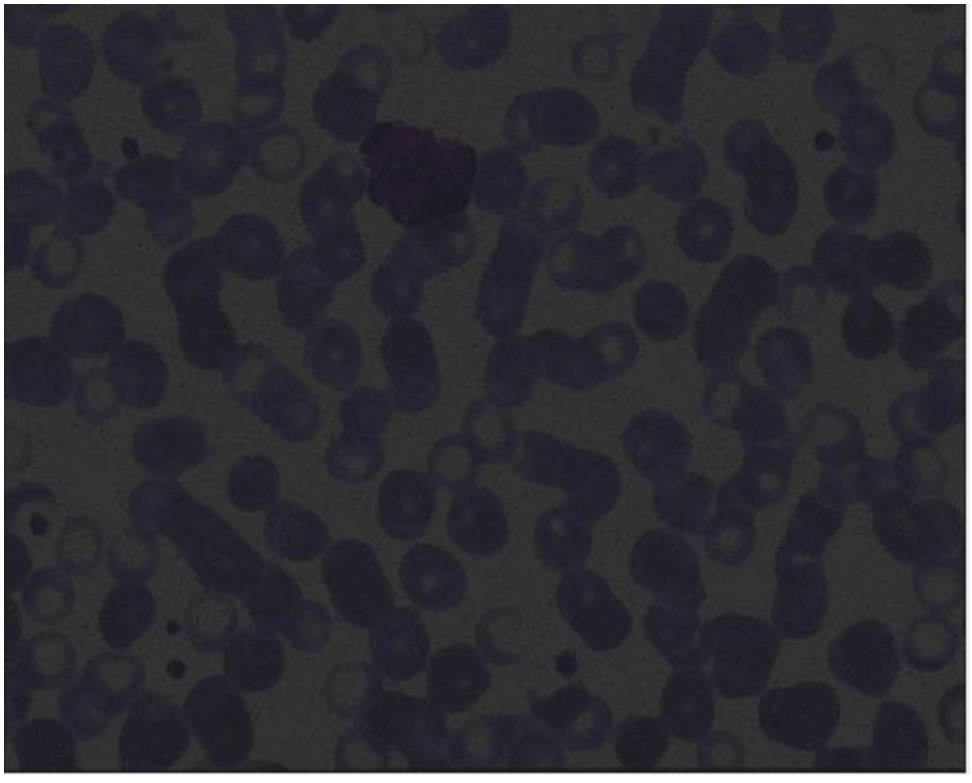
Part 1: Extraction of White Blood Cells

1) Iterate through the folders to process each image

```
Folder = dir('output_pre\');
Folder2 = dir('output\');
totalSimilarity=0;
total=0;
for j = 1 : 5
    File = dir(strcat('output_pre\',int2str(j),'\*.bmp'));
    File2 = dir(strcat('output\',int2str(j),'\*.bmp'));
    for i = 1 : length(File)
        filename2 = strcat('output\',int2str(j),'\',int2str(i),'.bmp');
        groundTruth =  imread(filename2);
        filename = strcat('output_pre\',int2str(j),'\',int2str(i),'.bmp');
        img = imread(filename);
```

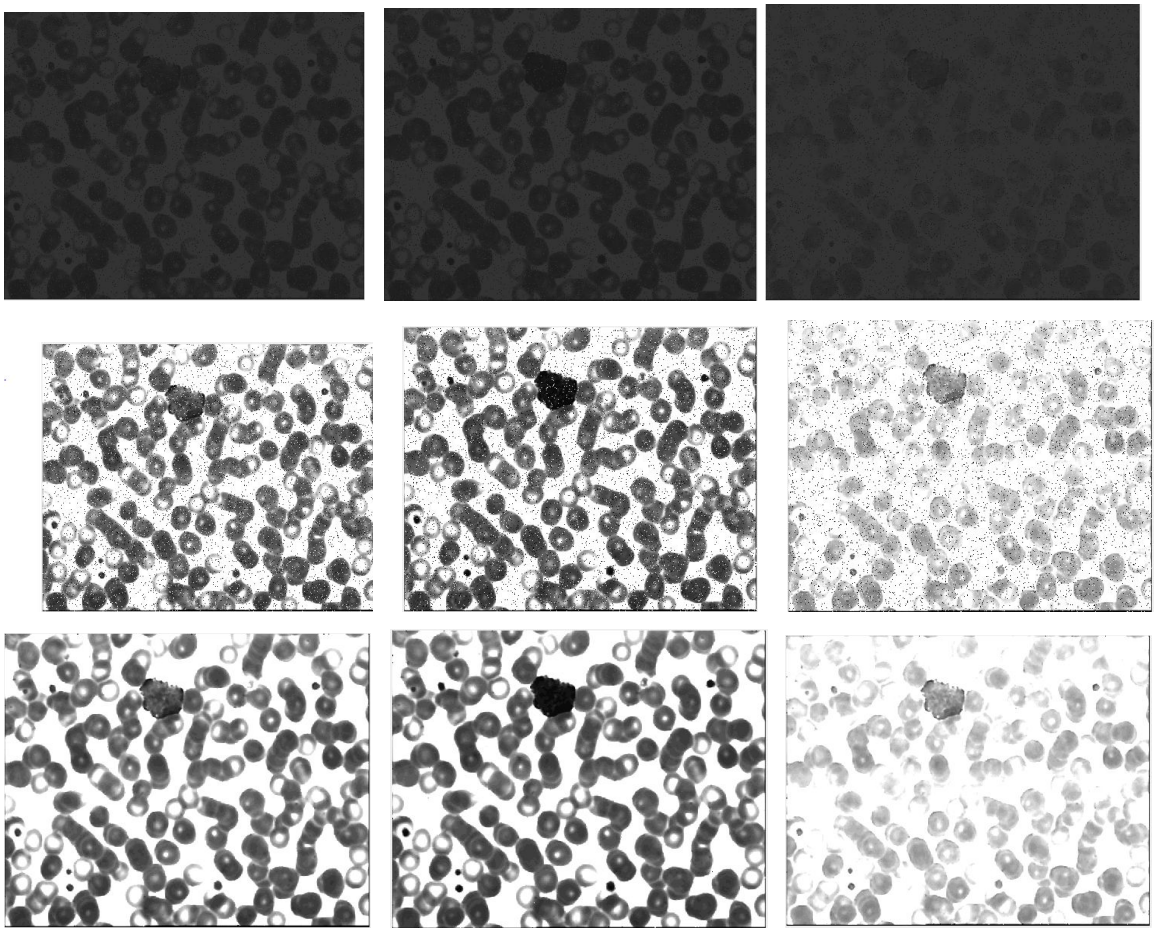
2) Example: Fist Basophil RGB image



3) As we can see the image suffers from low contrast and very noisy so we will split the image into the 3 channels (R, G, B) and apply imadjust function and median filter to each channel

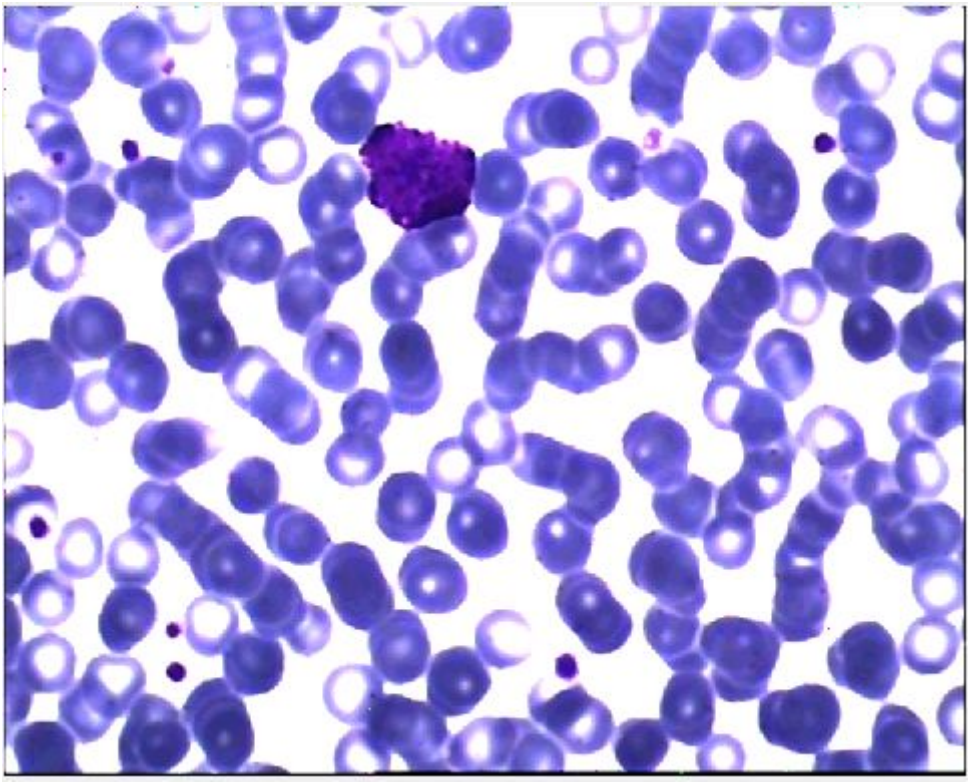
```
R = imadjust(img(:,:,1));
R = medfilt2(R);
G = imadjust(img(:,:,2));
G = medfilt2(G);
B = imadjust(img(:,:,3));
B = medfilt2(B);
```

R G B



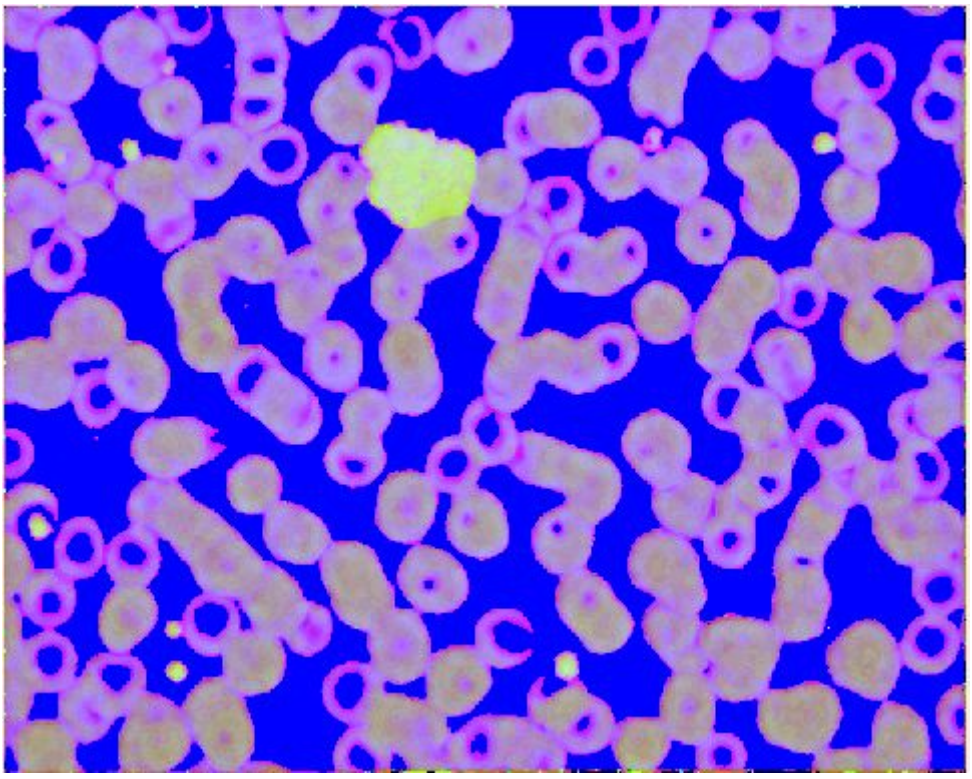
4) Concatenate again the 3 channels to display the processed RGB image

```
RGB = cat(3,R,G,B);
```

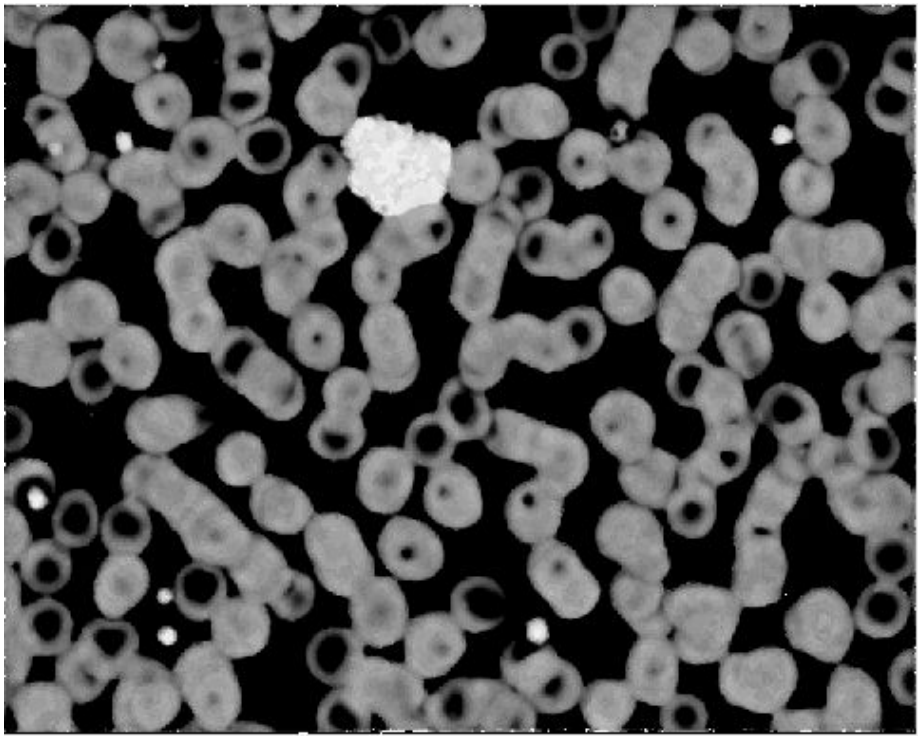


```
HSV = rgb2hsv(im2double(RGB));
sThresh = HSV(:,:,2) > 0.7 & HSV(:,:,2) < 1;
```

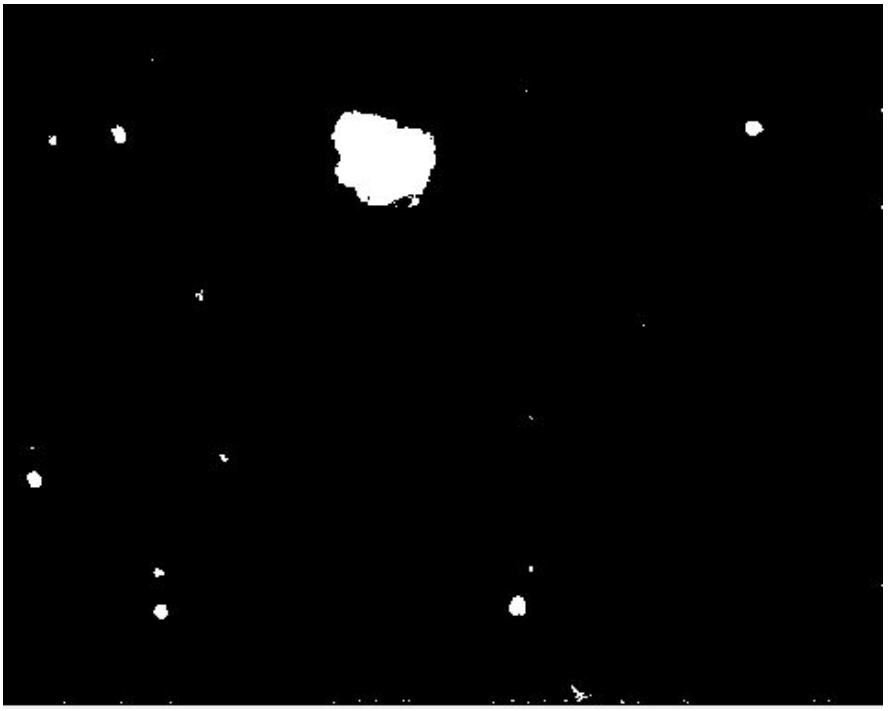
5) Convert the RGB image to HSV image



6) The saturation channel appears to be a good channel for segmenting the purple regions

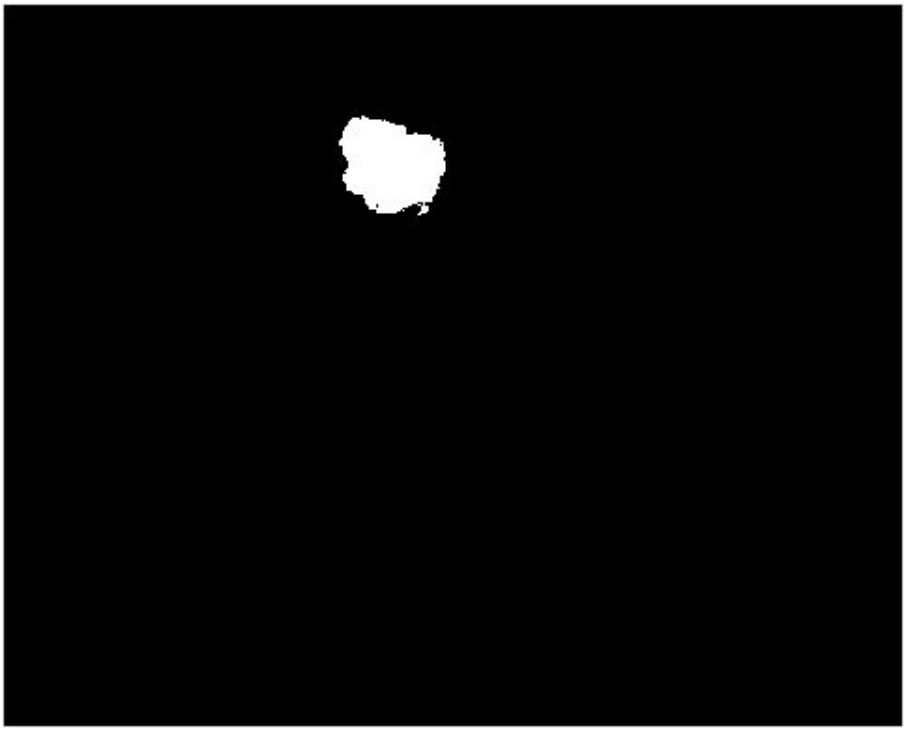


7) Threshold the intensities less than 1 and greater than 0.7



8) As we can see there are a lot of noise that have been segmented with the required region so we will extract the larger regions only

```
finall = bwareafilt(sThresh,[480,100000000]);
```



9) Other images have some holes after the last step so we will apply dilation to fill those holes

```
se = strel('disk',1,0);
final2 = imdilate(final1,se);
```

10) We are going to calculate the similarity between the output images and the ground truth using Jaccard index

```
sim = getJaccard(final2,groundTruth);
totalSimilarity = totalSimilarity + sim;
total = total+1;
```

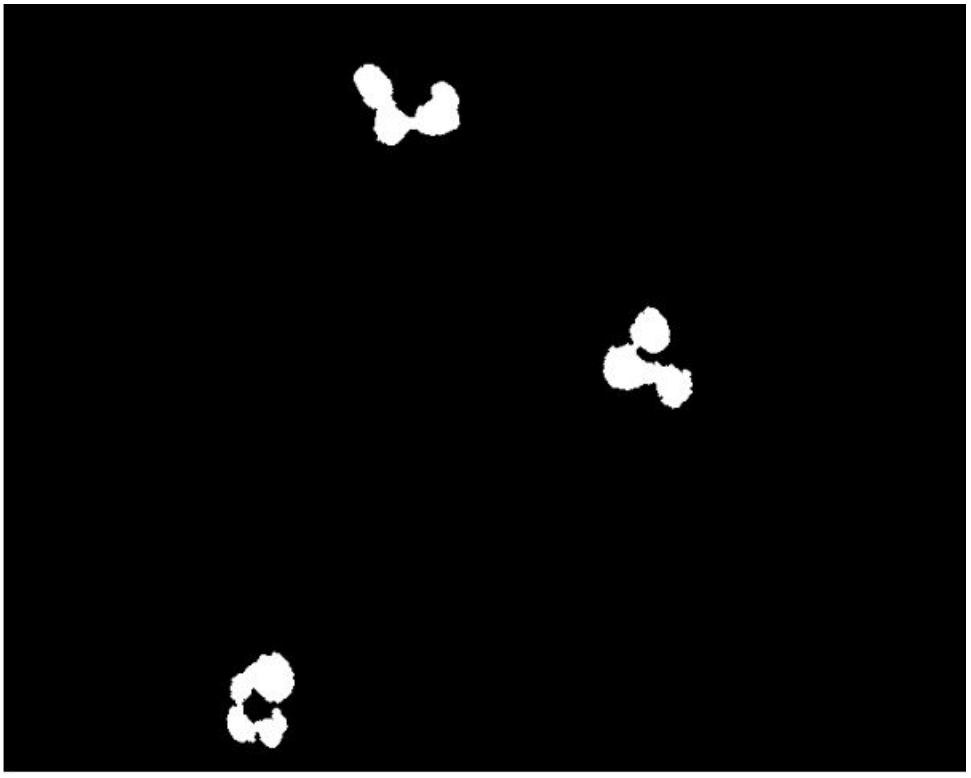
11) Jaccard Function

```
function J = getJaccard(A,B)
J = sum(A(:) & B(:))/sum(A(:) | B(:)) ;
```

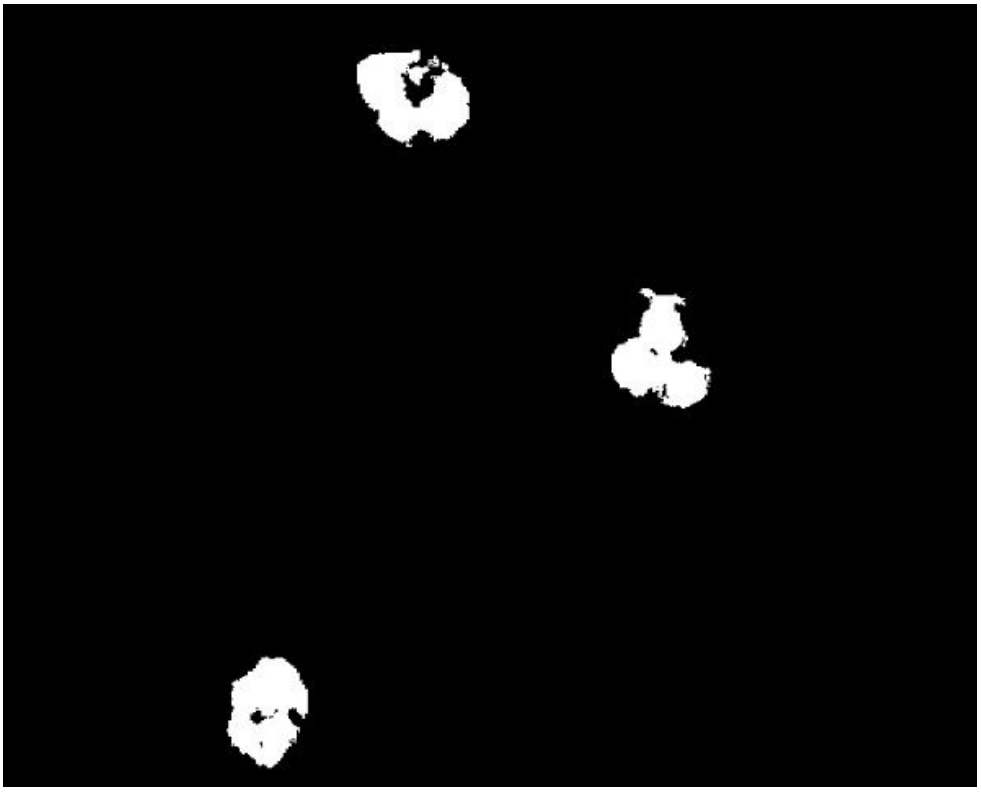
Accuracy = 79.67 %

12) Sample from the dataset

Mine:



Ground Truth:



13) Output images will be saved in folders like the input

```
fileName = sprintf('%d.bmp',i);
folder = strcat('myOutput\',int2str(j));
fullFileName = fullfile(folder,fileName);
imwrite(final2,fullFileName);
```

Part 2: Classification of White Blood Cells

This is a multi-class classification problem, meaning that there are more than two classes to be predicted, in fact there are five types of White Blood Cells. I am going to classify then using VGG19 CNN pretrained model implemented with keras.

1) Batch size = 1 (Every image will update the weights)

```
train_data_dir = '/content/drive/MyDrive/WhiteBloodCellsData'
validation_data_dir = '/content/drive/MyDrive/WhiteBloodCellsData'
batch_size=1
img_width,img_height =500,500
```

2) Data Augmentation (to artificially expand the size of a training dataset by creating modified versions of images in the dataset.) Note: Split the dataset into 70% Training Data and 30% Validation Data

```
datagen = ImageDataGenerator(
    rescale = 1./255,
    shear_range=0.3,
    horizontal_flip=True,
    rotation_range = 2,
    width_shift_range = 0.3,
    height_shift_range = 0.3,
    validation_split = 0.3
)
```

3) The training dataset (171 images)

```
[4] train_generator = datagen.flow_from_directory(
            train_data_dir,
            target_size=(img_width, img_height),
            batch_size=batch_size,
            subset = "training",
            class_mode='categorical',
            )
```

```
Found 171 images belonging to 5 classes.
```

4) The validation dataset (70 images)

```
[7]  validation_generator = datagen.flow_from_directory(
             validation_data_dir,
             target_size=(img_width, img_height),
             batch_size=batch_size,
             subset = "validation",
             class_mode='categorical',
             )
```

```
Found 70 images belonging to 5 classes.
```

5) Import VGG19 pretrained model trained on imagenet dataset

```
[9]  base_model = VGG19(include_top = False, weights = 'imagenet', input_shape = (500,500,3))
```

6) Create a new model and add VGG19 to it

```
[10] model= Sequential()
     model.add(base_model)
     model.add(Flatten())
```

```
[11] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 15, 15, 512)       20024384
_________________________________________________________________
flatten (Flatten)            (None, 115200)            0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

7) Add some fully connected layers after VGG19

```
#--------------
model.add(Dense(1024,activation=('relu'),input_dim=512))
model.add(Dense(512,activation=('relu')))
model.add(Dropout(.4))
model.add(Dense(256,activation=('relu')))
model.add(Dropout(.3))
model.add(Dense(128,activation=('relu')))
model.add(Dense(5,activation=('softmax')))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 15, 15, 512)       20024384
_________________________________________________________________
flatten (Flatten)            (None, 115200)            0
_________________________________________________________________
dense (Dense)                (None, 1024)              117965824
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 5)                 645
=================================================================
Total params: 138,679,877
Trainable params: 138,679,877
Non-trainable params: 0
_________________________________________________________________
```

8) Compiling the model and choosing the optimizer

```
batch_size= batch_size
epochs=500
learn_rate=.00001
sgd=SGD(lr=learn_rate,momentum=.9,nesterov=False)
adam=Adam(lr=learn_rate, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
model.compile(optimizer=sgd,loss='categorical_crossentropy',metrics=['accuracy'])
```

9) Create function to stop the training process when a chosen validation accuracy is met

```
[14] import tensorflow as tf
     class MyThresholdCallback(tf.keras.callbacks.Callback):
         def __init__(self, threshold):
             super(MyThresholdCallback, self).__init__()
             self.threshold = threshold

         def on_epoch_end(self, epoch, logs=None):
             val_acc = logs["val_accuracy"]
             if val_acc >= self.threshold:
                 self.model.stop_training = True
```

10) Start training the model and stop the training process when validation accuracy is equal 80%

```
from keras.callbacks import History
my_callback = MyThresholdCallback(threshold=0.8)
history = History()
model.fit_generator(
        train_generator,
        steps_per_epoch = int(171/batch_size),
        epochs = epochs,
        callbacks = [history, my_callback],
        verbose = 1,
        validation_data = validation_generator,
        validation_steps=int(70/batch_size),
)
```

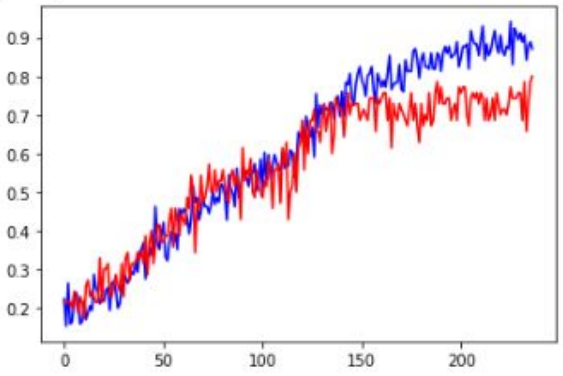
11) The model stopped training when accuracy is met stopped in epoch 237
(Last 4 epochs)

```
171/171 [==============================] - 27s 155ms/step - loss: 0.3597 - accuracy: 0.8343 - val_loss: 1.3613 - val_accuracy: 0.6571
Epoch 235/500
171/171 [==============================] - 27s 157ms/step - loss: 0.2489 - accuracy: 0.8831 - val_loss: 1.1932 - val_accuracy: 0.7286
Epoch 236/500
171/171 [==============================] - 27s 157ms/step - loss: 0.2760 - accuracy: 0.8694 - val_loss: 0.8640 - val_accuracy: 0.7857
Epoch 237/500
171/171 [==============================] - 27s 157ms/step - loss: 0.2209 - accuracy: 0.9103 - val_loss: 0.7098 - val_accuracy: 0.8000
<tensorflow.python.keras.callbacks.History at 0x7fc32cec7470>
```

12) Training Accuracy Vs Validation Accuracy

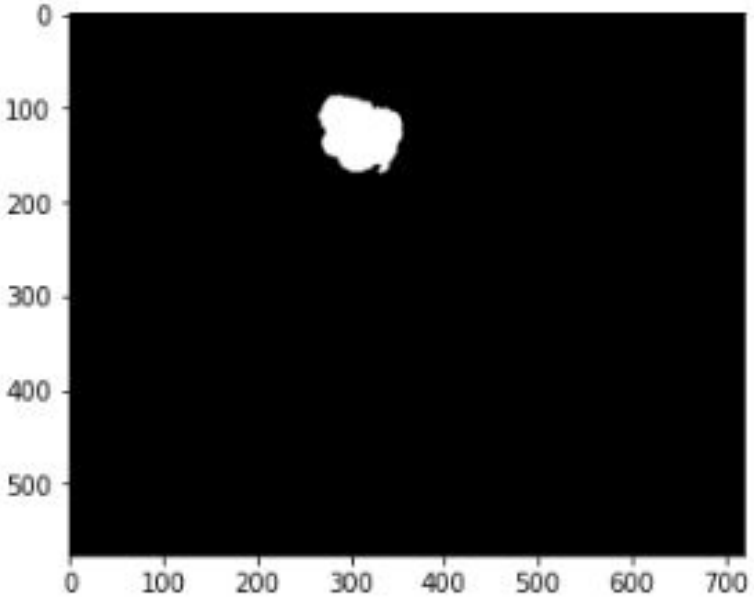
```
#Accuracy
ax[1].plot(model.history.history['accuracy'],color='b',label='Training  Accuracy')
ax[1].plot(model.history.history['val_accuracy'],color='r',label='Validation Accuracy')
```

```
[<matplotlib.lines.Line2D at 0x7fc2ce760828>]
```



13) And finally the model succeeded to recognise the type of this white blood cell (0 = Basophil)

```
/usr/local/lib/python3.6/dist-packages/tensorflow/p
  warnings.warn('`model.predict_classes()` is depre
[0]
```



Accuracy = 80%