In [1]:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPool
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, load_model, Sequential
import numpy as np
import pandas as pd
import shutil
import time
import cv2 as cv2
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import seaborn as sns
sns.set_style('darkgrid')
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
from IPython.core.display import display, HTML
# stop annoying tensorflow warning messages
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
print ('modules loaded')
```

```
modules loaded

C:\Users\mahir\AppData\Local\Temp\ipykernel_10900\1633298320.py:26: Deprecation
Warning: Importing display from IPython.core.display is deprecated since IPython 7.14, please import from IPython display
  from IPython.core.display import display, HTML
```

In [2]:
```
def show_image_samples(gen ):
    t_dict=gen.class_indices
    classes=list(t_dict.keys())
    images,labels=next(gen) # get a sample batch from the generator
    plt.figure(figsize=(20, 20))
    length=len(labels)
    if length<25:   #show maximum of 25 images
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5, 5, i + 1)
        image=images[i]/255
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color='blue', fontsize=12)
        plt.axis('off')
    plt.show()
```

In [3]:
```
def show_images(tdir):
    classlist=os.listdir(tdir)
    length=len(classlist)
    columns=5
    rows=int(np.ceil(length/columns))
    plt.figure(figsize=(20, rows * 4))
    for i, klass in enumerate(classlist):
        classpath=os.path.join(tdir, klass)
        imgpath=os.path.join(classpath, '1.jpg')
        img=plt.imread(imgpath)
        plt.subplot(rows, columns, i+1)
        plt.axis('off')
        plt.title(klass, color='blue', fontsize=12)
        plt.imshow(img)
```

In [4]:
```
def print_in_color(txt_msg,fore_tupple,back_tupple,):
    #prints the text_msg in the foreground color specified by fore_tupple wit
    #text_msg is the text, fore_tupple is foregroud color tupple (r,g,b), bac
    rf,gf,bf=fore_tupple
    rb,gb,bb=back_tupple
    msg='{0}' + txt_msg
    mat='\33[38;2;' + str(rf) +';' + str(gf) + ';' + str(bf) + ';48;2;' + str
    print(msg .format(mat), flush=True)
    print('\33[0m', flush=True) # returns default print color to back to blac
    return
```

In [5]:

```
class LRA(keras.callbacks.Callback):
    def __init__(self,model, base_model, patience,stop_patience, threshold, f
        super(LRA, self).__init__()
        self.model=model
        self.base_model=base_model
        self.patience=patience # specifies how many epochs without improvemen
        self.stop_patience=stop_patience # specifies how many times to adjust
        self.threshold=threshold # specifies training accuracy threshold when
        self.factor=factor # factor by which to reduce the learning rate
        self.dwell=dwell
        self.batches=batches # number of training batch to runn per epoch
        self.initial_epoch=initial_epoch
        self.epochs=epochs
        self.ask_epoch=ask_epoch
        self.ask_epoch_initial=ask_epoch # save this value to restore if rest
        # callback variables
        self.count=0 # how many times lr has been reduced without improvement
        self.stop_count=0
        self.best_epoch=1   # epoch with the lowest loss
        self.initial_lr=float(tf.keras.backend.get_value(model.optimizer.lr))
        self.highest_tracc=0.0 # set highest training accuracy to 0 initially
        self.lowest_vloss=np.inf # set lowest validation loss to infinity ini
        self.best_weights=self.model.get_weights() # set best weights to mode
        self.initial_weights=self.model.get_weights()   # save initial weight

    def on_train_begin(self, logs=None):
        if self.base_model != None:
            status=base_model.trainable
            if status:
                msg=' initializing callback starting training with base_model
            else:
                msg='initializing callback starting training with base_model
        else:
            msg='initialing callback and starting training'
        print_in_color (msg, (244, 252, 3), (55,65,80))
        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s

        print_in_color(msg, (244,252,3), (55,65,80))
        self.start_time= time.time()

    def on_train_end(self, logs=None):
        stop_time=time.time()
        tr_duration= stop_time- self.start_time
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))

        self.model.set_weights(self.best_weights) # set the weights of the mo
        msg=f'Training is completed - model is set with weights from epoch {s
        print_in_color(msg, (0,255,0), (55,65,80))
        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f}
        print_in_color(msg, (0,255,0), (55,65,80))

    def on_train_batch_end(self, batch, logs=None):
        acc=logs.get('accuracy')* 100  # get training accuracy
        loss=logs.get('loss')
```

```
        msg='{0:20s}processing batch {1:4s} of {2:5s} accuracy= {3:8.3f}  los
        print(msg, '\r', end='') # prints over on the same line to show runni

    def on_epoch_begin(self,epoch, logs=None):
        self.now= time.time()

    def on_epoch_end(self, epoch, logs=None):  # method runs on the end of ea
        later=time.time()
        duration=later-self.now
        lr=float(tf.keras.backend.get_value(self.model.optimizer.lr)) # get t
        current_lr=lr
        v_loss=logs.get('val_loss')  # get the validation loss for this epoch
        acc=logs.get('accuracy')  # get training accuracy
        v_acc=logs.get('val_accuracy')
        loss=logs.get('loss')
        if acc < self.threshold: # if training accuracy is below threshold ad
            monitor='accuracy'
            if epoch ==0:
                pimprov=0.0
            else:
                pimprov= (acc-self.highest_tracc )*100/self.highest_tracc
            if acc>self.highest_tracc: # training accuracy improved in the ep
                self.highest_tracc=acc # set new highest training accuracy
                self.best_weights=self.model.get_weights() # traing accuracy
                self.count=0 # set count to 0 since training accuracy improve
                self.stop_count=0 # set stop counter to 0
                if v_loss<self.lowest_vloss:
                    self.lowest_vloss=v_loss
                color= (0,255,0)
                self.best_epoch=epoch + 1  # set the value of best epoch for
            else:
                # training accuracy did not improve check if this has happene
                # if so adjust learning rate
                if self.count>=self.patience -1: # lr should be adjusted
                    color=(245, 170, 66)
                    lr= lr* self.factor # adjust the learning by factor
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) #
                    self.count=0 # reset the count to 0
                    self.stop_count=self.stop_count + 1 # count the number of
                    self.count=0 # reset counter
                    if self.dwell:
                        self.model.set_weights(self.best_weights) # return to
                    else:
                        if v_loss<self.lowest_vloss:
                            self.lowest_vloss=v_loss
                else:
                    self.count=self.count +1 # increment patience counter
        else: # training accuracy is above threshold so adjust learning rate
            monitor='val_loss'
            if epoch ==0:
                pimprov=0.0
            else:
                pimprov= (self.lowest_vloss- v_loss )*100/self.lowest_vloss
            if v_loss< self.lowest_vloss: # check if the validation loss impr
                self.lowest_vloss=v_loss # replace lowest validation loss wit
                self.best_weights=self.model.get_weights() # validation loss
                self.count=0 # reset count since validation loss improved
```

```
                self.stop_count=0
                color=(0,255,0)
                self.best_epoch=epoch + 1 # set the value of the best epoch t
            else: # validation loss did not improve
                if self.count>=self.patience-1: # need to adjust lr
                    color=(245, 170, 66)
                    lr=lr * self.factor # adjust the learning rate
                    self.stop_count=self.stop_count + 1 # increment stop coun
                    self.count=0 # reset counter
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) #
                    if self.dwell:
                        self.model.set_weights(self.best_weights) # return to
                else:
                    self.count =self.count +1 # increment the patience counte
                if acc>self.highest_tracc:
                    self.highest_tracc= acc
        msg=f'{str(epoch+1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc*100:^
        print_in_color (msg,color, (55,65,80))
        if self.stop_count> self.stop_patience - 1: # check if learning rate
            msg=f' training has been halted at epoch {epoch + 1} after {self.
            print_in_color(msg, (0,255,255), (55,65,80))
            self.model.stop_training = True # stop training
        else:
            if self.ask_epoch !=None:
                if epoch + 1 >= self.ask_epoch:
                    if base_model.trainable:
                        msg='enter H to halt training or an integer for numbe
                    else:
                        msg='enter H to halt training ,F to fine tune model,
                    print_in_color(msg, (0,255,255), (55,65,80))
                    ans=input('')
                    if ans=='H' or ans=='h':
                        msg=f'training has been halted at epoch {epoch + 1} d
                        print_in_color(msg, (0,255,255), (55,65,80))
                        self.model.stop_training = True # stop training
                    elif ans == 'F' or ans=='f':
                        if base_model.trainable:
                            msg='base_model is already set as trainable'
                        else:
                            msg='setting base_model as trainable for fine tun
                            self.base_model.trainable=True
                        print_in_color(msg, (0, 255,255), (55,65,80))
                        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9

                        print_in_color(msg, (244,252,3), (55,65,80))
                        self.count=0
                        self.stop_count=0
                        self.ask_epoch = epoch + 1 + self.ask_epoch_initial

                    else:
                        ans=int(ans)
                        self.ask_epoch +=ans
                        msg=f' training will continue until epoch ' + str(sel
                        print_in_color(msg, (0, 255,255), (55,65,80))
                        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9

                        print_in_color(msg, (244,252,3), (55,65,80))
```

In [6]:

```
def tr_plot(tr_data, start_epoch):
    #Plot the training and validation data
    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)#  this is the epoch with the lowest validatio
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(20,8))
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', l
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs')
    axes[0].set_ylabel('Loss')
    axes[0].legend()
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', l
    axes[1].set_title('Training and Validation Accuracy')
    axes[1].set_xlabel('Epochs')
    axes[1].set_ylabel('Accuracy')
    axes[1].legend()
    plt.tight_layout
    #plt.style.use('fivethirtyeight')
    plt.show()
```

In [7]:

```
def print_info( test_gen, preds, print_code, save_dir, subject ):
    class_dict=test_gen.class_indices
    labels= test_gen.labels
    file_names= test_gen.filenames
    error_list=[]
    true_class=[]
    pred_class=[]
    prob_list=[]
    new_dict={}
    error_indices=[]
    y_pred=[]
    for key,value in class_dict.items():
        new_dict[value]=key             # dictionary {integer of class number
    # store new_dict as a text fine in the save_dir
    classes=list(new_dict.values())     # list of string of class names
    errors=0
    for i, p in enumerate(preds):
        pred_index=np.argmax(p)
        true_index=labels[i]  # labels are integer values
        if pred_index != true_index: # a misclassification has occurred
            error_list.append(file_names[i])
            true_class.append(new_dict[true_index])
            pred_class.append(new_dict[pred_index])
            prob_list.append(p[pred_index])
            error_indices.append(true_index)
            errors=errors + 1
        y_pred.append(pred_index)
    tests=len(preds)
    acc= (1-errors/tests) *100
    msg= f'There were {errors} errors in {tests} test cases Model accuracy= {
    print_in_color(msg,(0,255,255),(55,65,80))
    if print_code !=0:
        if errors>0:
            if print_code>errors:
                r=errors
            else:
                r=print_code
            msg='{0:^28s}{1:^28s}{2:^28s}{3:^16s}'.format('Filename', 'Predic
            print_in_color(msg, (0,255,0),(55,65,80))
            for i in range(r):
                split1=os.path.split(error_list[i])
                split2=os.path.split(split1[0])
                fname=split2[1] + '/' + split1[1]
                msg='{0:^28s}{1:^28s}{2:^28s}{3:4s}{4:^6.4f}'.format(fname, p
                print_in_color(msg, (255,255,255), (55,65,60))
                #print(error_list[i]  , pred_class[i], true_class[i], prob_li
        else:
            msg='With accuracy of 100 % there are no errors to print'
            print_in_color(msg, (0,255,0),(55,65,80))
    if errors>0:
        plot_bar=[]
        plot_class=[]
        for  key, value in new_dict.items():
            count=error_indices.count(key)
            if count!=0:
                plot_bar.append(count) # list containg how many times a class
```

```
                plot_class.append(value)   # stores the class
        fig=plt.figure()
        fig.set_figheight(len(plot_class)/3)
        fig.set_figwidth(10)
        plt.style.use('fivethirtyeight')
        for i in range(0, len(plot_class)):
            c=plot_class[i]
            x=plot_bar[i]
            plt.barh(c, x, )
            plt.title( ' Errors by Class on Test Set')
    y_true= np.array(labels)
    y_pred=np.array(y_pred)
    if len(classes)<= 30:
        # create a confusion matrix
        cm = confusion_matrix(y_true, y_pred )
        length=len(classes)
        if length<8:
            fig_width=8
            fig_height=8
        else:
            fig_width= int(length * .5)
            fig_height= int(length * .5)
        plt.figure(figsize=(fig_width, fig_height))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False
        plt.xticks(np.arange(length)+.5, classes, rotation= 90)
        plt.yticks(np.arange(length)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes, digits=
    print("Classification Report:\n----------------------\n", clr)
    return acc/100
```

In [8]:

```
def saver(save_path, model, model_name, subject, accuracy,img_size, scalar, g
    # first save the model
    save_id=str (model_name +  '-' + subject +'-'+ str(acc)[:str(acc).rfind('
    model_save_loc=os.path.join(save_path, save_id)
    model.save(model_save_loc)
    print_in_color ('model was saved as ' + model_save_loc, (0,255,0),(55,65,
    # now create the class_df and convert to csv file
    class_dict=generator.class_indices
    height=[]
    width=[]
    scale=[]
    for i in range(len(class_dict)):
        height.append(img_size[0])
        width.append(img_size[1])
        scale.append(scalar)
    Index_series=pd.Series(list(class_dict.values()), name='class_index')
    Class_series=pd.Series(list(class_dict.keys()), name='class')
    Height_series=pd.Series(height, name='height')
    Width_series=pd.Series(width, name='width')
    Scale_series=pd.Series(scale, name='scale by')
    class_df=pd.concat([Index_series, Class_series, Height_series, Width_seri
    csv_name='class_dict.csv'
    csv_save_loc=os.path.join(save_path, csv_name)
    class_df.to_csv(csv_save_loc, index=False)
    print_in_color ('class csv file was saved as ' + csv_save_loc, (0,255,0),
    return model_save_loc, csv_save_loc
```

In [9]:

```
def predictor(sdir, csv_path,  model_path, averaged=True, verbose=True):
    # read in the csv file
    class_df=pd.read_csv(csv_path)
    class_count=len(class_df['class'].unique())
    img_height=int(class_df['height'].iloc[0])
    img_width =int(class_df['width'].iloc[0])
    img_size=(img_width, img_height)
    scale=class_df['scale by'].iloc[0]
    # determine value to scale image pixels by
    try:
        s=int(scale)
        s2=1
        s1=0
    except:
        split=scale.split('-')
        s1=float(split[1])
        s2=float(split[0].split('*')[1])
    path_list=[]
    paths=os.listdir(sdir)
    for f in paths:
        path_list.append(os.path.join(sdir,f))
    if verbose:
        print (' Model is being loaded- this will take about 10 seconds')
    model=load_model(model_path)
    image_count=len(path_list)
    image_list=[]
    file_list=[]
    good_image_count=0
    for i in range (image_count):
        try:
            img=cv2.imread(path_list[i])
            img=cv2.resize(img, img_size)
            img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            good_image_count +=1
            img=img*s2 - s1
            image_list.append(img)
            file_name=os.path.split(path_list[i])[1]
            file_list.append(file_name)
        except:
            if verbose:
                print ( path_list[i], ' is an invalid image file')
    if good_image_count==1: # if only a single image need to expand dimension
        averaged=True
    image_array=np.array(image_list)
    # make predictions on images, sum the probabilities of each class then fi
    # highest probability
    preds=model.predict(image_array)
    if averaged:
        psum=[]
        for i in range (class_count): # create all 0 values list
            psum.append(0)
        for p in preds: # iterate over all predictions
            for i in range (class_count):
                psum[i]=psum[i] + p[i]  # sum the probabilities
        index=np.argmax(psum) # find the class index with the highest probabi
        klass=class_df['class'].iloc[index] # get the class name that corresp
```

```
        prob=psum[index]/good_image_count * 100  # get the probability averag
        # to show the correct image run predict again and select first image
        for img in image_array:  #iterate through the images
            test_img=np.expand_dims(img, axis=0) # since it is a single image
            test_index=np.argmax(model.predict(test_img)) # for this image fi
            if test_index== index: # see if this image has the same index as
                if verbose: # show image and print result if verbose=1
                    plt.axis('off')
                    plt.imshow(img) # show the image
                    print (f'predicted species is {klass} with a probability
                break # found an image that represents the predicted class
        return klass, prob, img, None
    else: # create individual predictions for each image
        pred_class=[]
        prob_list=[]
        for i, p in enumerate(preds):
            index=np.argmax(p) # find the class index with the highest probab
            klass=class_df['class'].iloc[index] # get the class name that cor
            image_file= file_list[i]
            pred_class.append(klass)
            prob_list.append(p[index])
        Fseries=pd.Series(file_list, name='image file')
        Lseries=pd.Series(pred_class, name= 'species')
        Pseries=pd.Series(prob_list, name='probability')
        df=pd.concat([Fseries, Lseries, Pseries], axis=1)
        if verbose:
            length= len(df)
            print (df.head(length))
        return None, None, None, df
```

In [10]:

```
def trim (df, max_size, min_size, column):
    df=df.copy()
    original_class_count= len(list(df[column].unique()))
    print ('Original Number of classes in dataframe: ', original_class_count)
    sample_list=[]
    groups=df.groupby(column)
    for label in df[column].unique():
        group=groups.get_group(label)
        sample_count=len(group)
        if sample_count> max_size :
            strat=group[column]
            samples,_=train_test_split(group, train_size=max_size, shuffle=Tr
            sample_list.append(samples)
        elif sample_count>= min_size:
            sample_list.append(group)
    df=pd.concat(sample_list, axis=0).reset_index(drop=True)
    final_class_count= len(list(df[column].unique()))
    if final_class_count != original_class_count:
        print ('*** WARNING***  dataframe has a reduced number of classes' )
    balance=list(df[column].value_counts())
    print (balance)
    return df
```

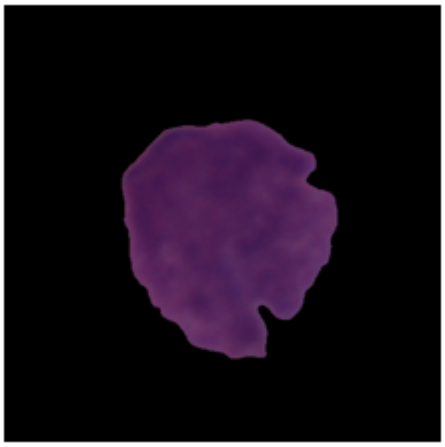
In [11]:

```
def balance(train_df,max_samples, min_samples, column, working_dir, image_siz
    train_df=train_df.copy()
    train_df=trim (train_df, max_samples, min_samples, column)
    # make directories to store augmented images
    aug_dir=os.path.join(working_dir, 'aug')
    if os.path.isdir(aug_dir):
        shutil.rmtree(aug_dir)
    os.mkdir(aug_dir)
    for label in train_df['labels'].unique():
        dir_path=os.path.join(aug_dir,label)
        os.mkdir(dir_path)
    # create and store the augmented images
    total=0
    gen=ImageDataGenerator(horizontal_flip=True,  rotation_range=20, width_sh
                           height_shift_range=.2, zoom_range=.2)
    groups=train_df.groupby('labels') # group by class
    for label in train_df['labels'].unique():  # for every class
        group=groups.get_group(label)  # a dataframe holding only rows with t
        sample_count=len(group)   # determine how many samples there are in t
        if sample_count< max_samples: # if the class has less than target num
            aug_img_count=0
            delta=max_samples-sample_count  # number of augmented images to c
            target_dir=os.path.join(aug_dir, label)  # define where to write
            aug_gen=gen.flow_from_dataframe( group,  x_col='filepaths', y_col
                                            class_mode=None, batch_size=1, sh
                                            save_to_dir=target_dir, save_pref
                                            save_format='jpg')
            while aug_img_count<delta:
                images=next(aug_gen)
                aug_img_count += len(images)
            total +=aug_img_count
    print('Total Augmented images created= ', total)
    # create aug_df and merge with train_df to create composite training set
    if total>0:
        aug_fpaths=[]
        aug_labels=[]
        classlist=os.listdir(aug_dir)
        for klass in classlist:
            classpath=os.path.join(aug_dir, klass)
            flist=os.listdir(classpath)
            for f in flist:
                fpath=os.path.join(classpath,f)
                aug_fpaths.append(fpath)
                aug_labels.append(klass)
        Fseries=pd.Series(aug_fpaths, name='filepaths')
        Lseries=pd.Series(aug_labels, name='labels')
        aug_df=pd.concat([Fseries, Lseries], axis=1)
        train_df=pd.concat([train_df,aug_df], axis=0).reset_index(drop=True)

    print (list(train_df['labels'].value_counts()) )
    return train_df
```

In [13]:
```
img_path=r'C:/Users/mahir/Documents/All the dreams in the world/Thesis Projec
img=plt.imread(img_path)
print ('Input image shape is ',img.shape)
plt.axis('off')
imshow(img)
```

```
Input image shape is  (450, 450, 3)
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x21e21ec9820>
```



In [14]:
```
def preprocess (sdir, trsplit, vsplit):
    filepaths=[]
    labels=[]
    folds=os.listdir(sdir)
    for fold in folds:
        foldpath=os.path.join(sdir,fold)
        classlist=os.listdir(foldpath)
        for klass in classlist:
            classpath=os.path.join(foldpath,klass)
            flist=os.listdir(classpath)
            for f in flist:
                fpath=os.path.join(classpath,f)
                filepaths.append(fpath)
                labels.append(klass)
    Fseries=pd.Series(filepaths, name='filepaths')
    Lseries=pd.Series(labels, name='labels')
    df=pd.concat([Fseries, Lseries], axis=1)
    dsplit=vsplit/(1-trsplit)
    strat=df['labels']
    train_df, dummy_df=train_test_split(df, train_size=trsplit, shuffle=True,
    strat=dummy_df['labels']
    valid_df, test_df= train_test_split(dummy_df, train_size=dsplit, shuffle=
    print('train_df length: ', len(train_df), '  test_df length: ',len(test_d
     # check that each dataframe has the same number of classes to prevent mo
    trcount=len(train_df['labels'].unique())
    tecount=len(test_df['labels'].unique())
    vcount=len(valid_df['labels'].unique())
    if trcount < tecount :
        msg='** WARNING ** number of classes in training set is less than the
        print_in_color(msg, (255,0,0), (55,65,80))
        msg='This will throw an error in either model.evaluate or model.predi
        print_in_color(msg, (255,0,0), (55,65,80))
    if trcount != vcount:
        msg='** WARNING ** number of classes in training set not equal to num
        print_in_color(msg, (255,0,0), (55,65,80))
        msg=' this will throw an error in model.fit'
        print_in_color(msg, (255,0,0), (55,65,80))
        print ('train df class count: ', trcount, 'test df class count: ', te
        ans=input('Enter C to continue execution or H to halt execution')
        if ans =='H' or ans == 'h':
            print_in_color('Halting Execution', (255,0,0), (55,65,80))
            import sys
            sys.exit('program halted by user')
    print(list(train_df['labels'].value_counts()))
    return train_df, test_df, valid_df
```

In [17]:
```
sdir=r'C:/Users/mahir/Documents/All the dreams in the world/Thesis Project/te
trsplit=.9
vsplit=.05
train_df, test_df, valid_df= preprocess(sdir,trsplit, vsplit)
```

```
train_df length:  9594   test_df length:  534   valid_df length:  533
[6544, 3050]
```

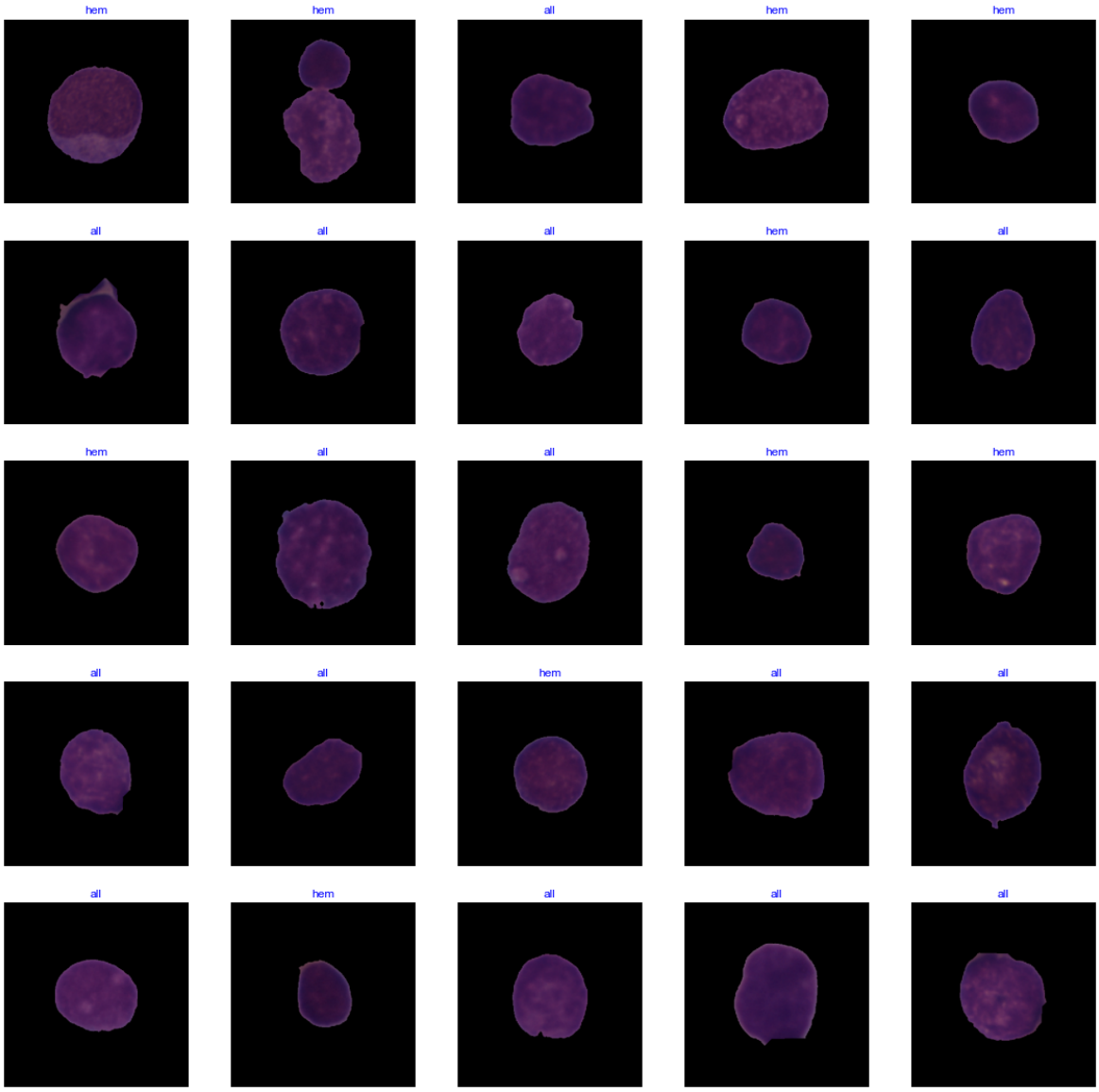
In [18]:
```
max_samples= 3050
min_samples=0
column='labels'
working_dir = r'./'
img_size=(300,300)
train_df=trim(train_df, max_samples, min_samples, column)
```

```
Original Number of classes in dataframe:  2
[3050, 3050]
```

In [19]:
```
channels=3
batch_size=40
img_shape=(img_size[0], img_size[1], channels)
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length %
test_steps=int(length/test_batch_size)
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps)
def scalar(img):
    return img  # EfficientNet expects pixelsin range 0 to 255 so no scaling
trgen=ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=True)
tvgen=ImageDataGenerator(preprocessing_function=scalar)
msg='                                                                    for the tr
print(msg, '\r', end='')
train_gen=trgen.flow_from_dataframe( train_df, x_col='filepaths', y_col='labe
                                    color_mode='rgb', shuffle=True, batch_siz
msg='                                                                    for the te
print(msg, '\r', end='')
test_gen=tvgen.flow_from_dataframe( test_df, x_col='filepaths', y_col='labels
                                    color_mode='rgb', shuffle=False, batch_si
msg='                                                                   for the val
print(msg, '\r', end='')
valid_gen=tvgen.flow_from_dataframe( valid_df, x_col='filepaths', y_col='labe
                                    color_mode='rgb', shuffle=True, batch_siz
classes=list(train_gen.class_indices.keys())
class_count=len(classes)
train_steps=int(np.ceil(len(train_gen.labels)/batch_size))
labels=test_gen.labels
```

```
test batch size:  6   test steps:  89
Found 6100 validated image filenames belonging to 2 classes.  for the train
generator
Found 534 validated image filenames belonging to 2 classes.   for the test
generator
Found 533 validated image filenames belonging to 2 classes.  for the valida
tion generator
```

In [20]:
```
show_image_samples(train_gen)
```



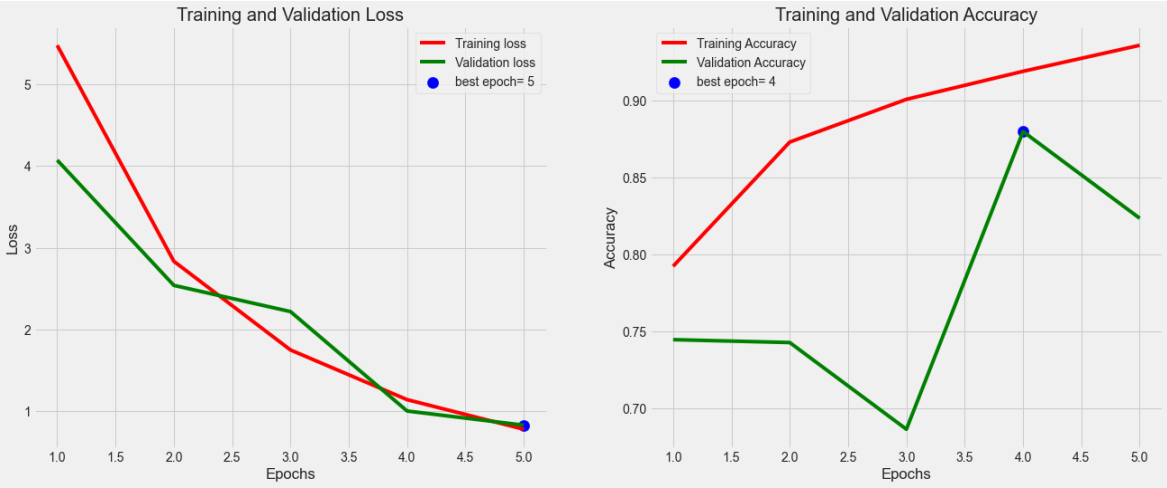
In [21]:
```
model_name='EfficientNetB3'
base_model=tf.keras.applications.efficientnet.EfficientNetB3(include_top=Fals
x=base_model.output
x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regul
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x
x=Dropout(rate=.45, seed=123)(x)
output=Dense(class_count, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
model.compile(Adamax(learning_rate=.001), loss='categorical_crossentropy', me
```

In [22]:
```
epochs =40
patience= 1 # number of epochs to wait to adjust lr if monitored value does n
stop_patience =3 # number of epochs to wait before stopping training if monit
threshold=.9 # if train accuracy is < threshhold adjust monitor accuracy, els
factor=.5 # factor to reduce lr by
dwell=True # experimental, if True and monitored metric does not improve on c
freeze=False # if true free weights of  the base model
ask_epoch=5 # number of epochs to run before asking if you want to halt train
batches=train_steps
callbacks=[LRA(model=model,base_model= base_model,patience=patience,stop_pati
                    factor=factor,dwell=dwell, batches=batches,initial_epoch=0
```

In [23]:
```
history=model.fit(x=train_gen,  epochs=epochs, verbose=0, callbacks=callbacks
                  validation_steps=None,  shuffle=False,  initial_epoch=0)
```

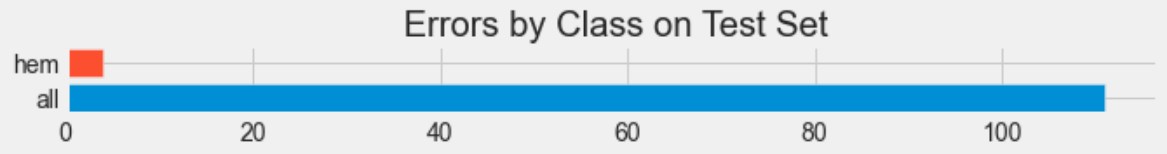
```
 initializing callback starting training with base_model trainable

 Epoch     Loss   Accuracy  V_loss    V_acc     LR      Next LR  Monitor  %
 Improv  Duration

 1 /40     5.482   79.246   4.07478  74.484   0.00100  0.00100  accuracy
0.00   15962.51

 2 /40     2.834   87.311   2.53896  74.296   0.00100  0.00100  accuracy
10.18   63879.28

 3 /40     1.747   90.082   2.21746  68.668   0.00100  0.00100  val_loss
12.66   16235.99

 4 /40     1.138   91.902   0.99870  87.992   0.00100  0.00100  val_loss
54.96   21648.26

 5 /40     0.775   93.590   0.82506  82.364   0.00100  0.00100  val_loss
17.39   23450.06

enter H to halt training or an integer for number of epochs to run then ask
again

H
training has been halted at epoch 5 due to user input

Training is completed - model is set with weights from epoch 5

training elapsed time was 41.0 hours, 53.0 minutes, 49.35 seconds)
```

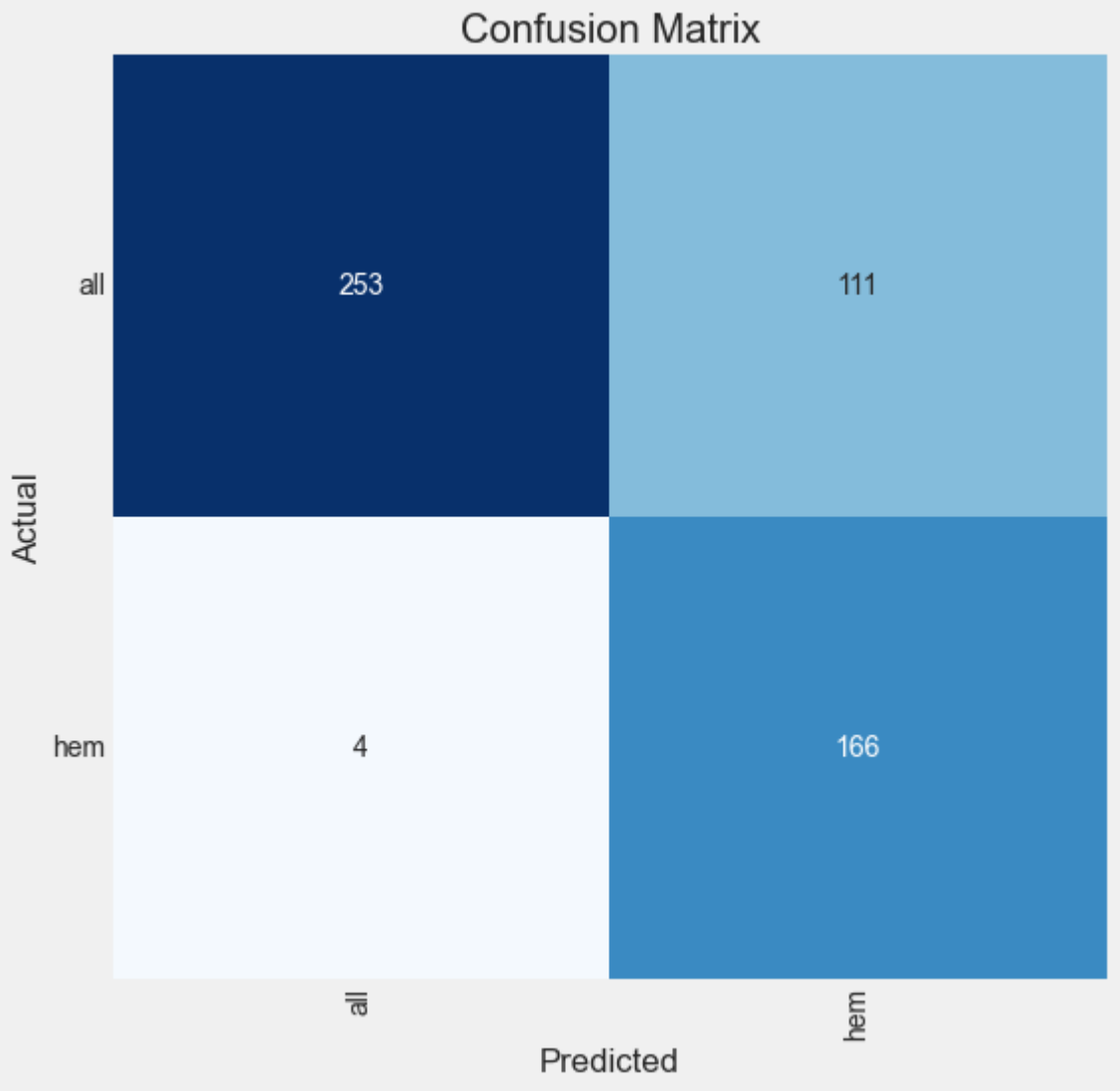
In [24]:
```
tr_plot(history,0)
```



In [25]:
```
subject='leukemia'
print_code=0
preds=model.predict(test_gen)
acc=print_info( test_gen, preds, print_code, working_dir, subject )
```

```
There were 115 errors in 534 test cases Model accuracy=  78.46 %
```






```
Classification Report:
----------------------
              precision    recall  f1-score   support

         all     0.9844    0.6951    0.8148       364
```

```
                hem       0.5993    0.9765    0.7427       170

           accuracy                           0.7846       534
          macro avg       0.7919    0.8358    0.7788       534
       weighted avg       0.8618    0.7846    0.7919       534
```

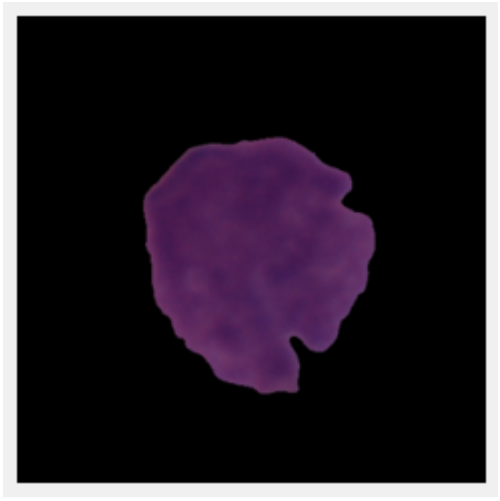
In [26]:
```
model_save_loc, csv_save_loc=saver(working_dir, model, model_name, subject, a
```

```
model was saved as ./EfficientNetB3-leukemia-0.78.h5

class csv file was saved as ./class_dict.csv
```

In [27]:

```
img=plt.imread(img_path)
print ('Input image shape is ', img.shape)
# resize the image so it is the same size as the images the model was trainea
img=cv2.resize(img, img_size) # in earlier code img_size=(224,224) was used f
print ('the resized image has shape ', img.shape)
### show the resized image
plt.axis('off')
plt.imshow(img)
# Normally the next line of code rescales the images. However the EfficientNe
# img= img/255
# plt.imread returns a numpy array so it is not necessary to convert the imag
# since we have only one image we have to expand the dimensions of img so it
# where the first dimension 1 is the batch size used by model.predict
img=np.expand_dims(img, axis=0)
print ('image shape after expanding dimensions is ',img.shape)
# now predict the image
pred=model.predict(img)
print ('the shape of prediction is ', pred.shape)
# this dataset has 15 classes so model.predict will return a list of 15 proba
# we want to find the index of the column that has the highest probability
index=np.argmax(pred[0])
# to get the actual Name of the class earlier Imade a list of the class names
klass=classes[index]
# lets get the value of the highest probability
probability=pred[0][index]*100
# print out the class, and the probability
print(f'the image is predicted as being {klass} with a probability of {probab
```

```
Input image shape is  (450, 450, 3)
the resized image has shape  (300, 300, 3)
image shape after expanding dimensions is  (1, 300, 300, 3)
the shape of prediction is  (1, 2)
the image is predicted as being all with a probability of  90.25 %
```



In [28]:

```
testdir=r'C:/Users/mahir/Documents/All the dreams in the world/Thesis Project
filepaths=[]
klass=[]
flist=os.listdir(testdir)
print ('number of test files is ', len(flist))
for f in flist:
    fpath=os.path.join(testdir,f)
    filepaths.append(fpath)
    klass=' '
Fseries=pd.Series(filepaths, name='filepaths')
Lseries=pd.Series(klass, name='Class')
test_df=pd.concat([Fseries, Lseries], axis=1)
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length %
test_steps=int(length/test_batch_size)
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps)
test_gen=tvgen.flow_from_dataframe( test_df, x_col='filepaths', y_col=None, t
                                   color_mode='rgb', shuffle=False, batch_si
preds=model.predict(test_gen, verbose=1, steps=test_steps)
for i, p in enumerate(preds):
    index=np.argmax(p)
    klass=classes[index]
    test_df['Class'].iloc[i]=klass
print (test_df.head())
csv_name='submission'
csv_path=os.path.join(working_dir, csv_name)
test_df.to_csv(csv_path, index=False)
# read in csv file to see if it is correct
df_test=pd.read_csv(csv_path)
print (df_test.head())
```

```
number of test files is  2586
test batch size:  6   test steps:  431
Found 2586 validated image filenames.
431/431 [==============================] - 760s 2s/step
                                    filepaths Class
0  C:/Users/mahir/Documents/All the dreams in the...   hem
1  C:/Users/mahir/Documents/All the dreams in the...   hem
2  C:/Users/mahir/Documents/All the dreams in the...   hem
3  C:/Users/mahir/Documents/All the dreams in the...   hem
4  C:/Users/mahir/Documents/All the dreams in the...   hem
                                    filepaths Class
0  C:/Users/mahir/Documents/All the dreams in the...   hem
1  C:/Users/mahir/Documents/All the dreams in the...   hem
2  C:/Users/mahir/Documents/All the dreams in the...   hem
3  C:/Users/mahir/Documents/All the dreams in the...   hem
4  C:/Users/mahir/Documents/All the dreams in the...   hem
```

In [ ]: