc**=**'upper left')

plt**.**show()

*# summarize history for loss*

plt**.**plot(history**.**history['loss'])

plt**.**plot(history**.**history['val\_loss'])

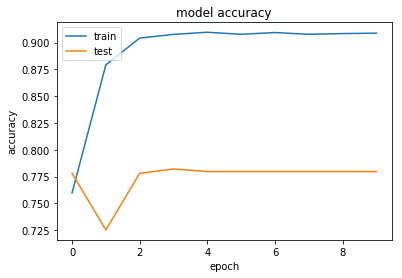
plt**.**title('model loss')

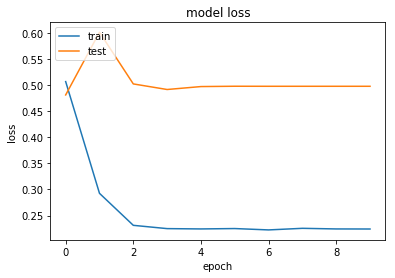
plt**.**ylabel('loss')

plt**.**xlabel('epoch')

plt**.**legend(['train', 'test'], loc**=**'upper left')

plt**.**show()





In [28]:

**from** sklearn.metrics **import** confusion\_matrix

pred **=** model**.**predict(X\_test)

pred **=** np**.**argmax(pred,axis **=** 1)

y\_true **=** np**.**argmax(y\_test,axis **=** 1)

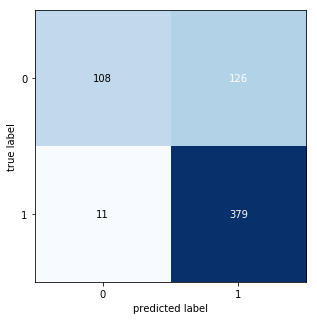
In [29]:

CM **=** confusion\_matrix(y\_true, pred)

**from** mlxtend.plotting **import** plot\_confusion\_matrix

fig, ax **=** plot\_confusion\_matrix(conf\_mat**=**CM , figsize**=**(5, 5))

plt**.**show()



In [16]:

*#PRECISION = (TP/(TP+FP))*

379**/**(379**+**126)

Out[16]:

0.7504950495049505

In [17]:

*#RECALL = (TP/(TP+FN))*

379 **/** (379 **+** 11)

Out[17]:

0.9717948717948718

In [18]:

*#ACCURACY = (TP+TN)/(TP+TN+FP+FN)*

(379**+**108)**/**(379**+**108**+**126**+**11)

Out[18]:

0.780448717948718

GITHUB AND DEMO LINK:

https://github.com/IBM-EPBL/IBM-Project-5647-1658812562