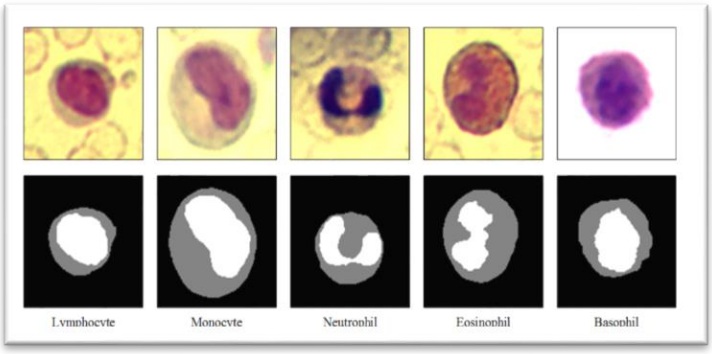
PATTERN RECOGNITION

Ahmed Ragabb (4180597)
Ahmed Taher
Youssef Moataz

Problem Definition

The project objective is to use the segmentation techniques of machine learning and deep learning to segment different colored white blood cells (WBC) images into five different categories **(Lymphocyte – Monocyte – Neutrophil – Eosinophil – Basophil),** using a dataset that contains different images with different conditions to train our model perfectly. The nuclei, cytoplasm and background including red blood cells are marked in white, gray and black respectively.



To solve the problem we used Three Techniques procedures:

- C-means Clustering algorithm
- Multi-class shallow neural network
- Support Vector Machine.

Importance:

In clinical practice, the identification and counting of WBCs in blood smear are often used for diagnosing many diseases such as in-fictions, inflammation, malignancy, leukemia. In the past, the ex-amination of blood smears is a highly complex, tedious, and time-consuming manual task. Nowadays, with the rapid development of computer-aided methods, an automatic cell analysis system can support faster and more reproducible image analysis than manual analysis.

Algorithms & Methods

C- Means Clustering:

Step 1: initialize weights:

Assume there are 2 clusters of black and white. Each data point will be assigned a random weight between 0 and 1 for which weight of data point x for black + weight for x for white = 1. For example:

	Black	White
X	W1=0.73	W2=0.27

Step 2: initialize and form the centroids:

The centroid formula is: $$V_{ij} = (\sum_{1}^{n}(\gamma_{ik}^{m} * x_k) / \sum_{1}^{n} \gamma_{ik}^{m}$$

Where, μ is fuzzy membership value of the data point, m is the fuzziness parameter (generally taken as 2), and x_k is the data point.

Step 3: Calculate the distance between each data point and the centroid:

Dki, Dkj where the distance is the distance between each data point from 1 to k to the cluster center.

Step 4: Update membership matrix:

Formula:

$$\gamma = \sum_{1}^{n} (d_{ki}^2 / d_{kj}^2)^{1/m-1}]^{-1}$$

$$J(U,V) = \sum_{i=1}^{n}\sum_{j=1}^{c} (\mu_{ij})^m \left\|x_i - v_j\right\|^2$$

By making the loop to be repeating the steps until this optimizing equation (which is the difference between the old membership of a previous iteration and the new) is nearly zero.

Step 5: **Defuzzify the obtained membership values and segment the picture for visualization.**

Shallow Neural Network:

The objective here is to use the deep learning (Neural Networks) and train a model using a shallow NN "One or two hidden layers Network" to classify the WBC images into Five different Categories.

Loading the Dataset: → **Used (Os and openCV libraries) to load it.**

-Loading the images was kind of an issue since I found some images with different dimensions than the (120x120). By iterating over the dataset I managed to start resizing the interrupted samples by padding it or resizing to get all of same size before feeding it to the model using cv2.copyMakeBorder() function.

After finishing organizing and cleaning the dataset I started to implement the Neural Network model Using Tensor Flow.

The Neural Network: → **Using Tensor Flow**

To build a one hidden layer network "Shallow" in tensor flow, I called few functions:

- **Preprocessing. Rescaling ()** → used to scale the images by dividing over 255 which indicated the maximum intensity level possible for each pixel.

- **Conv2D ()** → this method receives the convolution kernel size, the number of perceptron and the activation function I used the **(RELU)** Function.
    - The concept of convolution is to generate special features from the image by iterating over the image matrix with a kernel of specified size.
- **Dense ()** → the output layer to classify the image to one of the five classes using the Softmax Function to do so.

    - The softmax function extends the idea of logistic regression to produce a likelihood probabilities indicates how far the image to each class or category is.

Display image of each WBC

The file is loaded labels.csv into a data frame called labels, where the ID is the image name and the label column tells us the WBC volor.

The function get_image converts an ID value from the dataframe into a file path where the image is located, opens the image using the object in Pillow, and then returns the image as a numpy array.

Image manipulation with rgb2grey

Image data is represented as a matrix, where the depth is the number of channels. An RGB image has three channels (red, green, and blue) whereas the returned greyscale image has only one channel. Accordingly, the original color image has the dimensions 100x100x3 but after calling rgb2grey, the resulting greyscale image has only one channel, making the dimensions 100x100x1.

Histogram of oriented gradients

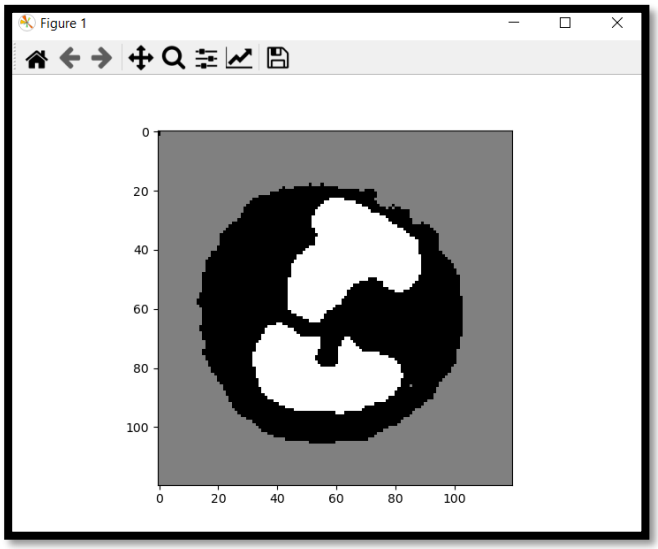
The images are converted into a format that a machine learning algorithm can understand.

An image is divided in a grid fashion into cells, and for the pixels within each cell, a histogram of gradient directions is compiled. To improve invariance to highlights and shadows in an image, cells are block normalized, meaning an intensity value is calculated for a larger region of an image called a block and used to contrast normalize all cell-level histograms within each block. The HOG feature vector for the image is the concatenation of these cell-level histograms.

Create image features and flatten into a single row

A function called create_features that combines these two sets of features by flattening the three-dimensional array into a one-dimensional (flat) array.

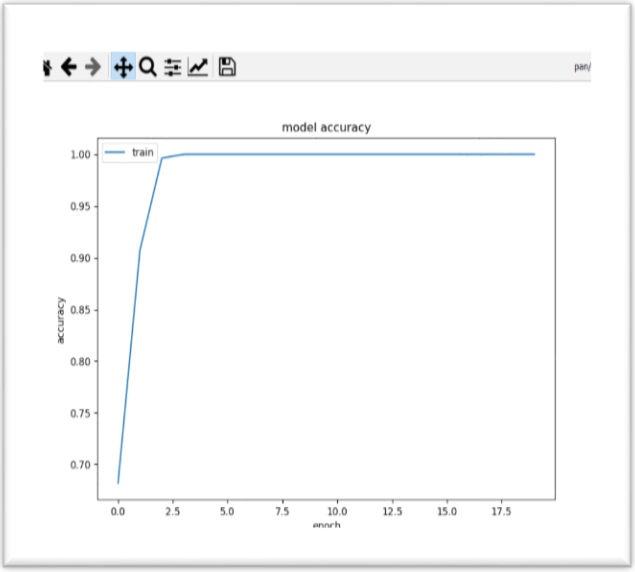
Project Results

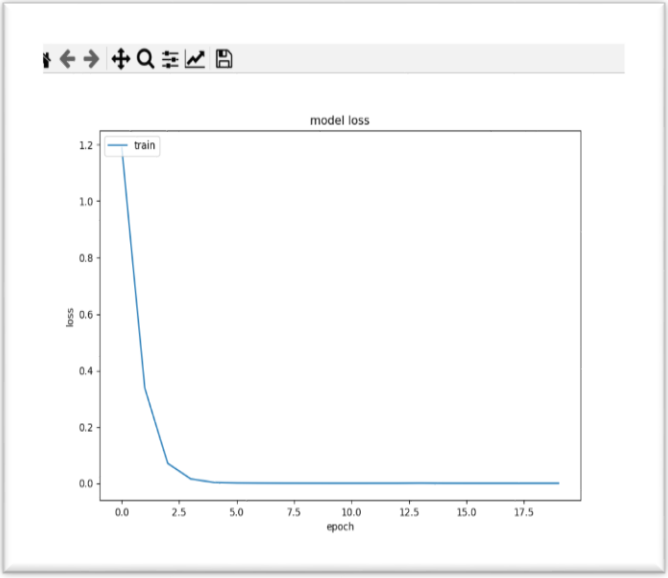


Discussion:

The concept of C- means clustering algorithm is each point has a weighting associated with a particular cluster, so a point doesn't sit "in a cluster" as much as has a weak or strong association to the cluster.

As Shown in the two figures The one on the right is the input image fed to the model to be clustered into different regions, and the one on the left is the resulted output of the model it shows how the clustering is done by dividing the image into 3 different regions (the nuclei, cytoplasm and RBCs) each got a specified grey level the background grey is for red blood cells and the white is for the nuclei.





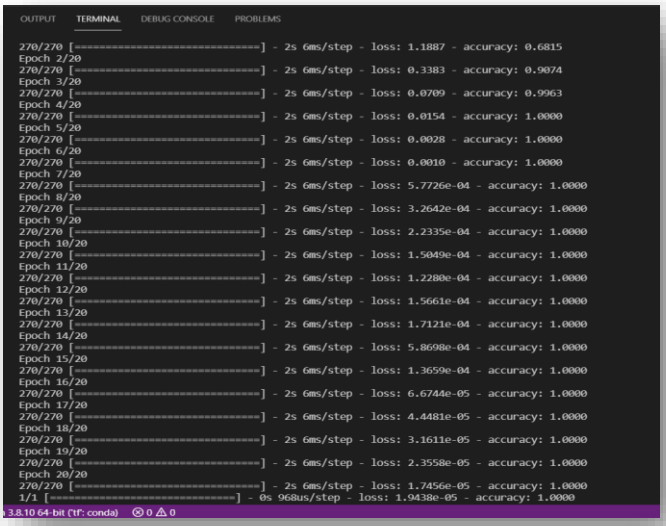
Discussion:

As shown in the two figures the model accuracy kept increasing until nearly 100%, and the loss dropped to nearly zero after using 20 epochs to optimize them, but to achieve such an accuracy and loss, I had to shuffle the dataset since when I fed the images in order to the model the accuracy was around 56% so I shuffled them to train the model instead of just initializing the inputs with images of the same class which affects the training procedure.

As Shown the process of optimizing the accuracy and loss:

- Accuracy was 100% since the 3rd epoch.
- The last line shows:

  Testing data accuracy and loss.



```
Traing_data_labels = []     ### True Y values for training examples
Testing_data_labels = []    ### True y For testing examples

##### Data as pairs "image,label"

Full_data = []
paired_train_data = []
paired_test_data = []

####  Counters To correctly adding the data
counnter = 0
counnter1 = 0

####
#
#   The for loop is used to create the different lists of the images in the dataset foldar.
#   prepairing the images was the main problem i faced and i did alot of processing on the images.
#   First i started to pad the images and resize them to match the 120x120 Dimension
#
####

for i in range(len(grey_images)):

    if i > (len(grey_images)*.9 - 1):
        image_path = 'C:/Users/Engah/Desktop/Pattern/segmentation_WBC-master/Dataset 1/'+ grey_images[randomlist[counnter]]
        image = cv2.imread(image_path,cv2.IMREAD_GRAYSCALE)
        delta_w = 120 - image.shape[1]
        delta_h = 120 - image.shape[0]
        top, bottom = delta_h//2, delta_h-(delta_h//2)
        left, right = delta_w//2, delta_w-(delta_w//2)
        image = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_CONSTANT,value=0)
        print(image.shape)
        Testing_data_labels.append(labels[randomlist[counnter]][0] -1)
        Testing_Data[counnter] = image
        Full_data.append([image,labels[counnter]])
        paired_test_data.append([image,labels[counnter]])
        counnter+=1
```

```
### Creating Lists of Images names in the foldar
for i in range(len(entries)):
    if entries[i].endswith('bmp'):
        colored_images.append(entries[i])
    else:
        grey_images.append(entries[i])

image_path = 'C:/Users/Engah/Desktop/Pattern/segmentation_WBC-master/Dataset 1/'+ colored_images[1]


#### Importing the Labels of the images from the csv file

a=pd.read_csv('C:/Users/Engah/Desktop/Pattern/segmentation_WBC-master/Class Labels of Dataset 1.csv')
b = a['class label'].tolist()
aa = np.array(b)
labels =aa.reshape(300,1)

##################

testing_size = len(grey_images)*.1  #### 30 testing image

Training_Data = np.zeros((270,120,120))
Testing_Data = np.zeros((30,120,120))

### Classes

classess = np.array([1,2,3,4,5])
Traing_data_labels = []     ### True Y values for training examples
Testing_data_labels = []    ### True y For testing examples

##### Data as pairs "image,label"

Full_data = []
paired_train_data = []
paired_test_data = []

####  Counters To correctly adding the data
```

```
#   The for loop is used to create the different lists of the images in the dataset foldar.
#   prepairing the images was the main problem i faced and i did alot of processing on the images.
#   First i started to pad the images and resize them to match the 120x120 Dimension
#
####

for i in range(len(grey_images)):

    if i > (len(grey_images)*.9 - 1):
        image_path = 'C:/Users/Engah/Desktop/Pattern/segmentation_WBC-master/Dataset 1/'+ grey_images[randomlist[counnter]]
        image = cv2.imread(image_path,cv2.IMREAD_GRAYSCALE)
        delta_w = 120 - image.shape[1]
        delta_h = 120 - image.shape[0]
        top, bottom = delta_h//2, delta_h-(delta_h//2)
        left, right = delta_w//2, delta_w-(delta_w//2)
        image = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_CONSTANT,value=0)
        print(image.shape)
        Testing_data_labels.append(labels[randomlist[counnter]][0] -1)
        Testing_Data[counnter] = image
        Full_data.append([image,labels[counnter]])
        paired_test_data.append([image,labels[counnter]])
        counnter+=1
    else:
        image_path = 'C:/Users/Engah/Desktop/Pattern/segmentation_WBC-master/Dataset 1/'+ grey_images[randomlist[i]]
        image = cv2.imread(image_path,cv2.IMREAD_GRAYSCALE)
        delta_w = 120 - image.shape[1]
        delta_h = 120 - image.shape[0]
        top, bottom = delta_h//2, delta_h-(delta_h//2)
        left, right = delta_w//2, delta_w-(delta_w//2)
        image = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_CONSTANT,value=0)
        print(image.shape)
        Training_Data[i] = image
        Traing_data_labels.append(labels[randomlist[i]][0]-1 )
        Full_data.append([image,labels[i]])
        paired_train_data.append([image,labels[i]])
        counnter1 += 1
```

```
X = Training_Data
Y= []
X = np.array(X)

for i in range(len(Traing_data_labels)):
    Y.append(int(Traing_data_labels[i]))

X = np.array(X).reshape(-1,120,120,1)
Y = np.array(Y)

#######################################################

num_classes = 5

########## Creating The Network ##############

model = Sequential(
[
    layers.experimental.preprocessing.Rescaling(1./255, input_shape=(120, 120, 1)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    # layers.MaxPooling2D(),            ##### adds one more hidden layer
    layers.Flatten(),
    layers.Dense(num_classes, activation=tf.nn.softmax)
]
)

model.compile(optimizer='Adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history=model.fit(X, Y, batch_size = 1, epochs=20)


########## Prepairing Testing Data ##############

X_test = Testing_Data
Y_test= []
X_test = np.array(X_test)
```

```
class ccluster:
    def __init__(self,image,image_bit,noclusters=2,fuzziness=2,max_iterations=150,epsilon=sys.float_info.epsilon):
        ##Give initial values to the parameters needed for the clustering

        self.noclus=noclusters
        self.fuzziness = fuzziness
        self.max_iterations = max_iterations
        self.image_bit = image_bit
        self.result=image
        self.shape = image.shape
        self.X = image.flatten().astype('float') # flatted image shape: (number of pixels,1)
        self.numPixels = image.size
        self.epsilon=epsilon


        #--------------------Check inputs------------------
        if np.ndim(image) != 2:
            raise Exception("<image> needs to be 2D (gray scale image).")
        if noclusters <= 0 or noclusters != int(noclusters):
            raise Exception("<noclusters> needs to be positive integer.")
        if fuzziness < 1:
            raise Exception("<m> needs to be >= 1.")
        if epsilon <= 0:
            raise Exception("<epsilon> needs to be > 0")


    def initializeMembershipMatrix(self):
        ## function that itializes the memberships
        n=self.numPixels
        nn=self.noclus
        membership_mat = np.random.random((nn, n))
        ## Intializes the omegas with a random values of random floats
        value = sum(membership_mat)
        membership_mat=np.divide(membership_mat,np.dot(np.ones((nn,1)),np.reshape(value,(1,n))))
        ## Normalize the omega values
```

```
    return membership_mat

def update_membership(self):
    ## Function that updates the membership weights based on the update of the clustering centeroids
    # function will be called iteratively
    '''Compute weights'''
    c_mesh,idx_mesh = np.meshgrid(self.noclus,self.X)
    m=self.fuzziness
    power = 2./(m-1) ## Fuzzy value power factor
    distance = abs(idx_mesh-c_mesh) ##compute the distance between the centers and the data values
    p1 = distance**power ## computation of the numerator of the weight update using the fuzzy c-means formula
    p2 = np.sum((1./distance)**power,axis=1) ## computation of the denomantor ....

    return 1./(p1*p2[:,None]) ##return updated value

def update_clusters(self):

    ## function that computes the centroids of the clusters
    denominator = np.sum(self.U**self.fuzziness,axis=0)## .... ^fuzzy number
    numerator = np.dot(self.X,self.U**self.fuzziness) ## the summation of weights[i,j]^fuzzy number * xi

    return numerator/denominator

def form_clusters(self):
# Function that loops and form clusters at each iteration while optimizing the error function

    d = 100
    self.U = np.transpose(self.initializeMembershipMatrix())
    ## We want to make the rows as the data and columns as number of clusters
    if self.max_iterations != -1:
        i = 0
        while True:
        #Loop that updates the clusters and the membership values untill the optimization function is minimum
            self.noclus = self.update_clusters()
            old_u = np.copy(self.U)
            self.U = self.update_membership()
```

```
            self.U = self.update_membership()
            d = np.sum(abs(self.U - old_u))
            print("Iteration %d : cost = %f" %(i, d))

            if d < self.epsilon or i > self.max_iterations:
                break
            i+=1
    else:
        i = 0
        while d > self.epsilon:
            self.C = self.update_C()
            old_u = np.copy(self.U)
            self.U = self.update_U()
            d = np.sum(abs(self.U - old_u))
            print("Iteration %d : cost = %f" %(i, d))

            if d < self.epsilon or i > self.max_iterations:
                break
            i+=1
    return self.segmentImage()


def deFuzzify(self):
    ## to turn the float of the cluster into binary numbers with number of clusters categorization
    return np.argmax(self.U, axis = 1)

def segmentImage(self):
    '''Segment image based on max weights'''
    ## defuzzify the weights and reshape for visualization
    result = self.deFuzzify()
    self.result = result.reshape(self.shape).astype('float')

    return self.result
```

```
import cv2
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename # Open dialog box
import sys
import ccluster


try:
            #--------------Lord image file--------------
            folder = askopenfilename(filetypes=[("images","*.*")])
            img= cv2.imread(folder,cv2.IMREAD_GRAYSCALE) # cf. 8bit image-> 0~255
            h,ch = img.shape

            #--------------Clustering--------------
            ## Take a cluster object from class ccluster
            cluster = ccluster.ccluster(img,image_bit=h,noclusters=3,fuzziness=2,max_iterations=80,epsilon=sys.float_info.ep
            ## Return the result of the fuzzy clustering on the image to result
            result=cluster.form_clusters()

            print("result is",result)

            #------------------Plot and save result-----------------------

            ## Show the result of clustering the chosen picture
            plt.imshow(result,cmap='gray')
            plt.show()
```