Capstone Project: Deep Learning

Identification of Plasmodium spp. from Thin Blood Smears using Deep Learning.

Malaria diagnosis is performed as a thin blood smear by a Parasitologist with specialized training. There currently exists a need to have an alternative method to read blood smear slides for the detection of Malaria. The goal is to build a computer vision model that is successful at identifying Malaria infections at the species level collection of infected and uninfected blood smears would need to be analyzed for key features.

Background

Malaria is a vector-borne human pathogen. Infection occurs when the *Plasmodium* parasite is transmitted to the bloodstream of a human host by way of the common mosquito, *Anopheles japonica*. There are four species of the Plasmodium parasite causing human disease. *Plasmodium falciparum, P. vivax, P. ovale, and Plasmodium malariae*. The parasite matures in the human liver. Key features of the parasite that would aid visualization of the parasite include chromatin dots, dark pigments, single mass dark pigmented, enlarged cytoplasm, chromatin dots, yellow-brown pigments, large chromatin, and dark-brown pigment.

According to the World Health Organization, there were an estimated 241 million infections and 627,000 deaths from Malaria. Of the world regions laboring under the greatest number of infections and death are the African Regions. The WHO African Region is the source of 95% of infections and 96% of deaths. Testing is performed as a thin blood smear stained with Giema-Wright stain to visualize all phases of the parasite life cycle.

Problem Definition

Context

The number of certified Parasitology Microbiologists with the skill and expertise to read blood smear slides for Malaria is rapidly diminishing and the technical expertise is evaporating. To best solve this subject matter gap it is proposed to *create a predictive algorithm to detect a Plasmodium spp. at any stage of the parasite life cycle among normal red blood cells?* Can the model be further trained to speciate the detected Plasmodium spp. based on key features in the parasite morphology?

Objectives

Diagnosis of Malaria is performed as a thin blood smear by a trained and certified medical laboratory of health scientist. Due to the specialized training required there are few certified professionals. There currently exists a need to have an alternative method to read blood smear slides for detection of Malaria. The goal is to build a computer vision model that is successful at identifying Malaria infections at the species level collection of infected and uninfected blood smears would need to be analyzed for key features. The parasite has five stages to the the life cycle. To successfully identify to the genus level common features that remain consistent across the parasite liefestyle.

Building a Convolutional Neural Network is the next step, the model should flatten and pool the dense layers trained over several epochs and model iterations to yield a high-performing model with a high degree of accuracy. False negative samples could have serious health outcomes including mortality thus precision and accuracy must match or exceed the accuracy rate of traditional light microscopy.

Key Questions

Can the system be trained to report the presence or absence of the Plasmodium parasite in blood smear images? What is the optimized model to detect the parasite that yields a high confidence result with accuracy and reproducibility?

Plasmodium parasites damage red blood cells leaving behind cellular debris. Can the code be trained to identify ruptured red blood cells and other cellular debris as noise to better detect parasites?

Plasmodium spp. that are of human interest are malariae, vivax, ovale, and falciparum; each with specific characteristics at each stage of growth. The options for key features can remain static as a matrix of growth stages crossed by cell morphology key features. Can the above be a matrix to classify by species?

Can the system be trained to report key information required for diagnosis? Future features should include ELR using LOINC and SNOMED codes and be linkable to the LMS. Refer to DPDx for reporting criteria.

Problem formulation

- Images are thin blood smears stained with Giemsa-Wright stain. This is a differential stain in which the parasite stains a different color than the red blood cells and extracellular debris.
- Use of uninfected smears will allow training to healthy normal red blood cells (RBC). This training should permit early detection of RBC distortions capturing parasites at the Gametophyte stage. This is particular to aid in the detection of *Plasmodium falciparum*.

- Classification of enlarged RBC at the trophozoite stage is distinctive for the detection of vivax species.
- Detection of Schüffner's dots at the ring stage of the parasite life cycle is another key feature as is the visualization of Ziemann's stippling when present from the trophozoite stage.
- The edges of parasite structures can be defined using the known blue to purple color for the staining of the parasite. Flattening the color range to highlight the parasite from background cell debris and healthy cells.

Capstone Project: Deep Learning, Milestone 1

Exploratory Data Analysis

Methods

Data Exploration

This data set contains a series of thin blood smear images stained with the Giemsa-Wright differential stain for the detection of Malaria. The dataset is divided into training and test images, further organized, and labeled as 'Uninfected' or 'Parasitized'. The data source is a trusted source.

Mount the Drive

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

Library Import

Dataset

To evaluate the data parameters the data stored in the zip file must be extracted and read data into a DataFrame. Then it is possible to evaluate the data for the dimensions and size. Additionally, the data should be examined for missing values, normalizing the data, and consistency in language.

In [ ]:
```
#Data Handling library
import numpy as np
#from numpy.core.fromnumeric import size
import pandas as pd

#Graphing library
import matplotlib.pyplot as plt
%matplotlib inline

#Data Visualization library
import seaborn as sns
sns.set_color_codes(palette='dark')
sns.dark_palette=('seagreen')

#Machine Learning library
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

#Suppress Python Library depreciation warnings
import warnings
warnings.filterwarnings('ignore')

#Set maximum size of pandas DataFrame
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)
```

In [ ]:
```
#The dataset is named 'cell_images.zip' and is located in the Colab Notebook
#The data is provided as a zip file that requires unzipping.  Load zipfile t
import zipfile
import os

#File location to be read
path='/content/drive/MyDrive/Colab Notebooks/Capstone Project/cell_images.zi

#Extract data files from zip file
with zipfile.ZipFile(path, 'r') as zip_ref:
    zip_ref.extractall()
```

Exploratory Data Analysis

Processing Image Data

The images should be evaluated for image format consistency, image size and dimension, and pixel depth. This can be accomplished using the Python library Pillow, an image processing library specifically with the capability to perform color space conversions, image resizing, and contrast enhancement.

In [ ]:
```
#Import Pillow Library as PIL
from PIL import Image
```

In [ ]:

```
#Assigning train data for infected and uninfected
train_dir='/content/cell_images/train/'

#Designate image size for normality across data set.
SIZE=64

#Create list containers for the image data
train_images=[]

#Create a list container for Boolean value indicating infection state (0:uni
train_label=[]

#Loading Training data into List
for folder_name in ['/parasitized/','/uninfected/']:
  images_path=os.listdir(train_dir + folder_name)
  for i, image_name in enumerate(images_path):
    try:
      #Open each image in sequence as exception coding
      image=Image.open(train_dir + folder_name + image_name)
      #Resize Images to 64x64 to standardize to same shape
      image=image.resize((SIZE,SIZE))
      #Convert the images to list 'train_images' as designated above
      train_images.append(np.array(image))
      #Create the infected and uninfected labels with conditional logic
      if folder_name=='/parasitized/':
        train_label.append(1)
      else:
        train_label.append(0)
    except Exception:
      pass

#Convert list above into NumPy Array
train_images=np.array(train_images)
train_label=np.array(train_label)
```

In [ ]:

```
#Assigning test data for infected and uninfected
test_dir='/content/cell_images/test/'

#Designate image size for normality across data set.
SIZE=64

#Create list containers for the image data
test_images=[]

#Create a list container for Boolean value indicating infection state (0:uni
test_label=[]

#Loading Test data into List
for folder_name in ['/parasitized/','/uninfected/']:
  images_path=os.listdir(test_dir + folder_name)
  for i, image_name in enumerate(images_path):
    try:
      #Open each image in sequence as exception coding
      image=Image.open(test_dir + folder_name + image_name)
      #Resize Images to 64x64 to standardize to same shape
      image=image.resize((SIZE,SIZE))
      #Convert the images to list 'train_images' as designated above
      test_images.append(np.array(image))
      #Create the infected and uninfected labels with conditional logic
      if folder_name=='/parasitized/':
        test_label.append(1)
      else:
        test_label.append(0)
    except Exception:
      pass

#Convert list above into NumPy Array
test_images=np.array(test_images)
test_label=np.array(test_label)
```

In [ ]:
```
#Determine the number of elements in each data set
print("There are a total of",len(train_images),"images in the Training Image
print("There are a total of",len(train_label),"labels in the Training Labels
print("There are a total of",len(test_images),"images in the Test Images dat
print("There are a total of",len(test_label),"labels in the Test Labels data
print(". . . . . .  . . . . . .")
#Check the shape of the dataset
print("The training dataset has the following shape: ", train_images.shape)
print("The test dataset has the following shape: ", test_images.shape)
print(". . . . . .  . . . . . .")
#Use NumPy max, min to determine the range of data in test and train data se
print("The maximum value of the training dataset is",np.max(train_images))
print("The minimum value of the training dataset is",np.min(train_images))
print("The maximum value of the test dataset is",np.max(test_images))
print("The minimum value of the test dataset is",np.min(test_images))
print(". . . . . .  . . . . . .")
#Pass NumPy Array to Pandas DataFrame
df_train=pd.DataFrame(train_label, columns=['Infection_Status'])
#Use the value_counts to count the total number of infected and uninfected c
print("The number of training images are:\n",
      df_train['Infection_Status'].value_counts())
```

```
There are a total of 24958 images in the Training Images dataset.
There are a total of 24958 labels in the Training Labels dataset.
There are a total of 2600 images in the Test Images dataset.
There are a total of 2600 labels in the Test Labels dataset.
. . . . . .  . . . . . .
The training dataset has the following shape:  (24958, 64, 64, 3)
The test dataset has the following shape:  (2600, 64, 64, 3)
. . . . . .  . . . . . .
The maximum value of the training dataset is 255
The minimum value of the training dataset is 0
The maximum value of the test dataset is 255
The minimum value of the test dataset is 0
. . . . . .  . . . . . .
The number of training images are:
 1    12582
0    12376
Name: Infection_Status, dtype: int64
```

Observations about the data:

- There are a total of 24,958 images in both the Training Images and Training Labels data sets. For each image, there is an attached label to designate infection status of the blood smear. The training data set is resized images to 64x64x3.
- There are a total of 2600 images in the test data set, each with a corresponding label for infection status.
- Of the images in the training data set, there are 12,376 uninfected smears and 12,582 infected blood smears. There is a representative number of samples for both infected and uninfected specimen images.
- The minimum and maximum values span the pixel range of 0 to 255, the full dark to the light spectrum.

Data Preparation

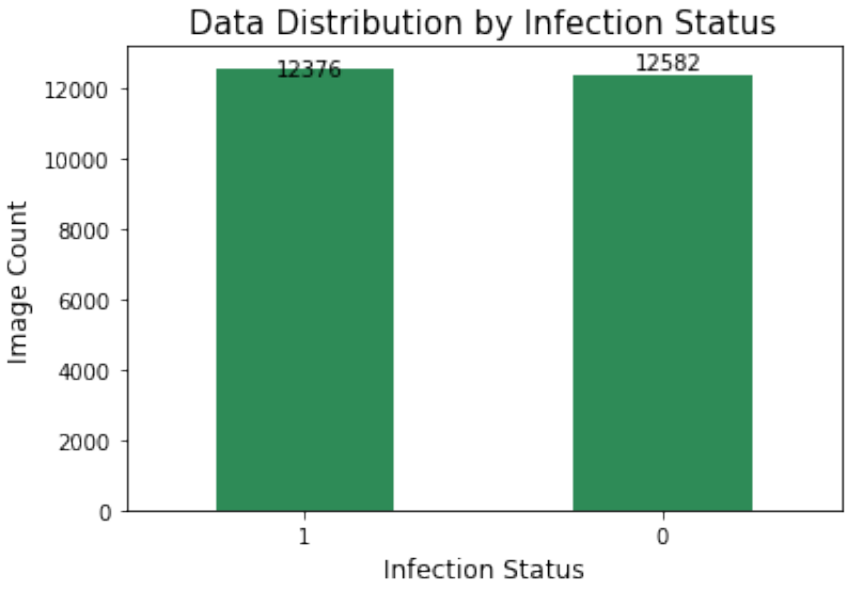
To prepare the images for machine learning the data shape and array must be determined. The data must then be normalized to ensure each pixel has a similar distribution.

In [ ]:
```
#Normalize the data.  There are 255 pixels thus all data will be normalized
norm_train_images = (train_images/255).astype('float32')
norm_test_images = (test_images/255).astype('float32')
```

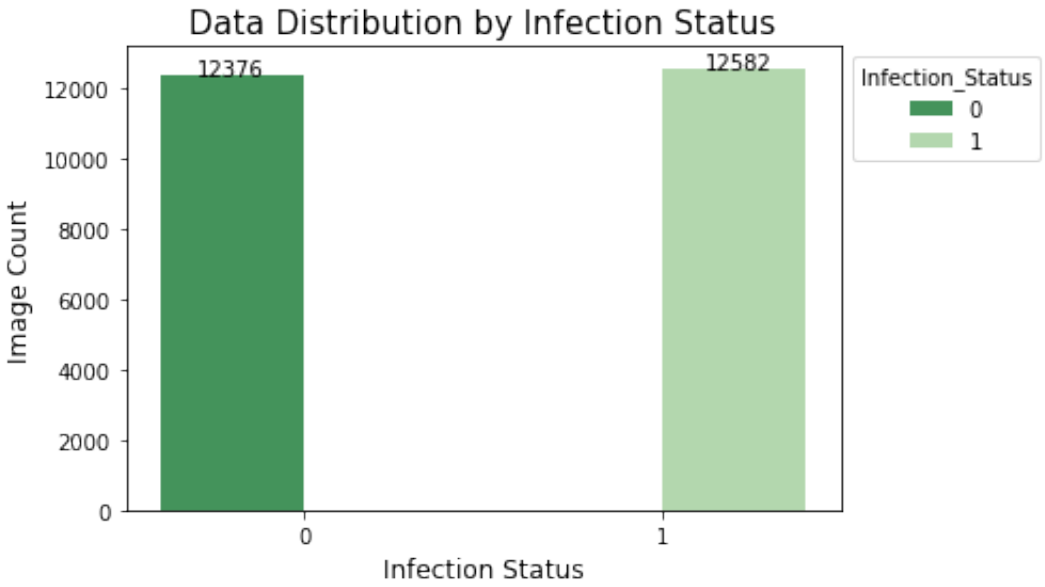
In [ ]:
```
#Determine new data shapes following normalization.
#Check the shape of the dataset
print("The training dataset has the following shape: ", norm_train_images.sh
print("The test dataset has the following shape: ", norm_test_images.shape)
print(". . . . . .  . . . . . .")
```

```
The training dataset has the following shape:  (24958, 64, 64, 3)
The test dataset has the following shape:  (2600, 64, 64, 3)
. . . . . .  . . . . . .
```

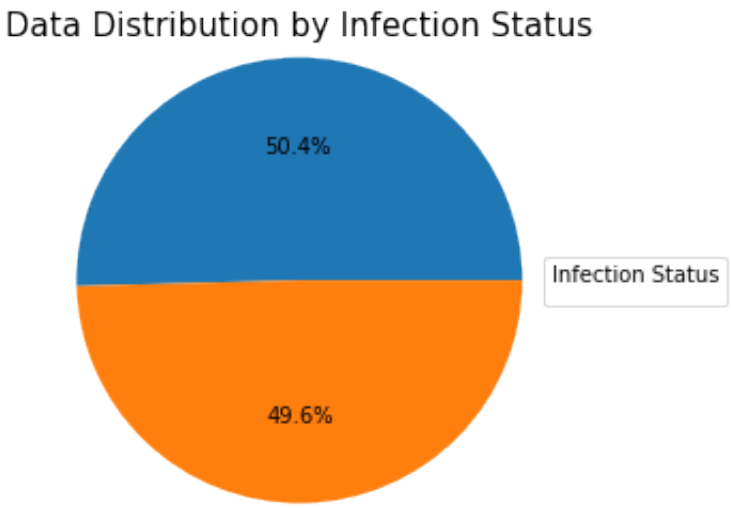
In [ ]:
```
#Plot the data to check for balance
#Bar Plot
set(df_train['Infection_Status'])
df_train['Infection_Status'].value_counts().plot(kind='bar', rot=0, color='s
plt.title('Data Distribution by Infection Status', fontsize=15)
plt.xlabel('Infection Status', fontsize=12)
plt.ylabel('Image Count', fontsize=12)
plt.text(x = -0.08, y = df_train.Infection_Status.value_counts()[0]+1, s = d
plt.text(x = 0.92, y = df_train.Infection_Status.value_counts()[1]+1, s = df
plt.show()
```



In [ ]:
```
#Count Plot to check for balance in test data
m=sns.countplot(data=df_train, x=df_train.Infection_Status, hue=df_train.Inf
plt.text(x=-0.3, y = df_train.Infection_Status.value_counts()[0]+1, s = df_t
plt.text(x=1.12, y = df_train.Infection_Status.value_counts()[1]+1, s = df_t

#Move the legend position to center, add titles, labels, display plot.
#sns.move_legend(m, "center")
sns.move_legend(m, "upper left", bbox_to_anchor=(1,1))
plt.title('Data Distribution by Infection Status', fontsize=15)
plt.xlabel('Infection Status', fontsize=12)
plt.ylabel('Image Count', fontsize=12)
plt.show()
```



In [ ]:
```
#Pie Plot to check for balance in test data
p=plt.pie(df_train.value_counts(), autopct = '%.1f%%', radius=1.2)

#Generate Plot Title and position labels to be seen on the plot
uninfected_train=[float(df_train.Infection_Status.value_counts()[0])]
infected_train=[float(df_train.Infection_Status.value_counts()[1])]
plt.title('Data Distribution by Infection Status', fontsize=15, loc="center"
plt.legend(uninfected_train, infected_train,
           title="Infection Status",
           loc="center left",
           bbox_to_anchor=(1,0,0.5,1))
plt.show()
```

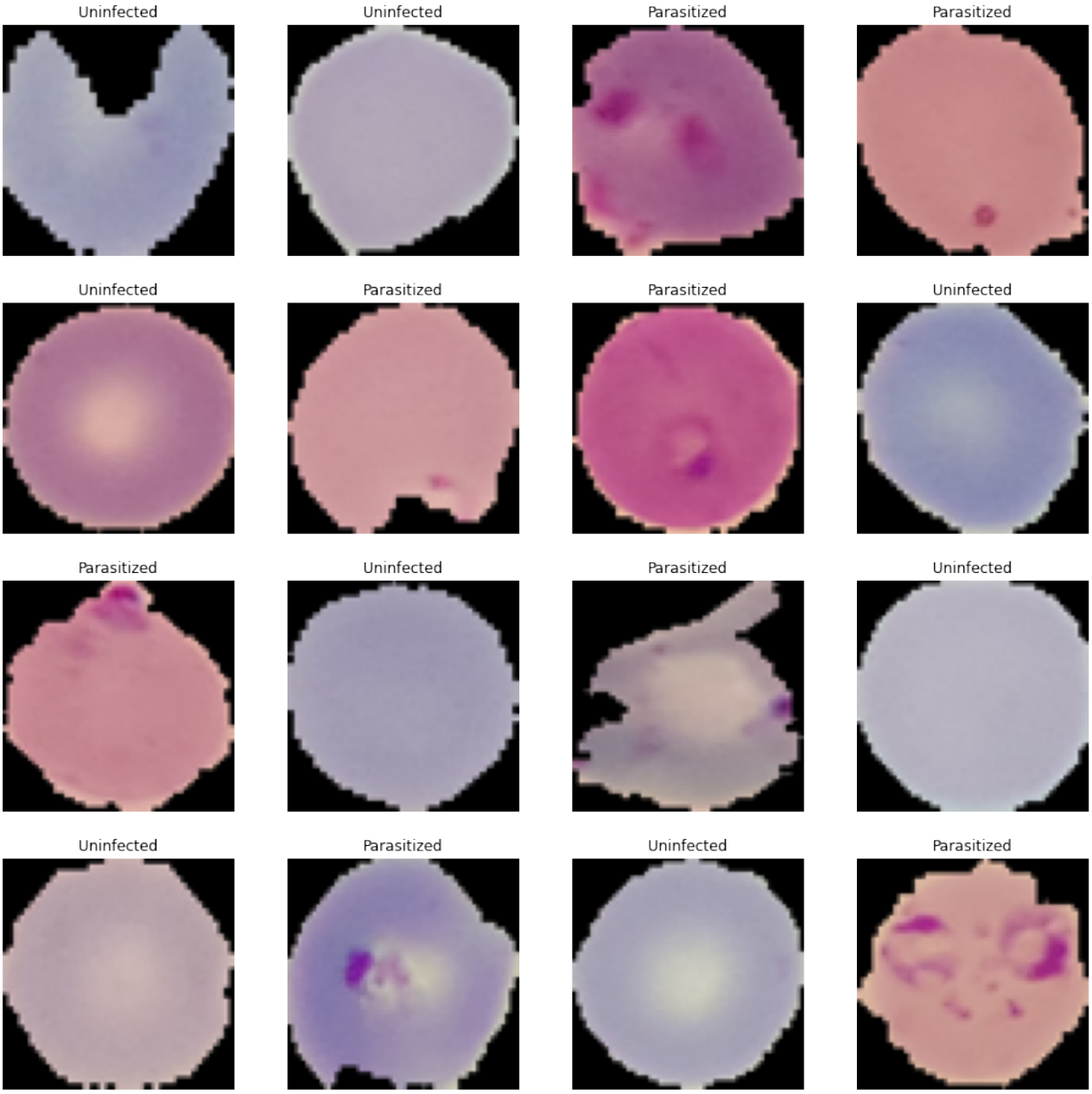


Observations about the data:

- Normalizing the data created a dataset with similar distribution effectively adds speed to processing the model. This normalization step also permits an even comparison of pixels in the model, with each pixel value now optimized for analysis. With the values now ranging from 0 to 1, the images containing parasites now stand out as significant values compared to those images without those hues.
- The count, bar, and pie plot all show an even number of infected and uninfected images in the test dataset. The balance in the infected and uninfected samples will permit clear learning of uninfected cells to better identify parasites in positive samples.

Data Exploration

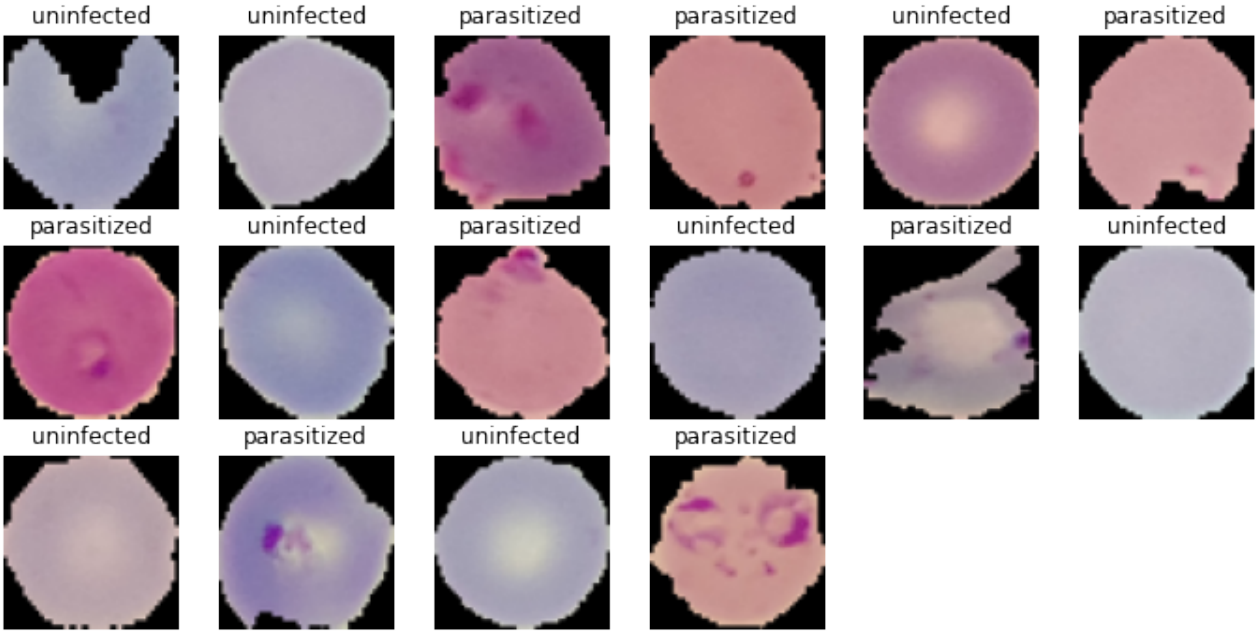
In [ ]:
```
#Visualize the images in the training dataset
np.random.seed(42)
plt.figure(1, figsize=(16,16))
for i in range (1,17):
  plt.subplot(4,4,i)
  index=int(np.random.randint(0,train_images.shape[0],1))
  if train_label[index]==1:
    plt.title('Parasitized')
  else:
    plt.title('Uninfected')
  plt.imshow(train_images[index])
  plt.axis('off')
plt.show()
```



Observations:

- The key features of the images have been reduced to the presence or absence of a dense pink to purple collection of color in the image. The shapes are inconsistent, and irregular in shape which prevents identification of the parasite life cycle but does clearly identify parasitized cells as containing dense pink to purple stained structures.
- The image normalization also removed most cellular debris which traditionally remains unstained or stains similar to the healthy red blood cells. The misidentification of cellular debris as a parasite has been eliminated through normalization.

In [ ]:
```
#Another visualization of the images
np.random.seed(42)
plt.figure(1, figsize=(12,12))
for i in range (1,17):
  plt.subplot(6,6,i)
  index=int(np.random.randint(0,train_images.shape[0],1))
  if train_label[index]==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
  plt.imshow(train_images[index])
  plt.axis('off')
plt.show()
```



Observations:

- As seen in the previous iteration of image classification; the presence of a dense, irregular pink-to-purple color in the image, clearly identifies parasitized cell structures.
- The 12x12 image appears clearer, with well-defined edges to the parasite when present in the image as compared to the larger 16x16 size image seen in the previous step.
- The color of the parasite present in the images is clearly defined from the background in the image. This suggests further processing should include leveraging the pigmentation of the stained parasite to enhance identification.

Plotting mean images for Infected and Uninfected Images

Before building the model the edges and shapes of the parasite must first be defined. To determine the edges a Gaussian filter can be used to extract the edges of the parasite selecting for the specific hue of the stained parasite. A Gaussian elimination method uses the image mean as a background subtraction element, centering the data around those values above the image mean.

In [ ]:
```
#Defining the function to calculate the mean images.
def find_mean_img(full_mat, title):
  #Calculating the average for each image
  mean_img=np.mean(full_mat, axis=0)[0]
  #Reshaping encode to matrix
  plt.imshow(mean_img)
  #Plot title
  plt.title('Average {title}')
  #Turn off axes labels
  plt.axis('off')
  plt.show()
  return mean_img

#Calculating the mean image for the infected cells, label=1
#Create container list for infected data mean image values
infected_data=[]
for img, label in zip(train_images, train_label):
  if label==1:
    infected_data.append([img])
#Use defined function to determine mean for each image
infected_mean=find_mean_img(np.array(infected_data), 'Infected')
print('Mean Image for Infected Cells:', infected_mean)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow w
ith RGB data ([0..1] for floats or [0..255] for integers).
```

```
Mean Image for Infected Cells: [[[0.07455095 0.05619138 0.05547608]
  [0.06922588 0.05269433 0.05221745]
  [0.14679701 0.1103958  0.11278016]
  ...
  [0.23732316 0.19138452 0.18653632]
  [0.1475918  0.11603879 0.11111111]
  [0.08925449 0.06763631 0.06525195]]

 [[0.08297568 0.06310602 0.06270863]
  [0.16237482 0.12096646 0.12199968]
  [0.37378795 0.28532825 0.2925608 ]
  ...
  [0.44190113 0.3605945  0.35113654]
  [0.23501828 0.18955651 0.18534414]
  [0.14099507 0.11095215 0.10681927]]

 [[0.13193451 0.09958671 0.09712287]
  [0.34771896 0.26164362 0.25886187]
  [0.75806708 0.58384994 0.59179781]
  ...
  [0.75552376 0.61476713 0.59744079]
  [0.3892068  0.31417899 0.306708  ]
  [0.13908759 0.10848832 0.10546813]]

 ...

 [[0.18081386 0.1371801  0.14012081]
  [0.28127484 0.21467175 0.21745351]
  [0.56088062 0.43760928 0.44198061]
  ...
  [0.64687649 0.51955174 0.52742012]
  [0.35844858 0.28564616 0.28707678]
  [0.14433317 0.10983945 0.10793197]]

 [[0.07725322 0.05778096 0.05809887]
  [0.12692736 0.09577174 0.09759975]
  [0.26609442 0.20362423 0.20942616]
  ...
  [0.38419965 0.30639008 0.30607217]
  [0.22373232 0.18081386 0.18025751]
  [0.09608965 0.07765061 0.07606104]]

 [[0.05189954 0.03981879 0.03838817]
  [0.06739787 0.05142267 0.0494357 ]
  [0.09767922 0.07502782 0.07455095]
  ...
  [0.15633445 0.12700684 0.12700684]
  [0.10077889 0.08392942 0.08345255]
  [0.07439199 0.05952949 0.05825783]]]
```

In [ ]:
```
#Defining the function to calculate the mean images.
def find_mean_img(full_mat, title):
  #Calculating the average for each image
  mean_img=np.mean(full_mat, axis=0)[0]
  #Reshaping encode to matrix
  plt.imshow(mean_img)
  #Plot title
  plt.title('Average {title}')
  #Turn off axes labels
  plt.axis('off')
  plt.show()
  return mean_img

#Calculating the mean image for the uninfected cells, label=0
#Create container list for uninfected data mean image values
uninfected_data=[]
for img, label in zip(train_images, train_label):
  if label==0:
    uninfected_data.append([img])
#Use defined function to determine mean for each image
uninfected_mean=find_mean_img(np.array(uninfected_data), 'Uninfected')
print('Mean Image for Uninfected Cells:', uninfected_mean)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow w
ith RGB data ([0..1] for floats or [0..255] for integers).
```

```
Mean Image for Uninfected Cells: [[[0.10754686 0.08944732 0.08928571]
  [0.19683258 0.16628959 0.16903685]
  [0.26866516 0.22769877 0.23327408]
  ...
  [0.17275372 0.14382676 0.14204913]
  [0.07886231 0.06423723 0.062298  ]
  [0.03910795 0.02989657 0.02844215]]

 [[0.10706206 0.09316419 0.09478022]
  [0.24442469 0.20353911 0.20685197]
  [0.44909502 0.36885908 0.37273756]
  ...
  [0.36158694 0.28975436 0.28967356]
  [0.16022948 0.12790886 0.12605042]
  [0.08492243 0.0674693  0.06585326]]

 [[0.18333872 0.15206852 0.15021008]
  [0.40853264 0.32958953 0.33306399]
  [0.77246283 0.62128313 0.63211054]
  ...
  [0.59841629 0.48408209 0.48739496]
  [0.29734971 0.23682935 0.23682935]
  [0.11546542 0.09558824 0.09518423]]

 ...

 [[0.19424693 0.15336134 0.15206852]
  [0.37112153 0.29557207 0.29427925]
  [0.61845507 0.49935359 0.49620233]
  ...
  [0.38720103 0.30631868 0.31286361]
  [0.22672915 0.18026826 0.18374273]
  [0.09251778 0.07449903 0.07700388]]

 [[0.11498061 0.09227537 0.09065934]
  [0.20733678 0.16612799 0.16531997]
  [0.3435682  0.27868455 0.27706852]
  ...
  [0.218649   0.17380414 0.1771978 ]
  [0.13914027 0.1093245  0.11142534]
  [0.07975113 0.06512605 0.06472204]]

 [[0.05235941 0.0418552  0.0414512 ]
  [0.11094053 0.08977052 0.08968972]
  [0.18818681 0.15465417 0.15457337]
  ...
  [0.12144473 0.09978992 0.0989819 ]
  [0.06876212 0.05526826 0.05591467]
  [0.0312702  0.02577569 0.02440207]]]
```

Observations:

- The image mean for images labeled parasitized have low mean values in the red and green channels of the image but the display ranges up to 0.1 in the blue channel in some cases. This is consistent with the blue stain color the parasite retains when stained.
- The image mean for the uninfected cells show higher image mean values in the red channel, ranging from 0.1 to 0.6 while the green and blue channels remain below 0.2. This is consistent with the stained red blood cells and pink color is seen in the uninfected slide without color in the blue channel where the parasites are seen.
- The image mean can be used to eliminate layers, adding a low pass filter to blur the image and assist in edge definition of the parasite using a Gaussian Blur as a filter.

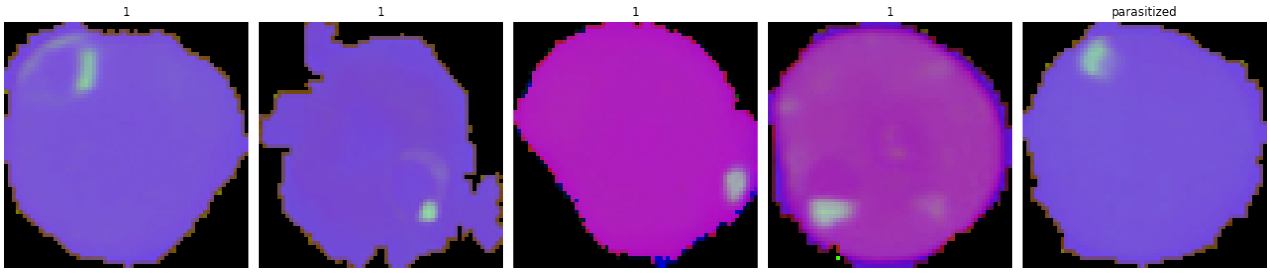
Image Processing for training data features.

The Giemsa-Wright is a differential light microscopy stain that utilizes a pinkish stain for red blood cells, and deep pink for platelets. *Plasmodium spp.* stain cytoplasms blue while parasitic nuclear structures uptake a red to purple stain inside the blue cytoplasm. Schüffner's dots and other inclusion bodies stain red.

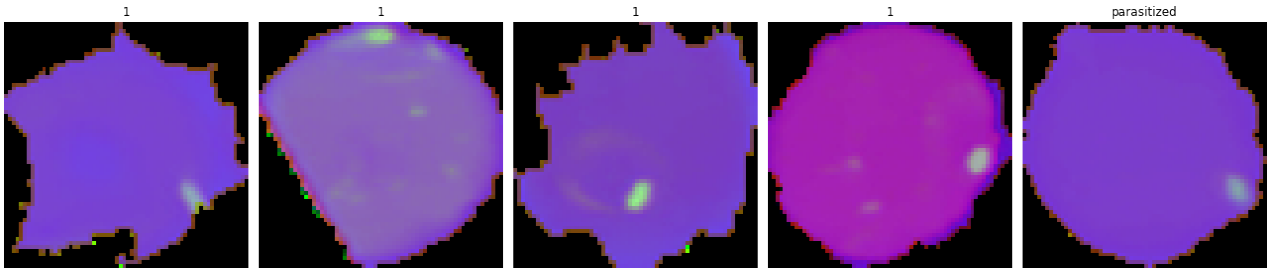
Knowing the colors of the structures when stained, a hue saturation value can be assigned to the identifying structures of *Plasmodium spp.*

In [ ]:

```
#Import ComputerVision image processing library
import cv2

#Create a List Container for Hue Saturation Value Array
gfx=[]

#Convert RBG data to HSV list
for i in np.arange(0,100,1):
  a=cv2.cvtColor(train_images[i], cv2.COLOR_BGR2HSV)
  gfx.append(a)
#Create HSV NumPy Array
gfx=np.array(gfx)

#Apply to and convert training data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=train_label[i]
  ax[t].set_title(Title)
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
```



In [ ]:
```
#Import ComputerVision image processing library
import cv2

#Create a List Container for Hue Saturation Value Array
gfx=[]

#Convert RBG data to HSV list
for i in np.arange(0,100,1):
  a=cv2.cvtColor(test_images[i], cv2.COLOR_BGR2HSV)
  gfx.append(a)
#Create HSV NumPy Array
gfx=np.array(gfx)

#Apply to and convert training data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=train_label[i]
  ax[t].set_title(Title)
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
```



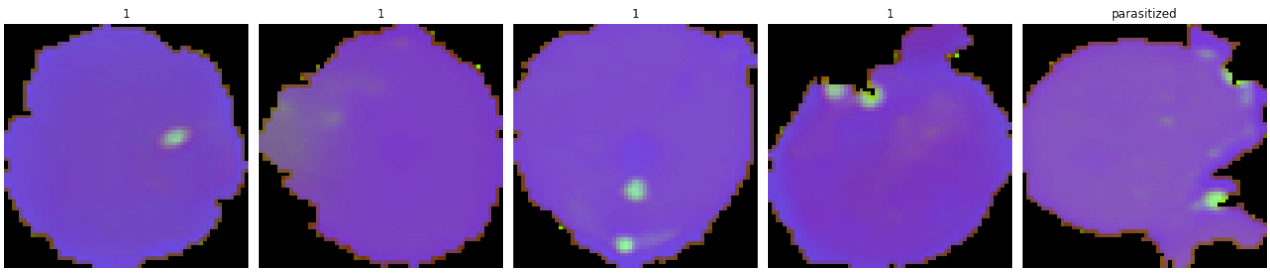
Observations:

- Applying the hue saturation filter allows the parasite to be distinctly seen as a green, irregular shape with high luminosity while the red blood cells and any additional non-parasite structures are flattened and removed from the image, leaving only parasites against a purple background.
- The distinctive color of the parasites now allows training to the edges of the parasite and visualization of the parasite by the machine learning model.
- Next steps include applying the Gaussian Blur as a filter to the image and defining key feature edges.

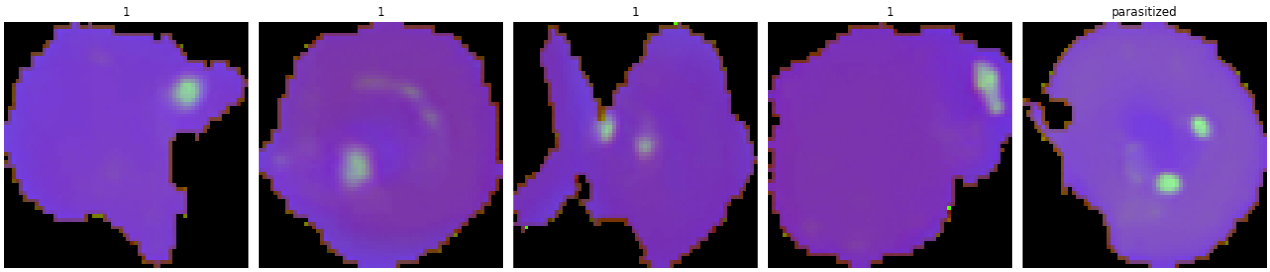
Processing Image Data with Gaussian Blurring

The Gaussian Blur is a filter applied to computer vision models to define the edges of images while reducing the noise in the image. This filter will allow contrastive learning between the parasitized and uninfected images to better detect the presence of parasites in the infected images.

In [ ]:
```
#Create container list to hold blurred image data
gbx=[]
#Convert image data to Gaussian Blur
for i in np.arange(0,100,1):
  b=cv2.GaussianBlur(train_images[i], (5,5),0)
  gbx.append(b)
#Pass gbx list to NumPy Array
gbx=np.array(gbx)

#Apply to and convert training data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=train_label[i]
  ax[t].set_title(Title)
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
```



In [ ]:
```
#Create container list to hold blurred image data
gbx=[]
#Convert image data to Gaussian Blur
for i in np.arange(0,100,1):
  b=cv2.GaussianBlur(train_images[i], (5,5),0)
  gbx.append(b)
#Pass gbx list to NumPy Array
gbx=np.array(gbx)

#Apply to and convert test data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=test_label[i]
  ax[t].set_title(Title)
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
```



Observations:

- The subtraction of the image mean, hue saturation, and subsequent Gaussian Blur highlights the presence of shapes consistent with the parasite as vibrant green against a purple background. This allows clear visualization of the parasite in the sample using machine learning.
- The relative value of the pixels in the green spectrum above a designated threshold could work as a prediction of the presence or absence of parasites in thin blood smears.
- As a high pass filter the Gaussian Blur establishes the edges of the image without adding noise and retaining key shape features.
- Building a Convolutional Neural Network is the next step, using the Gaussian Blur performance as a baseline. The model should flatten and pool the dense layers trained over several epochs and model iterations to yield a high-performing model with a high degree of accuracy. False negative samples could have serious health outcomes including mortality thus precision and accuracy must match or exceed the accuracy rate of traditional light microscopy.

Capstone Project: Deep Learning, Milestone 2

Model Building, Fit, and Evaluation

Exploratory Data Analysis

Processing Image Data

Reload unaugmented images to test and train directories and model testing.

In [ ]:
```
#Assigning train data for infected and uninfected
train_dir='/content/cell_images/train/'

#Designate image size for normality across data set.
SIZE=64

#Create list containers for the image data
train_images=[]

#Create a list container for Boolean value indicating infection state (0:uni
train_label=[]

#Loading Training data into List
for folder_name in ['/parasitized/','/uninfected/']:
  images_path=os.listdir(train_dir + folder_name)
  for i, image_name in enumerate(images_path):
    try:
      #Open each image in sequence as exception coding
      image=Image.open(train_dir + folder_name + image_name)
      #Resize Images to 64x64 to standardize to same shape
      image=image.resize((SIZE,SIZE))
      #Convert the images to list 'train_images' as designated above
      train_images.append(np.array(image))
      #Create the infected and uninfected labels with conditional logic
      if folder_name=='/parasitized/':
        train_label.append(1)
      else:
        train_label.append(0)
    except Exception:
      pass

#Convert list above into NumPy Array
train_images=np.array(train_images)
train_label=np.array(train_label)
```

In [ ]:
```
#Assigning test data for infected and uninfected
test_dir='/content/cell_images/test/'

#Designate image size for normality across data set.
SIZE=64

#Create list containers for the image data
test_images=[]

#Create a list container for Boolean value indicating infection state (0:uni
test_label=[]

#Loading Test data into List
for folder_name in ['/parasitized/','/uninfected/']:
  images_path=os.listdir(test_dir + folder_name)
  for i, image_name in enumerate(images_path):
    try:
      #Open each image in sequence as exception coding
      image=Image.open(test_dir + folder_name + image_name)
      #Resize Images to 64x64 to standardize to same shape
      image=image.resize((SIZE,SIZE))
      #Convert the images to list 'train_images' as designated above
      test_images.append(np.array(image))
      #Create the infected and uninfected labels with conditional logic
      if folder_name=='/parasitized/':
        test_label.append(1)
      else:
        test_label.append(0)
    except Exception:
      pass

#Convert list above into NumPy Array
test_images=np.array(test_images)
test_label=np.array(test_label)
```

In [ ]:
```
#Pass NumPy Array to Pandas DataFrame
df_train=pd.DataFrame(train_label, columns=['Infection_Status'])
```

Data Preparation

To prepare the images for machine learning the data shape and array must be determined. The data must then be normalized to ensure each pixel has a similar distribution.

In [ ]:
```
#Normalize the data.  There are 255 pixels thus all data will be normalized
train_images = (train_images/255).astype('float32')
test_images = (test_images/255).astype('float32')
```

One Hot Encoding on the train and test labels

Categorical data seen in the labels identifying the infection status must be converted into vector array. This conversion makes the data scalable, converting the string data 'Parasitized' and 'Uninfected' into numerical values that can be read by the machine learning model. This vectorized format is easily read by the machine learning algorithm.

In [ ]:
```
#Model Building library
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Activation, BatchNormali
from tensorflow.keras.losses import categorical_crossentropy, binary_crossen
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.utils import to_categorical

print(tf.__version__)
```

```
2.9.2
```

In [ ]:
```
# Encoding Train Labels
train_label = to_categorical(train_label, 2)

# Similarly let us try to encode test labels
test_label = to_categorical(test_label, 2)

#Visualize the new vector array
print(train_label)
```

```
[[0. 1.]
 [0. 1.]
 [0. 1.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

Base Model

The basic model will take the input layer and pass the image data through hidden layer neurons as specified in the model. The data is passed then to an output layer which processes the data and returns an output to the model.

In [ ]:

```
# Clearing backend
from tensorflow.keras import backend
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dr
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from random import shuffle

backend.clear_session()

# Fixing the seed for random number generators so that we can ensure we rece
np.random.seed(42)
import random
random.seed(42)
tf.random.set_seed(42)
```

Building the Base Model

The Convolutional Neural Network (CNN) is a series of operations applied to an input image that translates the information into a machine learning algorithm creating a predictive analytic from trainable features. Images are two dimensional and the Conv2D will designate each layer in the model. The CNN is comprised of multiple layers, each building on the previous to extract the key features and allow interpretation of those features using labels in the data.

The base layer begins a class that passes the attributes to each layer. To find the ideal parameters to apply to the images in the dataset layers are created that allow the model to interpret the values.

- Core layers include activations designate the functions used to generate the tensors passed to the hidden layers in the next layer of neurons. In particular the relu function is a rectified linear activation function that generates a max value based on the input shapes.
- Dense layers act as core layers of the CNN generated by the dot product of the input matrix and kernel, designating the dimensionality of the output space. Filters designate the number of output filters in each convolution while the kernel size specifies the height and width of the convolutional window.
- Pooling layers use the input height and width to return the maximum value from the designated window size allowing the window to shift with each dimension in the input layers.

- Normalization layers normalize the continuous features by scaling and/or shifting inputs based on the mean values and variance. This acts as a background subtraction allowing weight to be added to significant values.

- Regularization layers randomly assign zero input values to help prevent overfitting during training.
- Reshaping layers These methods include flattening which reshape the input thereby reflecting a change in the dimensionality of the output.

In [ ]:

```
# Creating sequential model
def cnn_model():
  model = Sequential()
  model.add(Conv2D(filters = 32, kernel_size = 2,
                   padding = "same",
                   activation = "relu",
                   input_shape = (64, 64, 3)))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))

  model.add(Conv2D(filters = 32, kernel_size = 2,
                   padding = "same",
                   activation = "relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))

  model.add(Conv2D(filters = 32, kernel_size = 2,
                   padding = "same",
                   activation = "relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  model.add(Flatten())
  model.add(Dense(512, activation = "relu"))

  model.add(Dropout(0.4))
  model.add(Dense(2, activation = "softmax")) #Output Layer Neurons: 2

#Compile
  model.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])
  return model

#Build the model
model=cnn_model()

#Print Model Summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        416

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        4128

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        4128

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout_2 (Dropout)         (None, 8, 8, 32)          0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 1,058,786
Trainable params: 1,058,786
Non-trainable params: 0
_________________________________________________________________
```

Compiling the Model Observations

Loss functions act to compute the value that a model should seek to minimize during training. Binary crossentropy compares the predicted probability distribution to the actual output from the model. This becomes a penalty score based on the difference of the calculated probabilities from the expected value. The close these values are to the actual value, the better the predictions from the model. To calculate success of the model and locate improvement, it is expected to see a training accuracy as close to 100% as possible with minimal loss.

Using Callbacks

Callbacks are a series of methods that allow periodic saving of the model to a disk when data sets are large as seen in the total trainable parameters above. The use of early stopping allows the model to fit, stopping the number of iterations of the model when a specified parameter such as validation loss does not incrementally change by a specified amount during each epoch. This stopping indicates a point at which no additional training can be achieved without considerable loss to the model. EarlyStopping also prevents excessive use of computing power when the model is at peak efficiency. The patience score designates how many additional epoch cycles the model will perform before early termination. When there is no further improvement in the model, the training stops.

In [ ]:
```
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('.mdl_wts.hdf5', monitor='val_loss', save_best_only=True
```

Fit and Train the Model

Fitting and training the data to the model uses the methods designated in the algorithm and begins to apply each technique to the layers. The algorithm uses incremental performance processing a designated number of samples each iteration until all the samples are processed. The batch size affects the speed of the model as well as the performance. The total number of samples designated in the batch size are loaded into the memory and the larger the batch size the more memory required to process each layer. A balance between the number of samples in the batch must be achieved with the number of epochs to achieve the optimal efficiency and accuracy of the model. Smaller batch sizes process more quickly but with lower accuracy.

In [ ]:
```
# Fit the model with min batch size as 32 can tune batch size to some factor
history = model.fit(train_images,
                    train_label,
                    batch_size = 64,
                    callbacks = callbacks,
                    validation_split = 0.2,
                    epochs = 20,
                    verbose = 1)
```

```
Epoch 1/20
312/312 [==============================] - 104s 332ms/step - loss: 0.5016 -
accuracy: 0.7480 - val_loss: 0.2610 - val_accuracy: 0.8910
Epoch 2/20
312/312 [==============================] - 94s 301ms/step - loss: 0.1406 - a
ccuracy: 0.9441 - val_loss: 0.1483 - val_accuracy: 0.9679
Epoch 3/20
312/312 [==============================] - 102s 326ms/step - loss: 0.1112 -
accuracy: 0.9621 - val_loss: 0.1287 - val_accuracy: 0.9782
Epoch 4/20
312/312 [==============================] - 93s 299ms/step - loss: 0.0968 - a
ccuracy: 0.9682 - val_loss: 0.0831 - val_accuracy: 0.9872
Epoch 5/20
312/312 [==============================] - 94s 301ms/step - loss: 0.0866 - a
ccuracy: 0.9714 - val_loss: 0.1399 - val_accuracy: 0.9804
Epoch 6/20
312/312 [==============================] - 105s 338ms/step - loss: 0.0776 -
accuracy: 0.9742 - val_loss: 0.0953 - val_accuracy: 0.9806
Epoch 7/20
312/312 [==============================] - 97s 311ms/step - loss: 0.0717 - a
ccuracy: 0.9752 - val_loss: 0.0533 - val_accuracy: 0.9884
Epoch 8/20
312/312 [==============================] - 97s 312ms/step - loss: 0.0673 - a
ccuracy: 0.9768 - val_loss: 0.0776 - val_accuracy: 0.9840
Epoch 9/20
312/312 [==============================] - 97s 310ms/step - loss: 0.0645 - a
ccuracy: 0.9766 - val_loss: 0.0522 - val_accuracy: 0.9902
Epoch 10/20
312/312 [==============================] - 93s 298ms/step - loss: 0.0599 - a
ccuracy: 0.9791 - val_loss: 0.0483 - val_accuracy: 0.9896
Epoch 11/20
312/312 [==============================] - 101s 324ms/step - loss: 0.0637 -
accuracy: 0.9786 - val_loss: 0.0556 - val_accuracy: 0.9878
Epoch 12/20
312/312 [==============================] - 97s 309ms/step - loss: 0.0611 - a
ccuracy: 0.9779 - val_loss: 0.0554 - val_accuracy: 0.9854
Epoch 13/20
312/312 [==============================] - 100s 321ms/step - loss: 0.0570 -
accuracy: 0.9795 - val_loss: 0.0752 - val_accuracy: 0.9834
Epoch 14/20
312/312 [==============================] - 91s 292ms/step - loss: 0.0566 - a
ccuracy: 0.9790 - val_loss: 0.0667 - val_accuracy: 0.9848
Epoch 15/20
312/312 [==============================] - 96s 306ms/step - loss: 0.0575 - a
ccuracy: 0.9792 - val_loss: 0.0679 - val_accuracy: 0.9844
Epoch 15: early stopping
```

Evaluating the Model on Test Data

It is expected that a minimum of 90% should be achieved for the base model. This allows room to improve the model through adjustment of the batch size, callback parameters, and number of layers.

In [ ]:
```
accuracy = model.evaluate(test_images, test_label, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

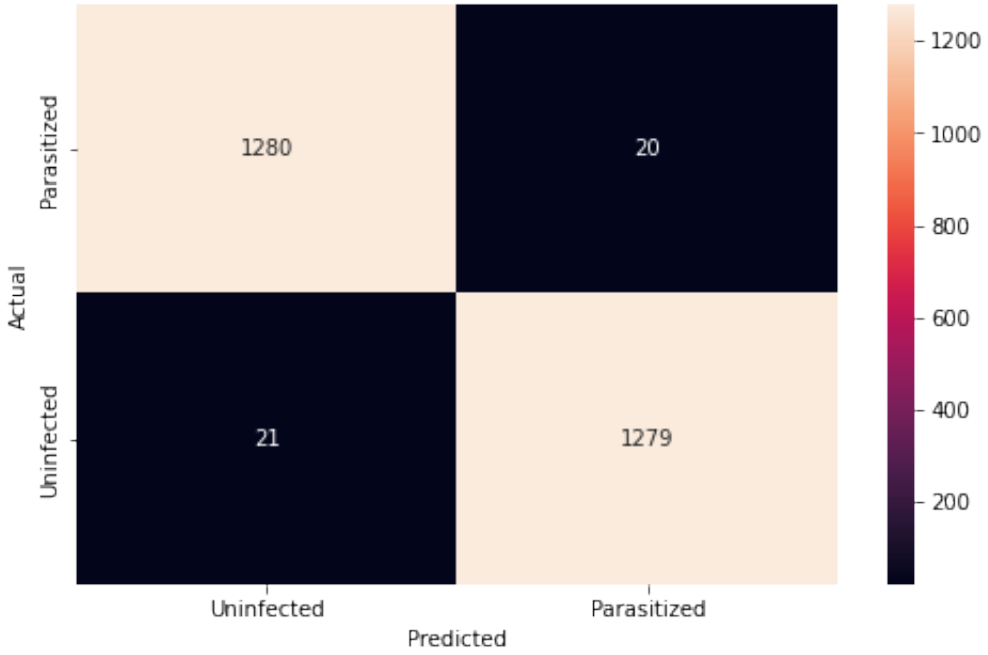
```
82/82 [==============================] - 3s 40ms/step - loss: 0.0580 - accur
acy: 0.9842

 Test_Accuracy:- 0.9842307567596436
```

Plotting the Confusion Matrix

The test accuracy was calculated indicating the model can correctly identify the presence of a parasite infection at least 98.42% accuracy. This should be supported in the confusion matrix through a high degree of accuracy and values for predicted and actual that are similar. The goal is to have the prediction and actual values as close as possible.

In [ ]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

pred = model.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_label, axis = 1)

# Printing the classification report
print(classification_report(y_true, pred))

# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f',
            xticklabels = ['Uninfected', 'Parasitized'],
            yticklabels = ['Parasitized', 'Uninfected'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

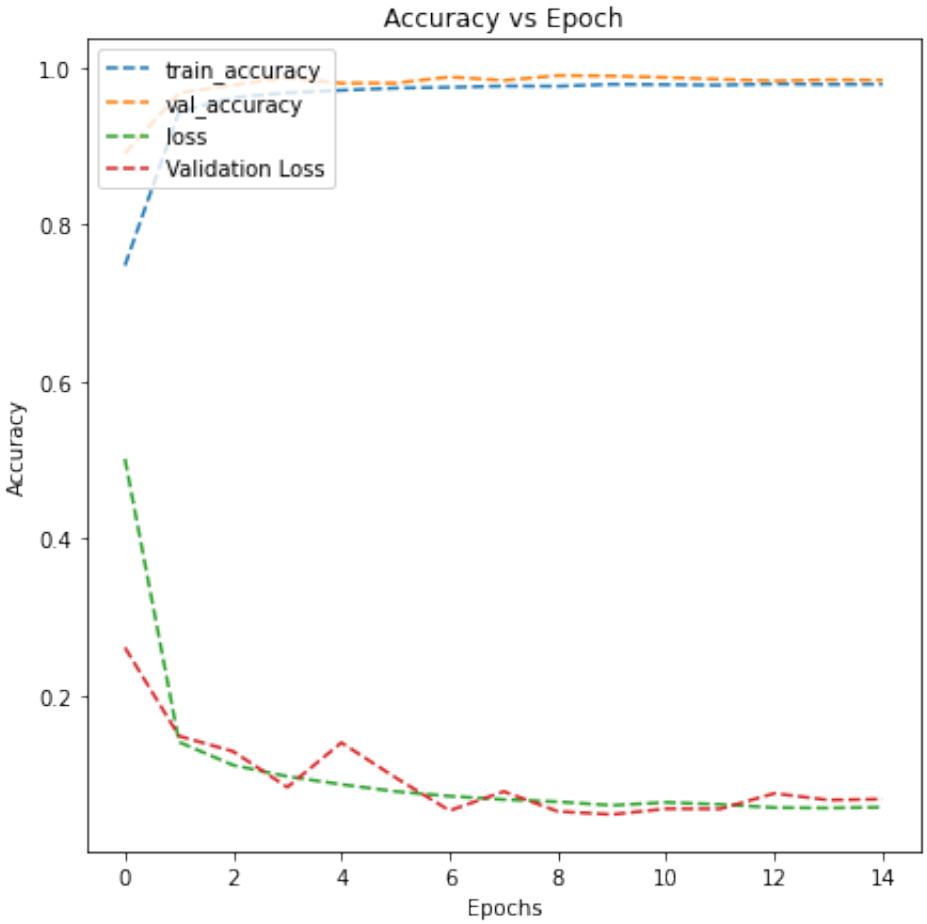
```
82/82 [==============================] - 3s 40ms/step
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1300
           1       0.98      0.98      0.98      1300

    accuracy                           0.98      2600
   macro avg       0.98      0.98      0.98      2600
weighted avg       0.98      0.98      0.98      2600
```



The model predicts the presence of the parasite with 98% precision indicating a high connection between the trainable key features used to identify the parasite in blood smears. The recall shows the model can correctly identify the parasite despite variability in the images which could include cellular debris. The F1 score demonstrates the precision and accuracy in predictions. A high F1 score indicates high precision in the predictions made by the algorithm. There is only one more parasitized cell not predicted by the model. This is a concern as using this model would produce a false negative. This could result in treatment not being prescribed and lead to death if left untreated.

Plotting and Training Validation Curves

In [ ]:
```
# Function to plot train and validation accuracy
def plot_accuracy(history):
    N = len(history.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history.history["accuracy"],
             label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_accuracy"],
             label = "val_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["loss"],
             label = "loss", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_loss"],
             label = "Validation Loss", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")

plot_accuracy(history)
```



Observations

- Comparison of the training accuracy to the validation accuracy shows a similar curve pattern to the training model for the test data. The model seems to have a similar performance from roughly five cycles through to the end.
- This particular model was stopped early at 15 epochs.
- As the accuracy of the models improves the loss function and validation loss function decline logrithmically, tapering to below 0.2 by five epochs. As the model continues to perform across the twenty epochs, the loss function remains minimized. This data suggests the model is a good fit for the data.
- The next model should expand on this base model to improve the precision and accuracy. This can be achieved by adding layers to the existing model. Adding a fourth convolutional layer, pooling layer, core layer as dense, and dropout layer should improve the fitness of the model without overfitting.

Model 1

To expand on the base model additional layers can be added to improve performance. In the base model the batch size was structured to 64 images per batch which is balanced between compute engine use and model performance.

- Add a fourth convolutional layer to the model. This will add an additional tensor of outputs.
- Add a max pooling layer to find the maximum values and exploit key features.
- Add dropout layer to prevent overfitting.

In [ ]:
```
backend.clear_session() # Clearing the backend for new model
```

In [ ]:

```
# Creating sequential model
def cnn_model_1():
  model = Sequential()
  #First Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = 2,
                   padding = "same",
                   activation = "relu",
                   input_shape = (64, 64, 3)))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Second Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = 2,
                   padding = "same", activation = "relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Third Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = 2,
                   padding = "same", activation = "relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Fourth Convolutional Layer
  model.add(Conv2D(filters = 64, kernel_size = 2,
                   padding = "same", activation = "relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))

  #Flatten previous layer
  model.add(Flatten())
  #Add Density Layer
  model.add(Dense(512, activation = "relu"))
  model.add(Dropout(0.4))

  #Output Layer
  model.add(Dense(2, activation = "softmax")) # 2 represents output layer ne

  #Compile
  model.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])
  return model

#Build the model
model_1=cnn_model_1()

#Print Model Summary
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        416

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        4128

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        4128

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout_2 (Dropout)         (None, 8, 8, 32)          0

 conv2d_3 (Conv2D)           (None, 8, 8, 64)          8256

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_3 (Dropout)         (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 512)               524800

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 542,754
Trainable params: 542,754
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
#Callback
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('.mdl_wts.hdf5', monitor='val_loss', save_best_only=True
```

In [ ]:
```
#Fit and train the model
history_1 = model_1.fit(train_images,
                    train_label,
                    batch_size = 64,
                    callbacks = callbacks,
                    validation_split = 0.2,
                    epochs = 20,
                    verbose = 1)
```

```
Epoch 1/20
312/312 [==============================] - 98s 311ms/step - loss: 0.5271 - a
ccuracy: 0.7260 - val_loss: 0.1560 - val_accuracy: 0.9441
Epoch 2/20
312/312 [==============================] - 92s 294ms/step - loss: 0.1109 - a
ccuracy: 0.9607 - val_loss: 0.0700 - val_accuracy: 0.9890
Epoch 3/20
312/312 [==============================] - 104s 332ms/step - loss: 0.0815 -
accuracy: 0.9710 - val_loss: 0.0626 - val_accuracy: 0.9878
Epoch 4/20
312/312 [==============================] - 92s 296ms/step - loss: 0.0782 - a
ccuracy: 0.9740 - val_loss: 0.0427 - val_accuracy: 0.9900
Epoch 5/20
312/312 [==============================] - 97s 310ms/step - loss: 0.0732 - a
ccuracy: 0.9744 - val_loss: 0.0904 - val_accuracy: 0.9754
Epoch 6/20
312/312 [==============================] - 97s 310ms/step - loss: 0.0712 - a
ccuracy: 0.9757 - val_loss: 0.0770 - val_accuracy: 0.9808
Epoch 7/20
312/312 [==============================] - 92s 296ms/step - loss: 0.0694 - a
ccuracy: 0.9760 - val_loss: 0.0415 - val_accuracy: 0.9884
Epoch 8/20
312/312 [==============================] - 101s 324ms/step - loss: 0.0664 -
accuracy: 0.9771 - val_loss: 0.0461 - val_accuracy: 0.9876
Epoch 9/20
312/312 [==============================] - 93s 297ms/step - loss: 0.0652 - a
ccuracy: 0.9778 - val_loss: 0.0669 - val_accuracy: 0.9826
Epoch 10/20
312/312 [==============================] - 97s 311ms/step - loss: 0.0618 - a
ccuracy: 0.9775 - val_loss: 0.0431 - val_accuracy: 0.9874
Epoch 11/20
312/312 [==============================] - 97s 310ms/step - loss: 0.0619 - a
ccuracy: 0.9786 - val_loss: 0.0410 - val_accuracy: 0.9878
Epoch 12/20
312/312 [==============================] - 93s 297ms/step - loss: 0.0593 - a
ccuracy: 0.9785 - val_loss: 0.0617 - val_accuracy: 0.9824
Epoch 13/20
312/312 [==============================] - 101s 323ms/step - loss: 0.0596 -
accuracy: 0.9789 - val_loss: 0.0538 - val_accuracy: 0.9850
Epoch 14/20
312/312 [==============================] - 92s 296ms/step - loss: 0.0575 - a
ccuracy: 0.9794 - val_loss: 0.0489 - val_accuracy: 0.9852
Epoch 15/20
312/312 [==============================] - 97s 310ms/step - loss: 0.0597 - a
ccuracy: 0.9786 - val_loss: 0.0670 - val_accuracy: 0.9810
Epoch 16/20
312/312 [==============================] - 97s 310ms/step - loss: 0.0616 - a
ccuracy: 0.9778 - val_loss: 0.0592 - val_accuracy: 0.9814
Epoch 16: early stopping
```

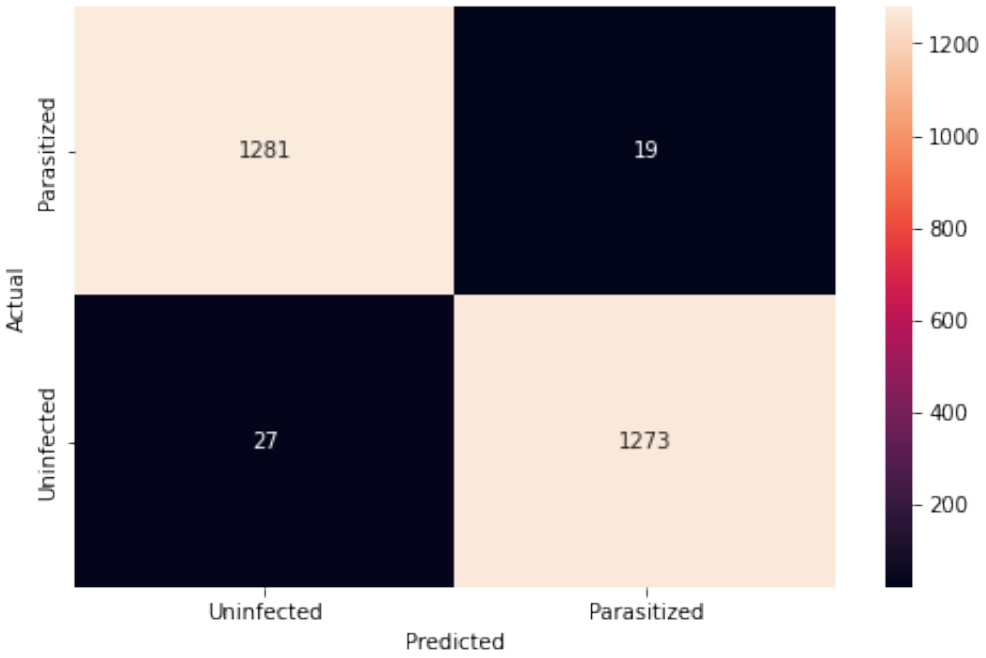
In [ ]:

```
#Evaluate the model
accuracy = model_1.evaluate(test_images, test_label, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

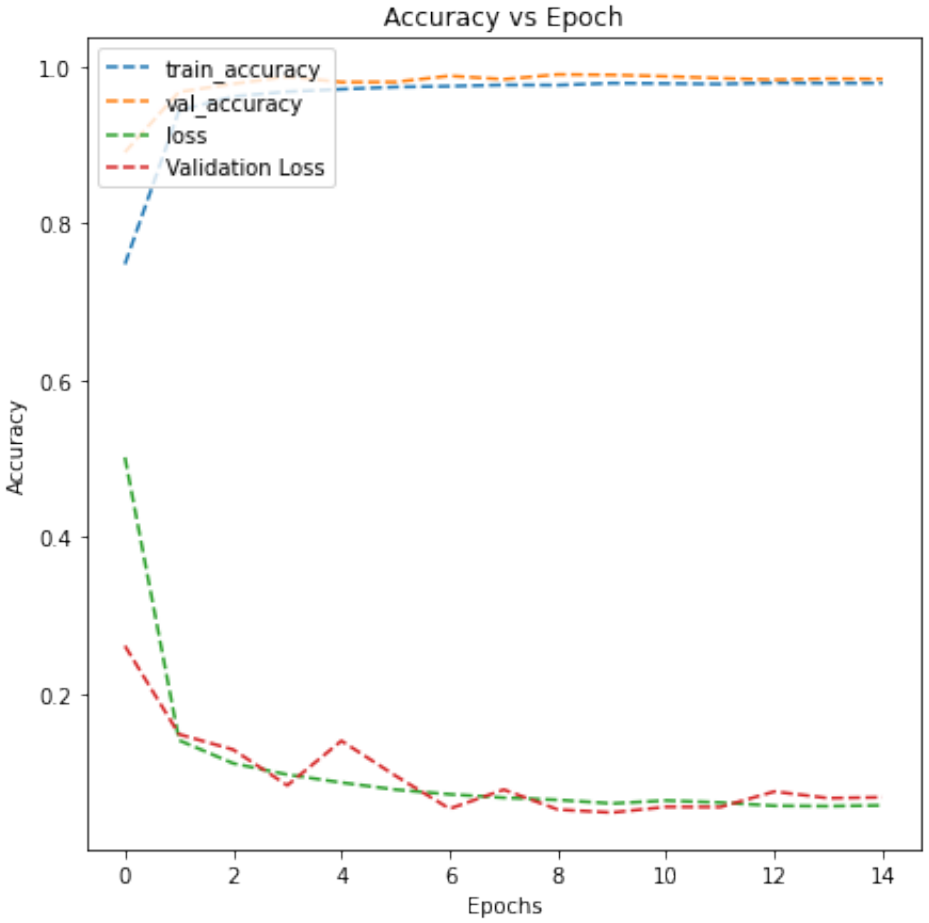
```
82/82 [==============================] - 3s 40ms/step - loss: 0.0523 - accur
acy: 0.9823

 Test_Accuracy:- 0.9823076725006104
```

In [ ]:
```python
#Plotting Confusion Matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

pred = model_1.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_label, axis = 1)

# Printing the classification report
print(classification_report(y_true, pred))

# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f',
            xticklabels = ['Uninfected', 'Parasitized'],
            yticklabels = ['Parasitized', 'Uninfected'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
82/82 [==============================] - 3s 39ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1300
           1       0.99      0.98      0.98      1300

    accuracy                           0.98      2600
   macro avg       0.98      0.98      0.98      2600
weighted avg       0.98      0.98      0.98      2600
```



In [ ]:
```
#Plotting training and validation curves
# Function to plot train and validation accuracy
def plot_accuracy(history_1):
    N = len(history.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history.history["accuracy"],
             label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_accuracy"],
             label = "val_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["loss"],
             label = "loss", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_loss"],
             label = "Validation Loss", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")

plot_accuracy(history_1)
```



Observations:

Adding layers to the model has improved overall performance as compared to the base model. The confusion matrix supports the plot showing accuracy across epochs with 98% precision, recall, and F1 score.

- The predicted infected cells are eight samples lower than the actual number, indicating an increase in false negative rate from the base model.
- Mild overfitting is observed in the model suggesting the Dropout Layer may have reduced some of the overfitting but did not eliminate overfitting of the model to the data.
- The model performance can be further improved through use of the LeakyRelu function. The LeakyReLU function leaks small positive samples close to zero allowing additional balance. As the ReLU values approach zero the model learning rate slows, and no additional performance is achieved. Through the random leaking of values close to zero the model can continue to learn and improve recall.
- Adding Batch Normalization will use the mean and variance to normalize the data, shifting the distribution close to the actual values with each epoch.

Model 2

To expand on the previous model, the batch size will be increased to 128 images per batch which leverages more compute engine use to improve model performance.

- Increase the kernel size to a 3x3, a larger kernel size will reduce the compute power used by increasing the batch size.
- Change the activation to a LeakyRelu to improve model performance, balancing the data.
- Add a Batch Normalization layer to shift the distribution closer to Normal.

In [ ]:
```
#Model 2
backend.clear_session() # Clearing the backend for new model
```

In [ ]:

```
#Building the Model
def cnn_model_2():
  model = Sequential()
  #First Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same",
                   activation = "leaky_relu",
                   input_shape = (64, 64, 3)))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Second Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Third Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Fourth Convolutional Layer
  model.add(Conv2D(filters = 64, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))

  #Add a BatchNormalization Layer
  model.add(tf.keras.layers.BatchNormalization())
  #Flatten previous layer
  model.add(Flatten())
  #Add Density Layer
  model.add(Dense(512, activation = "leaky_relu"))
  model.add(Dropout(0.4))

  #Output Layer
  model.add(Dense(2, activation = "softmax")) # 2 represents output layer ne

  #Compile
  model.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])
  return model

#Build the model
model_2=cnn_model_2()

#Print Model Summary
model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout_2 (Dropout)         (None, 8, 8, 32)          0

 conv2d_3 (Conv2D)           (None, 8, 8, 64)          18496

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_3 (Dropout)         (None, 4, 4, 64)          0

 batch_normalization (BatchN  (None, 4, 4, 64)         256
 ormalization)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 512)               524800

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 563,970
Trainable params: 563,842
Non-trainable params: 128
_________________________________________________________________
```

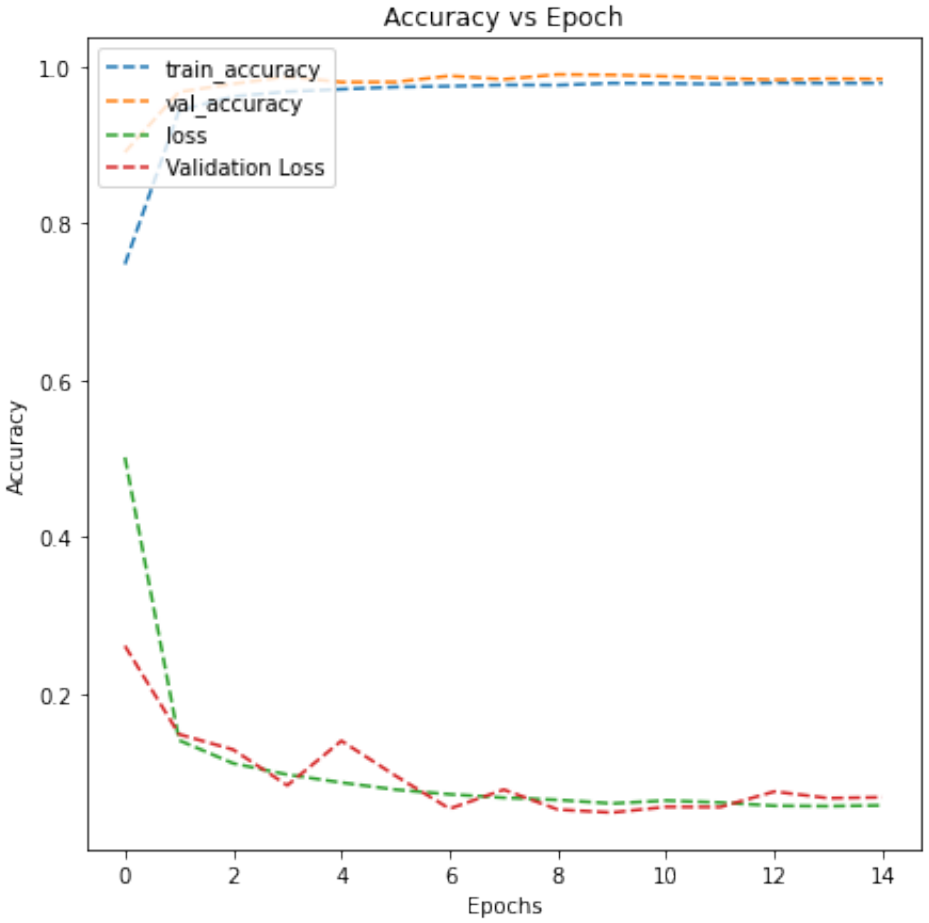
In [ ]:

```
#Callbacks will help us in saving our checkpoints and stopping at an accurac
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('.mdl_wts.hdf5', monitor='val_loss', save_best_only=True
```

In [ ]:
```
#Fit and Train Model
history_2 = model_2.fit(train_images,
                    train_label,
                    batch_size = 128,
                    callbacks = callbacks,
                    validation_split = 0.2,
                    epochs = 20,
                    verbose = 1)
```

```
Epoch 1/20
156/156 [==============================] - 155s 987ms/step - loss: 0.5624 -
accuracy: 0.7176 - val_loss: 0.6548 - val_accuracy: 0.8664
Epoch 2/20
156/156 [==============================] - 141s 902ms/step - loss: 0.1409 -
accuracy: 0.9484 - val_loss: 0.6554 - val_accuracy: 0.7951
Epoch 3/20
156/156 [==============================] - 149s 957ms/step - loss: 0.0938 -
accuracy: 0.9675 - val_loss: 0.1450 - val_accuracy: 0.9834
Epoch 4/20
156/156 [==============================] - 149s 957ms/step - loss: 0.0831 -
accuracy: 0.9721 - val_loss: 0.0567 - val_accuracy: 0.9932
Epoch 5/20
156/156 [==============================] - 140s 896ms/step - loss: 0.0786 -
accuracy: 0.9724 - val_loss: 0.0782 - val_accuracy: 0.9854
Epoch 6/20
156/156 [==============================] - 148s 953ms/step - loss: 0.0783 -
accuracy: 0.9732 - val_loss: 0.0854 - val_accuracy: 0.9840
Epoch 7/20
156/156 [==============================] - 140s 896ms/step - loss: 0.0722 -
accuracy: 0.9759 - val_loss: 0.0535 - val_accuracy: 0.9912
Epoch 8/20
156/156 [==============================] - 149s 955ms/step - loss: 0.0718 -
accuracy: 0.9752 - val_loss: 0.0643 - val_accuracy: 0.9890
Epoch 9/20
156/156 [==============================] - 149s 955ms/step - loss: 0.0709 -
accuracy: 0.9762 - val_loss: 0.0684 - val_accuracy: 0.9868
Epoch 10/20
156/156 [==============================] - 140s 898ms/step - loss: 0.0668 -
accuracy: 0.9774 - val_loss: 0.0656 - val_accuracy: 0.9880
Epoch 11/20
156/156 [==============================] - 148s 949ms/step - loss: 0.0687 -
accuracy: 0.9771 - val_loss: 0.0676 - val_accuracy: 0.9860
Epoch 12/20
156/156 [==============================] - 139s 890ms/step - loss: 0.0648 -
accuracy: 0.9785 - val_loss: 0.0722 - val_accuracy: 0.9858
Epoch 12: early stopping
```

In [ ]:
```
#Function to plot train and validation accuracy
def plot_accuracy(history_2):
    N = len(history.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history.history["accuracy"],
             label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_accuracy"],
             label = "val_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["loss"],
             label = "loss", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_loss"],
             label = "Validation Loss", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")

plot_accuracy(history_2)
```

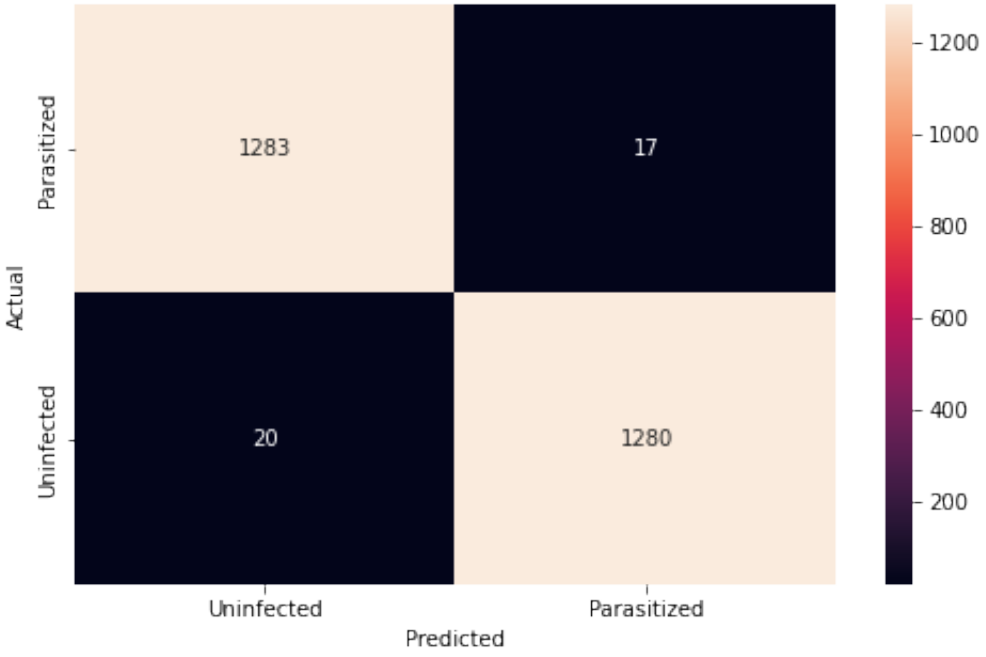


In [ ]:
```
# Evaluate the model to calculate the accuracy
accuracy_2= model_2.evaluate(test_images, test_label, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

```
82/82 [==============================] - 5s 64ms/step - loss: 0.0526 - accur
acy: 0.9858

 Test_Accuracy:- 0.9823076725006104
```

In [ ]:
```
#Generate the classification report and confusion matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

pred = model_2.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_label, axis = 1)

# Printing the classification report
print(classification_report(y_true, pred))

# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f',
            xticklabels = ['Uninfected', 'Parasitized'],
            yticklabels = ['Parasitized', 'Uninfected'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
82/82 [==============================] - 5s 63ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1300
           1       0.99      0.98      0.99      1300

    accuracy                           0.99      2600
   macro avg       0.99      0.99      0.99      2600
weighted avg       0.99      0.99      0.99      2600
```



Observations :

- The model has improved overall performance as compared to the Model 1. The confusion matrix supports the validation accuracy across epochs plot showing 99% precision, 98% recall, and 99% F1 score for infected cells.
- There are three false negatives from this model, a reduction from eight in the previous model. The misdiagnosis of three samples could represent serious negative health outcomes. While this is better precision compared to the previously run models, additional work should be performed to prevent false negatives.
- Overfitting is not observed in the model suggesting the Dropout Layer was successful in preventing overfitting of the model to the data.
- The model performance is further improved through use of the LeakyRelu function.
- Batch Normalization was successful normalizing the data with each epoch.

Model 3: Image Augmentation

The previous model assumes the parasite is static in size and shape without variability. This is not true of the biology of the parasite which changes shape and size throughout the life cycle. This model should be capable of selecting the parasite from a blood smear at any stage of the life cycle. This would require training the samples against a variety of possibilities in shape, size, and rotation of the images. Can the parasite be detected as well if the images are zoomed in, blurring focus? Can the parasite be detected if the images are rotated? The ImageDataGenerator allows alteration of the images as a batch method to determine if the changes affect the detection, accuracy, and precision rates. Rotation, magnification, and horizontal flipping of the image changes the trainable parameters through adjustments to the size and shape of the key features.

In [ ]:
```
#Model 3 with Data Augmentation
backend.clear_session() # Clearing backend for new model
```

Using image data generator

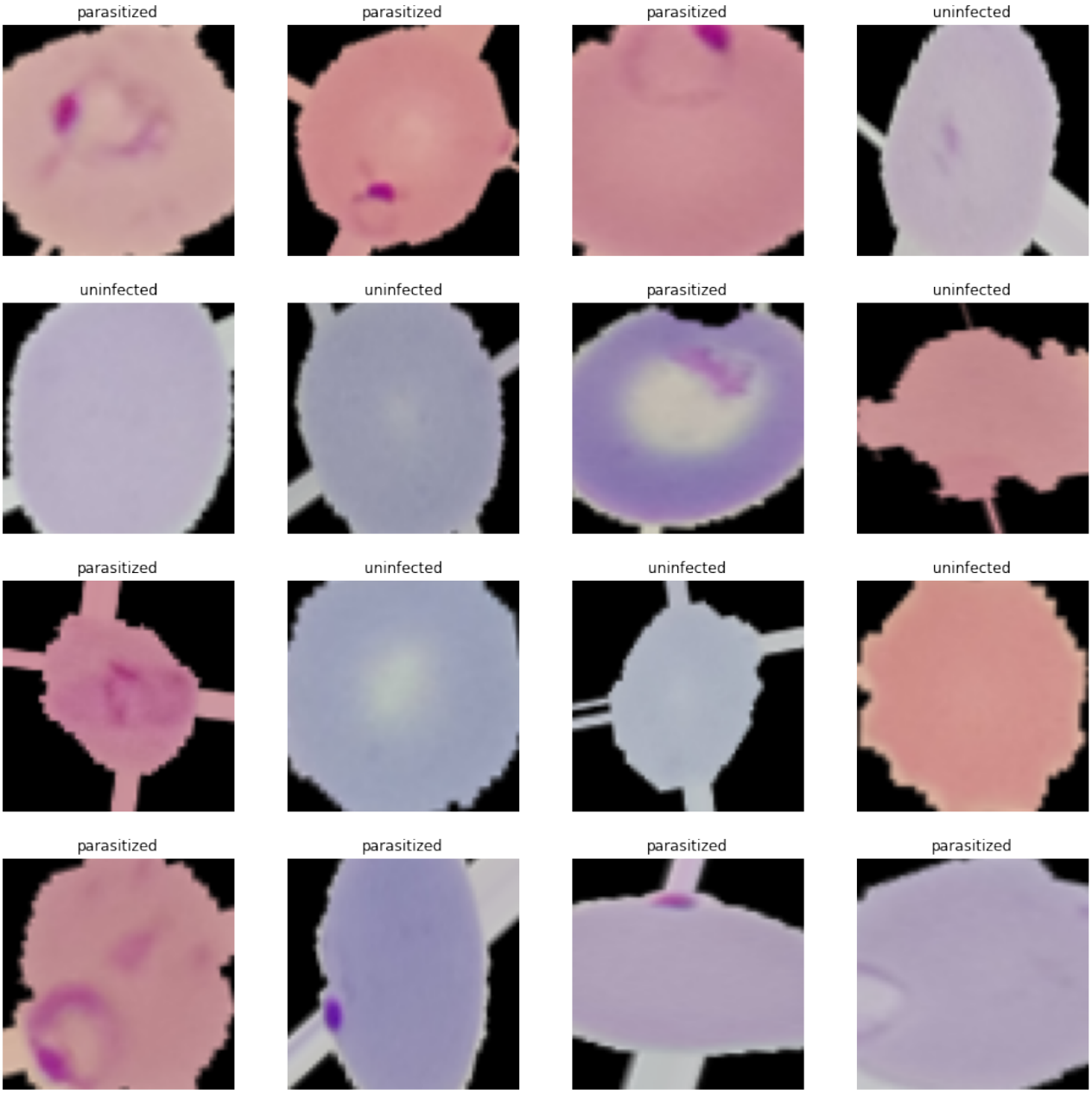
In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(train_images, train_label,

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Using ImageDataGenerator to generate images
train_datagen = ImageDataGenerator(horizontal_flip = True,
                                   zoom_range = 0.5, rotation_range = 30)
val_datagen  = ImageDataGenerator()

# Flowing training images using train_datagen generator
train_generator = train_datagen.flow(x = X_train, y = y_train, batch_size =

# Flowing validation images using val_datagen generator
val_generator =  val_datagen.flow(x = X_val, y = y_val, batch_size = 64, see
```

Observations :

The model assumes the parasite is static in size and shape without variability. This is not true of the biology of the parasite which changes shape and size throughout the life cycle. This model should be capable of selecting the parasite from a blood smear at any stage of the life cycle. This would require training the samples against a variety of possibilities in shape, size, and rotation of the images. Can the parasite be detected as well if the images are zoomed in, blurring focus? Can the parasite be detected if the images are rotated? The ImageDataGenerator allows alteration of the images as a batch method to determine if the changes affect the detection, accuracy, and precision rates.

Visualizing Augmented images

In [ ]:
```
# Creating an iterable for images and labels from the training data
images, labels = next(train_generator)

# Plotting 16 images from the training data
fig, axes = plt.subplots(4, 4, figsize = (16, 8))

fig.set_size_inches(16, 16)
for (image, label, ax) in zip(images, labels, axes.flatten()):
    ax.imshow(image)
    if label[1] == 1:
        ax.set_title('parasitized')
    else:
        ax.set_title('uninfected')
    ax.axis('off')
```



Observations

In [ ]:

```
#Building the Model
def cnn_model_3():
  model = Sequential()
  #First Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same",
                   activation = "leaky_relu",
                   input_shape = (64, 64, 3)))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Second Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Third Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Fourth Convolutional Layer
  model.add(Conv2D(filters = 64, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))

  #Add a BatchNormalization Layer
  model.add(tf.keras.layers.BatchNormalization())
  #Flatten previous layer
  model.add(Flatten())
  #Add Density Layer
  model.add(Dense(512, activation = "leaky_relu"))
  model.add(Dropout(0.4))

  #Output Layer
  model.add(Dense(2, activation = "softmax")) # 2 represents output layer ne

  #Compile
  adam = optimizers.Adam(learning_rate = 0.001)
  model.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])
  return model

#Build the model
model_3=cnn_model_2()

#Print Model Summary
model_3.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        896

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout_2 (Dropout)         (None, 8, 8, 32)          0

 conv2d_3 (Conv2D)           (None, 8, 8, 64)          18496

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_3 (Dropout)         (None, 4, 4, 64)          0

 batch_normalization (BatchN  (None, 4, 4, 64)         256
 ormalization)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 512)               524800

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 563,970
Trainable params: 563,842
Non-trainable params: 128
_________________________________________________________________
```

In [ ]:
```
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('.mdl_wts.hdf5', monitor='val_loss', save_best_only=True
```

In [ ]:
```
#Fit and Train Model
history_3 = model_3.fit(train_generator,
                        validation_data = val_generator,
                        batch_size = 128,
                        callbacks = callbacks,
                        validation_split = 0.2,
                        epochs = 20,
                        verbose = 1)
```

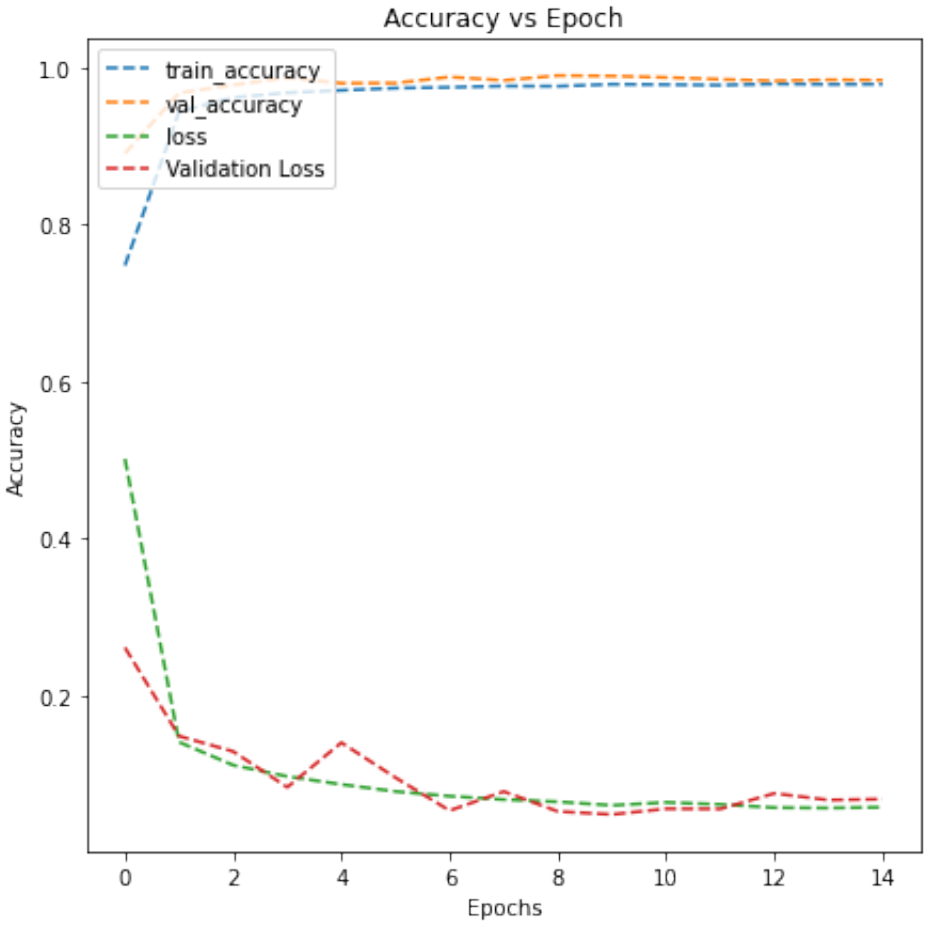
```
Epoch 1/20
312/312 [==============================] - 173s 551ms/step - loss: 0.5545 -
accuracy: 0.7126 - val_loss: 0.5401 - val_accuracy: 0.5545
Epoch 2/20
312/312 [==============================] - 172s 551ms/step - loss: 0.1966 -
accuracy: 0.9307 - val_loss: 0.0886 - val_accuracy: 0.9742
Epoch 3/20
312/312 [==============================] - 167s 534ms/step - loss: 0.1746 -
accuracy: 0.9399 - val_loss: 0.0890 - val_accuracy: 0.9792
Epoch 4/20
312/312 [==============================] - 162s 520ms/step - loss: 0.1713 -
accuracy: 0.9443 - val_loss: 0.0823 - val_accuracy: 0.9784
Epoch 5/20
312/312 [==============================] - 173s 554ms/step - loss: 0.1629 -
accuracy: 0.9456 - val_loss: 0.0812 - val_accuracy: 0.9754
Epoch 6/20
312/312 [==============================] - 171s 548ms/step - loss: 0.1623 -
accuracy: 0.9469 - val_loss: 0.0789 - val_accuracy: 0.9774
Epoch 7/20
312/312 [==============================] - 171s 547ms/step - loss: 0.1614 -
accuracy: 0.9471 - val_loss: 0.0809 - val_accuracy: 0.9786
Epoch 8/20
312/312 [==============================] - 166s 533ms/step - loss: 0.1596 -
accuracy: 0.9461 - val_loss: 0.0811 - val_accuracy: 0.9796
Epoch 9/20
312/312 [==============================] - 164s 523ms/step - loss: 0.1552 -
accuracy: 0.9508 - val_loss: 0.0871 - val_accuracy: 0.9754
Epoch 10/20
312/312 [==============================] - 167s 534ms/step - loss: 0.1458 -
accuracy: 0.9522 - val_loss: 0.0758 - val_accuracy: 0.9782
Epoch 11/20
312/312 [==============================] - 172s 552ms/step - loss: 0.1518 -
accuracy: 0.9504 - val_loss: 0.0824 - val_accuracy: 0.9774
Epoch 12/20
312/312 [==============================] - 167s 536ms/step - loss: 0.1479 -
accuracy: 0.9524 - val_loss: 0.0715 - val_accuracy: 0.9808
Epoch 13/20
312/312 [==============================] - 163s 521ms/step - loss: 0.1453 -
accuracy: 0.9519 - val_loss: 0.0730 - val_accuracy: 0.9772
Epoch 14/20
312/312 [==============================] - 165s 529ms/step - loss: 0.1503 -
accuracy: 0.9513 - val_loss: 0.0764 - val_accuracy: 0.9776
Epoch 15/20
312/312 [==============================] - 170s 545ms/step - loss: 0.1489 -
accuracy: 0.9512 - val_loss: 0.0705 - val_accuracy: 0.9802
Epoch 16/20
312/312 [==============================] - 169s 542ms/step - loss: 0.1456 -
```

```
accuracy: 0.9519 - val_loss: 0.0782 - val_accuracy: 0.9774
Epoch 17/20
312/312 [==============================] - 164s 527ms/step - loss: 0.1384 -
accuracy: 0.9557 - val_loss: 0.0753 - val_accuracy: 0.9788
Epoch 18/20
312/312 [==============================] - 161s 516ms/step - loss: 0.1434 -
accuracy: 0.9532 - val_loss: 0.0722 - val_accuracy: 0.9788
Epoch 19/20
312/312 [==============================] - 166s 533ms/step - loss: 0.1383 -
accuracy: 0.9545 - val_loss: 0.0743 - val_accuracy: 0.9792
Epoch 20/20
312/312 [==============================] - 169s 543ms/step - loss: 0.1408 -
accuracy: 0.9557 - val_loss: 0.0715 - val_accuracy: 0.9790
Epoch 20: early stopping
```

Evaluate the model

In [ ]:

```
#Function to plot train and validation accuracy
def plot_accuracy(history_3):
    N = len(history.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history.history["accuracy"],
             label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_accuracy"],
             label = "val_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["loss"],
             label = "loss", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_loss"],
             label = "Validation Loss", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")

plot_accuracy(history_3)
```

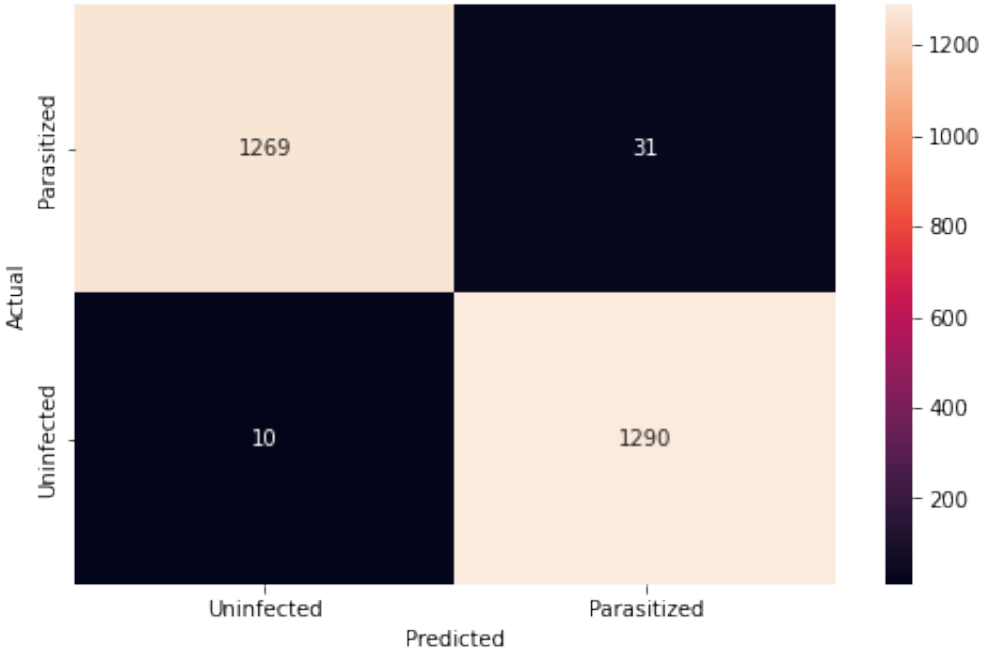


In [ ]:
```
# Evaluate the model to calculate the accuracy
accuracy_3 = model_3.evaluate(test_images, test_label, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

```
82/82 [==============================] - 5s 65ms/step - loss: 0.0632 - accur
acy: 0.9842

 Test_Accuracy:- 0.9823076725006104
```

In [ ]:
```
#Generate the classification report and confusion matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

pred = model_3.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_label, axis = 1)

# Printing the classification report
print(classification_report(y_true, pred))

# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f',
            xticklabels = ['Uninfected', 'Parasitized'],
            yticklabels = ['Parasitized', 'Uninfected'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
82/82 [==============================] - 5s 64ms/step
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1300
           1       0.98      0.99      0.98      1300

    accuracy                           0.98      2600
   macro avg       0.98      0.98      0.98      2600
weighted avg       0.98      0.98      0.98      2600
```



Observations :

- The model does not have improved overall performance as compared to the previous models. The confusion matrix and the accuracy plot show accuracy across epochs with 98% precision, 99% recall, and 98% F1 score.
- The model predicted 21 false positives in this model. This is concerning due to the exposure to drug toxicity from treatment drugs.
- The previously designed Model 2 currently performs better than Model 3 but does indicate additional training with rotated, magnified, or other alterations to the image affect the model's ability to predict with the same precision.

Model 4: VGG16

A brief evaluation of pretrained model like vgg16 performance on our data as compared to the previously trained models.

In [ ]:

```
# Clearing backend
from tensorflow.keras import backend
backend.clear_session()

# Fixing the seed for random number generators
import numpy as np
np.random.seed(42)
import random
random.seed(42)
tf.random.set_seed(42)

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.layers import Dense, Dropout, Flatten
from pathlib import Path


from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(train_images, train_label,

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Using ImageDataGenerator to generate images
train_datagen = ImageDataGenerator(horizontal_flip = True,
                                   zoom_range = 0.5, rotation_range = 30)
val_datagen  = ImageDataGenerator()

# Flowing training images using train_datagen generator
train_generator = train_datagen.flow(x = X_train, y = y_train, batch_size =

# Flowing validation images using val_datagen generator
val_generator =  val_datagen.flow(x = X_val, y = y_val, batch_size = 64, see

vgg = VGG16(include_top = False,
            weights = 'imagenet',
            input_shape = (64, 64, 3),
            classifier_activation=None)
vgg.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 64, 3)]       0

 block1_conv1 (Conv2D)       (None, 64, 64, 64)        1792

 block1_conv2 (Conv2D)       (None, 64, 64, 64)        36928

 block1_pool (MaxPooling2D)  (None, 32, 32, 64)        0

 block2_conv1 (Conv2D)       (None, 32, 32, 128)       73856

 block2_conv2 (Conv2D)       (None, 32, 32, 128)       147584

 block2_pool (MaxPooling2D)  (None, 16, 16, 128)       0

 block3_conv1 (Conv2D)       (None, 16, 16, 256)       295168

 block3_conv2 (Conv2D)       (None, 16, 16, 256)       590080

 block3_conv3 (Conv2D)       (None, 16, 16, 256)       590080

 block3_pool (MaxPooling2D)  (None, 8, 8, 256)         0

 block4_conv1 (Conv2D)       (None, 8, 8, 512)         1180160

 block4_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 block5_conv1 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 2, 2, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
#Building the Model
def vgg16_model_4():
  transfer_layer = vgg.get_layer('block5_pool')
  vgg.trainable = False
  # Add classification layers on top of it
  x = Flatten()(transfer_layer.output)  # Flatten the output from the 3rd bl
  x = Dense(256, activation = 'relu')(x)

  # Similarly add a dense layer with 128 neurons
  x = Dropout(0.3)(x)
  x = Dense(128, activation = 'relu')(x)

  # Add a dense layer with 64 neurons
  x = BatchNormalization()(x)
  pred = Dense(64, activation = 'softmax')(x)

  # Initializing the model
  model_4 = Model(vgg.input, pred)

  #Compile
  adam = optimizers.Adam(learning_rate = 0.001)
  model.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])
  return model

#Build the model
model_4=vgg16_model_4()

#Print Model Summary
model_4.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 32)        416

 max_pooling2d (MaxPooling2D  (None, 32, 32, 32)       0
 )

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        4128

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        4128

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout_2 (Dropout)         (None, 8, 8, 32)          0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 1,058,786
Trainable params: 1,058,786
Non-trainable params: 0
_________________________________________________________________
```

Compiling the model

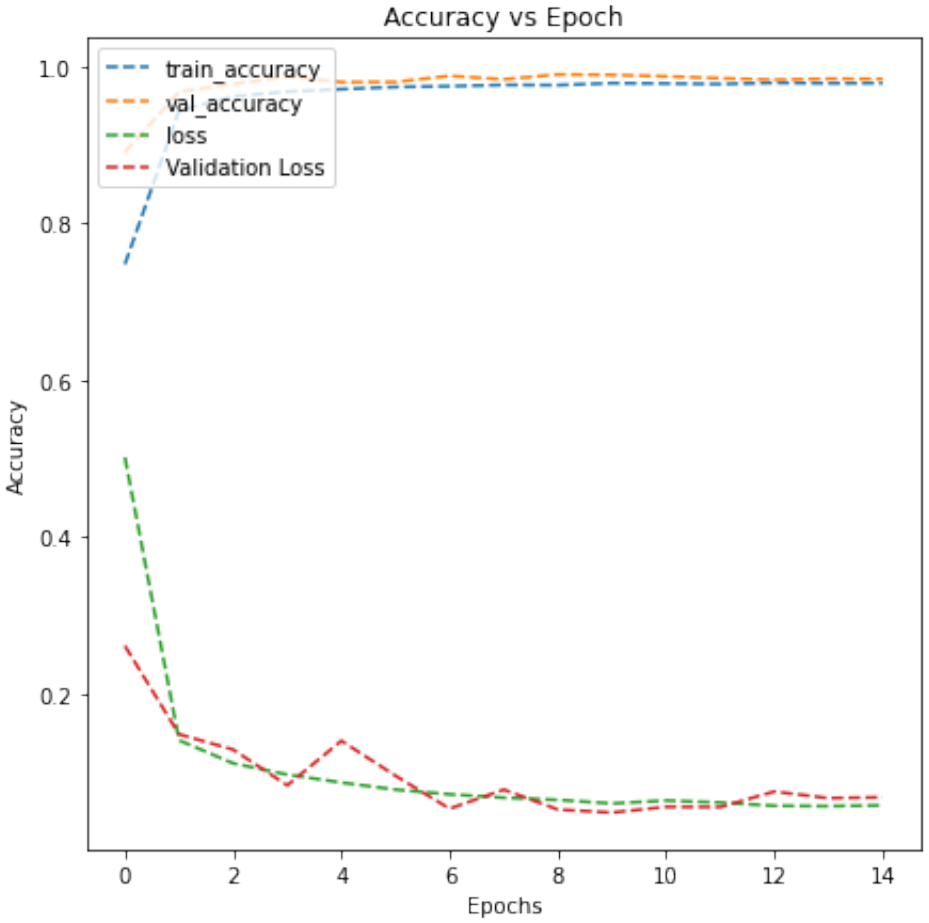
In [ ]:
```
#Compiling the Model
adam = optimizers.Adam(learning_rate = 0.001)
model_4.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])

# Adding Callbacks to the model
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1),
    ModelCheckpoint('vgg16_1.h5', monitor='val_loss', save_best_only=True)]

#Fit and Train the Model and running the model for 10 epochs
history_4 = model_4.fit(train_generator,
                        validation_data = val_generator,
                        batch_size = 128,
                        callbacks = callbacks,
                        validation_split = 0.2,
                        epochs = 20,
                        verbose = 1)
```

```
Epoch 1/20
312/312 [==============================] - 115s 366ms/step - loss: 0.1609 -
accuracy: 0.9461 - val_loss: 0.0620 - val_accuracy: 0.9834
Epoch 2/20
312/312 [==============================] - 114s 364ms/step - loss: 0.1502 -
accuracy: 0.9483 - val_loss: 0.0618 - val_accuracy: 0.9818
Epoch 3/20
312/312 [==============================] - 115s 367ms/step - loss: 0.1462 -
accuracy: 0.9505 - val_loss: 0.0625 - val_accuracy: 0.9808
Epoch 4/20
312/312 [==============================] - 115s 368ms/step - loss: 0.1430 -
accuracy: 0.9515 - val_loss: 0.0649 - val_accuracy: 0.9802
Epoch 5/20
312/312 [==============================] - 114s 366ms/step - loss: 0.1380 -
accuracy: 0.9533 - val_loss: 0.0622 - val_accuracy: 0.9816
Epoch 6/20
312/312 [==============================] - 115s 367ms/step - loss: 0.1482 -
accuracy: 0.9496 - val_loss: 0.0628 - val_accuracy: 0.9802
Epoch 7/20
312/312 [==============================] - 115s 369ms/step - loss: 0.1449 -
accuracy: 0.9507 - val_loss: 0.0654 - val_accuracy: 0.9802
Epoch 7: early stopping
```

In [ ]:
```
# Evaluate the model to calculate the accuracy
accuracy_4 = model_4.evaluate(test_images, test_label, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

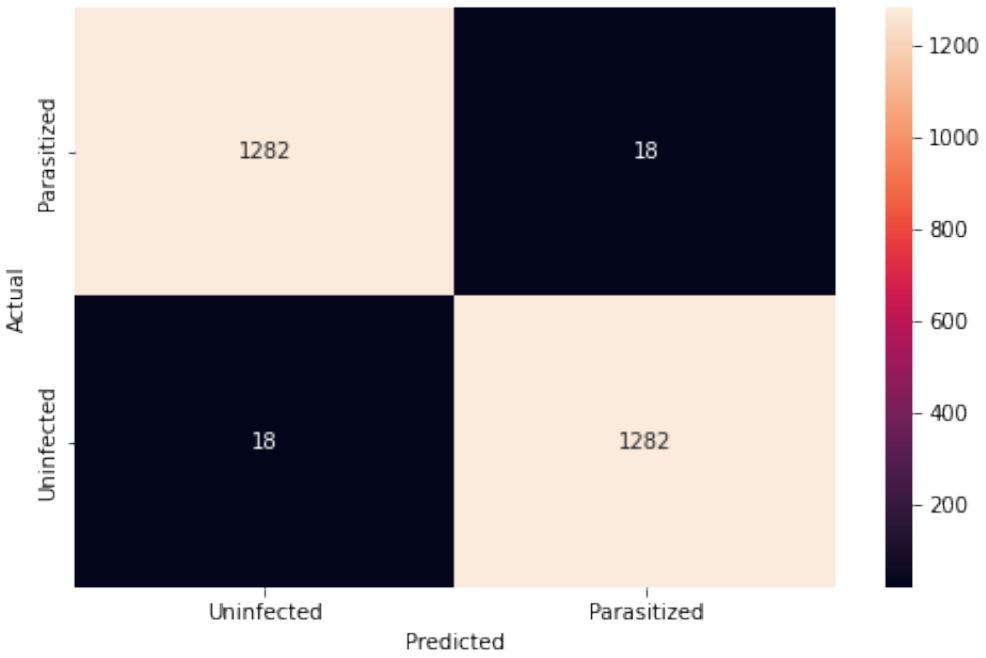
```
82/82 [==============================] - 3s 41ms/step - loss: 0.0473 - accur
acy: 0.9862

 Test_Accuracy:- 0.9823076725006104
```

In [ ]:
```
#Function to plot train and validation accuracy
def plot_accuracy(history_4):
    N = len(history.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history.history["accuracy"],
             label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_accuracy"],
             label = "val_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["loss"],
             label = "loss", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_loss"],
             label = "Validation Loss", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")

plot_accuracy(history_4)
```



In [ ]:
```
# Plot the confusion matrix and generate a classification report for the moc
#Generate the classification report and confusion matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

pred = model_4.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_label, axis = 1)

# Printing the classification report
print(classification_report(y_true, pred))

# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f',
            xticklabels = ['Uninfected', 'Parasitized'],
            yticklabels = ['Parasitized', 'Uninfected'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
82/82 [==============================] - 3s 41ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1300
           1       0.99      0.99      0.99      1300

    accuracy                           0.99      2600
   macro avg       0.99      0.99      0.99      2600
weighted avg       0.99      0.99      0.99      2600
```



Observations and insights:

- Test accuracy for the vgg16 model is 98.23%, the same as the previous model performance. The predictive nature of this model shows significantly better recall and precision as compared to previous models.
- The validation and test accuracy match suggesting good training with minimal overfitting.
- The confusion matrix shows the model accurately predicted 1,282 infected cells which matches the actual exactly. The accuracy, recall, and F1 score for the VGG16 model is 99%.
- This model has since been used to identify human thyroid cancer cells in digital images, breast cancer tumors in digital mammography, and many more medical applications; making it a good fit for the blood smear slides.

Future Consideration

- Addition of a recommendation system for directed point-of-care testing for positive samples. Integrate textbot to send reminder messages for appointments to follow up.
- Web-based submission for localities without subject matter expertise as a telehealth lab. Add precision to the model by adding digital PCR data when both are available.
- Abnormal liver function test (LFT) could be used in tandem to add weighted nodes to the algorithm and improve precision. Future considerations to focus efforts to control the mosquito vectors by utilizing a comparison of breeding temperatures for mosquitoes and predict when to implement seasonal outreach to the community. How is this affected by climate change? Earlier season, hotter temperatures, longer season, seasonal rains leading to standing water. The geographical distribution of *Plasmodium spp.* requires knowledge of differences in key features.
- Methods to reduce the endemic number of infections are currently limited to vector control. Controlling the mosquito population by interrupting the Anopheles life cycle reduces the number of infected cases. The Global Technical Strategy for Malaria 2016-2030 report by the World Health Organization lists a goal to reduce global Malaria mortality rates from the 2015 baseline.

Final Key Observations and Insights

Final Model, Fit, Evaluation

This final model seeks to test the previous HSV and Gaussian Blur augmented images in Model 2 to improve performance as compared to the VGG16 model.

Model 5: Hue Saturation Augmented Images Convolutional Neural Network

This section will create one final model using the Hue saturation Gaussian Blur augmented images.

In [ ]:
```
#Load Libraries, Unzip Images, Add only HSV Preprocessing Steps
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [ ]:
```
#Data Handling library
import numpy as np
#from numpy.core.fromnumeric import size
import pandas as pd

#Graphing library
import matplotlib.pyplot as plt
%matplotlib inline

#Data Visualization library
import seaborn as sns
sns.set_color_codes(palette='dark')
sns.dark_palette=('seagreen')

#Machine Learning library
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

#Suppress Python Library depreciation warnings
import warnings
warnings.filterwarnings('ignore')

#Set maximum size of pandas DataFrame
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)
```

In [ ]:
```
#The dataset is named 'cell_images.zip' and is located in the Colab Notebook
#The data is provided as a zip file that requires unzipping.  Load zipfile t
import zipfile
import os

#File location to be read
path='/content/drive/MyDrive/Colab Notebooks/Capstone Project/cell_images.zi

#Extract data files from zip file
with zipfile.ZipFile(path, 'r') as zip_ref:
    zip_ref.extractall()
```

In [ ]:
```
#Import Pillow Library as PIL
from PIL import Image
```

In [ ]:

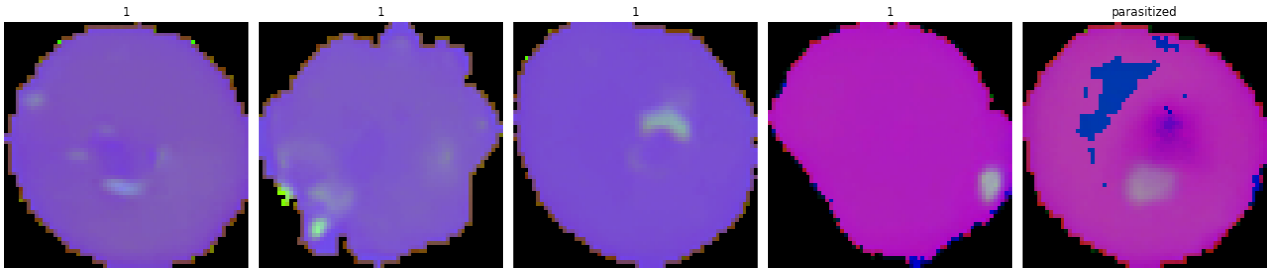
```
#Assigning train data for infected and uninfected
train_dir='/content/cell_images/train/'

#Designate image size for normality across data set.
SIZE=64

#Create list containers for the image data
train_images=[]

#Create a list container for Boolean value indicating infection state (0:uni
train_label=[]

#Loading Training data into List
for folder_name in ['/parasitized/','/uninfected/']:
  images_path=os.listdir(train_dir + folder_name)
  for i, image_name in enumerate(images_path):
    try:
      #Open each image in sequence as exception coding
      image=Image.open(train_dir + folder_name + image_name)
      #Resize Images to 64x64 to standardize to same shape
      image=image.resize((SIZE,SIZE))
      #Convert the images to list 'train_images' as designated above
      train_images.append(np.array(image))
      #Create the infected and uninfected labels with conditional logic
      if folder_name=='/parasitized/':
        train_label.append(1)
      else:
        train_label.append(0)
    except Exception:
      pass

#Convert list above into NumPy Array
train_images=np.array(train_images)
train_label=np.array(train_label)
```

In [ ]:
```
#Assigning test data for infected and uninfected
test_dir='/content/cell_images/test/'

#Designate image size for normality across data set.
SIZE=64

#Create list containers for the image data
test_images=[]

#Create a list container for Boolean value indicating infection state (0:uni
test_label=[]

#Loading Test data into List
for folder_name in ['/parasitized/','/uninfected/']:
  images_path=os.listdir(test_dir + folder_name)
  for i, image_name in enumerate(images_path):
    try:
      #Open each image in sequence as exception coding
      image=Image.open(test_dir + folder_name + image_name)
      #Resize Images to 64x64 to standardize to same shape
      image=image.resize((SIZE,SIZE))
      #Convert the images to list 'train_images' as designated above
      test_images.append(np.array(image))
      #Create the infected and uninfected labels with conditional logic
      if folder_name=='/parasitized/':
        test_label.append(1)
      else:
        test_label.append(0)
    except Exception:
      pass

#Convert list above into NumPy Array
test_images=np.array(test_images)
test_label=np.array(test_label)
```

In [ ]:
```
#Normalize the data.  There are 255 pixels thus all data will be normalized
norm_train_images = (train_images/255).astype('float32')
norm_test_images = (test_images/255).astype('float32')
```

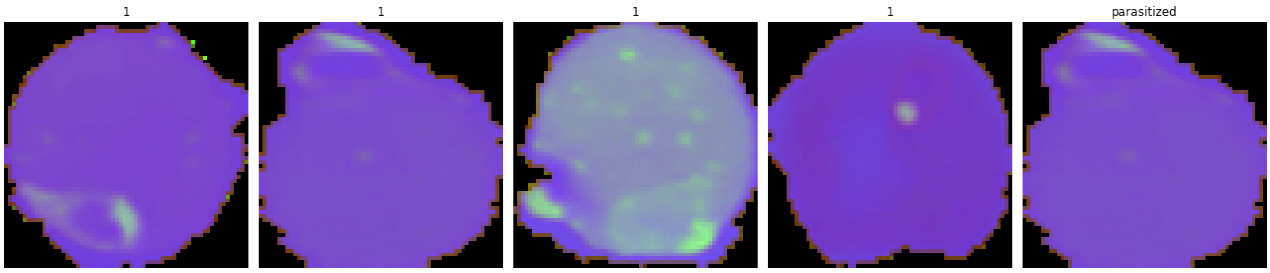
In [ ]:

```
#Import ComputerVision image processing library
import cv2

#Create a List Container for Hue Saturation Value Array
gfx=[]

#Convert RBG data to HSV list
for i in np.arange(0,100,1):
  a=cv2.cvtColor(train_images[i], cv2.COLOR_BGR2HSV)
  gfx.append(a)
#Create HSV NumPy Array
gfx=np.array(gfx)

#Apply to and convert training data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=train_label[i]
  ax[t].set_title(Title)
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
```

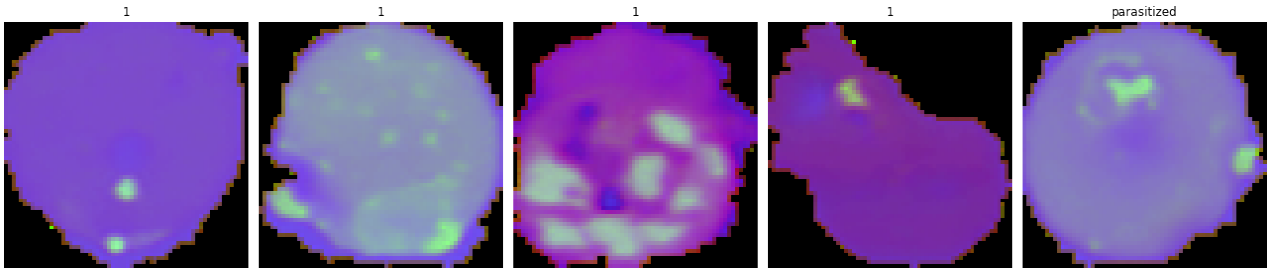


In [ ]:

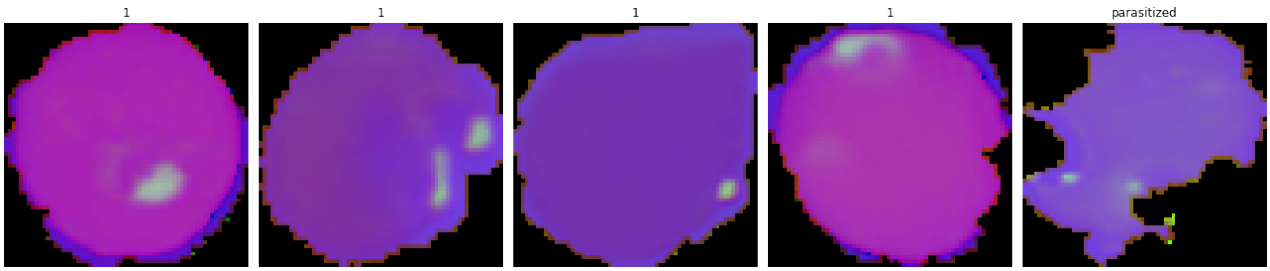
```
#Import ComputerVision image processing library
import cv2

#Create a List Container for Hue Saturation Value Array
gfx=[]

#Convert RBG data to HSV list
for i in np.arange(0,100,1):
  a=cv2.cvtColor(test_images[i], cv2.COLOR_BGR2HSV)
  gfx.append(a)
#Create HSV NumPy Array
gfx=np.array(gfx)

#Apply to and convert training data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=train_label[i]
  ax[t].set_title(Title)
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
```



In [ ]:
```
#Create container list to hold blurred image data
gbx=[]
#Convert image data to Gaussian Blur
for i in np.arange(0,100,1):
  b=cv2.GaussianBlur(train_images[i], (5,5),0)
  gbx.append(b)
#Pass gbx list to NumPy Array
gbx=np.array(gbx)

#Apply to and convert training data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=train_label[i]
  ax[t].set_title(Title)
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
```



In [ ]:
```
#Create container list to hold blurred image data
gbx=[]
#Convert image data to Gaussian Blur
for i in np.arange(0,100,1):
  b=cv2.GaussianBlur(train_images[i], (5,5),0)
  gbx.append(b)
#Pass gbx list to NumPy Array
gbx=np.array(gbx)

#Apply to and convert test data
viewimage=np.random.randint(1,100,5)
fig,ax=plt.subplots(1,5,figsize=(18,18))
for t, i in zip(range(5),viewimage):
  Title=test_label[i]
  ax[t].set_title(Title)
  ax[t].imshow(gfx[i])
  ax[t].set_axis_off()
  fig.tight_layout()
  if Title==1:
    plt.title('parasitized')
  else:
    plt.title('uninfected')
```



In [ ]:
```
#Load libraries, create model compiling Model 2 and VGG16, Add Metrics
```

In [ ]:
```
#Model Building library
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Activation, BatchNormali
from tensorflow.keras.losses import categorical_crossentropy, binary_crossen
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.utils import to_categorical

# Fixing the seed for random number generators
import numpy as np
np.random.seed(42)
import random
random.seed(42)
tf.random.set_seed(42)

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.layers import Dense, Dropout, Flatten
from pathlib import Path

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(train_images, train_label,

from tensorflow.keras.preprocessing.image import ImageDataGenerator

print(tf.__version__)
```

```
2.9.2
```

In [ ]:
```
# Encoding Train Labels
train_label = to_categorical(train_label, 2)

# Similarly let us try to encode test labels
test_label = to_categorical(test_label, 2)

#Visualize the new vector array
print(train_label)
```

```
[[0. 1.]
 [0. 1.]
 [0. 1.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

In [ ]:

```
#Model 5
# Clearing backend for a new session
from tensorflow.keras import backend
backend.clear_session()
```

In [ ]:

```
#Build Model 5 Architecture
def cnn_model_5():
  model = Sequential()
  #First Convolutional Layer
  model.add(Conv2D(filters = 32, kernel_size = (3,3),
                   padding = "same",
                   activation = "leaky_relu",
                   input_shape = (64, 64, 3)))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Second Convolutional Layer
  model.add(Conv2D(filters = 64, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Third Convolutional Layer
  model.add(Conv2D(filters = 64, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Fourth Convolutionl Layer
  model.add(Conv2D(filters = 128, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))
  #Fifth Convolutionl Layer
  model.add(Conv2D(filters = 256, kernel_size = (3,3),
                   padding = "same", activation = "leaky_relu"))
  model.add(MaxPooling2D(pool_size = 2))
  model.add(Dropout(0.2))

  #Add a BatchNormalization Layer
  model.add(tf.keras.layers.BatchNormalization())
  #Flatten previous layer
  model.add(Flatten())
  #Add Density Layer
  model.add(Dense(512, activation = "leaky_relu"))
  model.add(Dropout(0.2))

  #Output Layer
  model.add(Dense(2, activation = "softmax")) # 2 represents output layer ne

  #Compile
  model.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])
  return model

#Build the model
model_5=cnn_model_5()

#Print Model Summary
model_5.summary()
```

Model: "sequential"

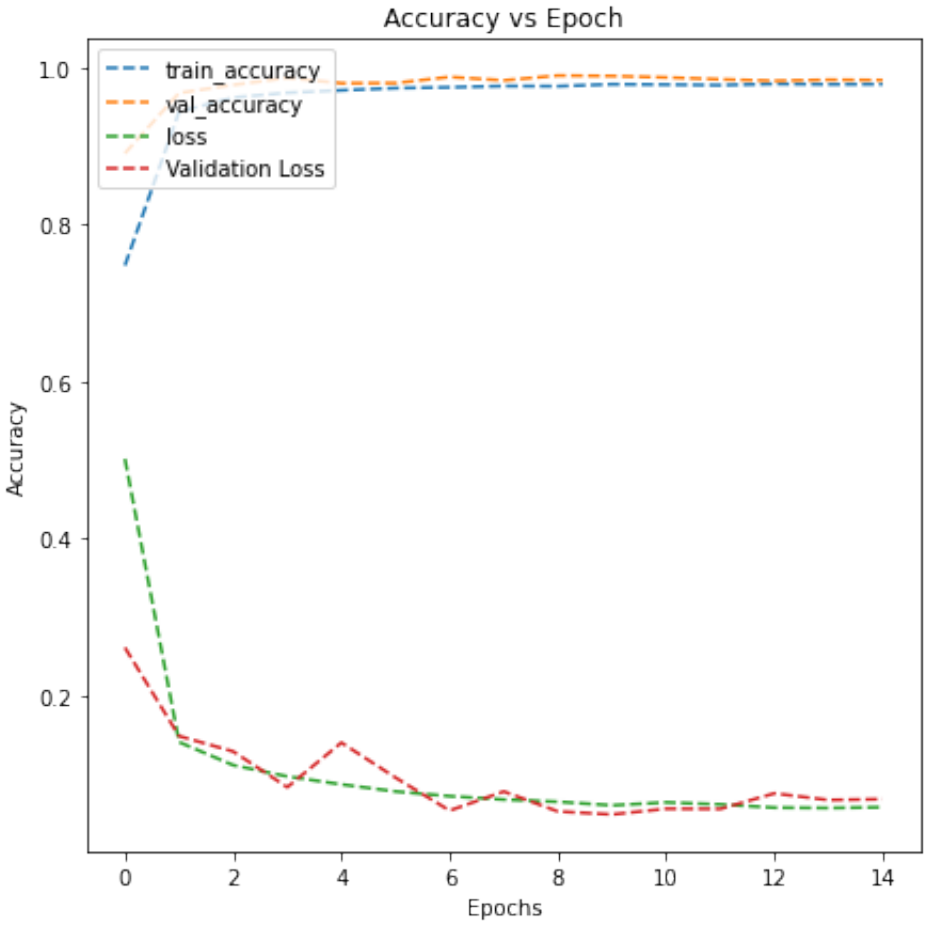
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
conv2d_3 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_4 (Dropout)	(None, 2, 2, 256)	0
batch_normalization (BatchNormalization)	(None, 2, 2, 256)	1024
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dropout_5 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1026

Total params: 952,194
Trainable params: 951,682
Non-trainable params: 512

In [ ]:
```
#Compiling the Model
adam = optimizers.Adam(learning_rate = 0.001)
model_5.compile(
      loss = 'binary_crossentropy',
      optimizer = 'adam',
      metrics = ['accuracy'])

# Adding Callbacks to the model
callbacks = [EarlyStopping(monitor='val_loss', patience=2, verbose=1),
    ModelCheckpoint('.mdl_wts.hdf5', monitor='val_loss', save_best_only=True

#Fit and Train the Model and running the model for 10 epochs
history_5 = model_5.fit(train_images,
                    train_label,
                    batch_size = 256,
                    callbacks = callbacks,
                    validation_split = 0.2,
                    epochs = 10,
                    verbose = 1)
```

```
Epoch 1/10
78/78 [==============================] - 238s 3s/step - loss: 0.6722 - accur
acy: 0.6647 - val_loss: 0.4063 - val_accuracy: 0.8736
Epoch 2/10
78/78 [==============================] - 231s 3s/step - loss: 0.1720 - accur
acy: 0.9347 - val_loss: 0.0327 - val_accuracy: 0.9948
Epoch 3/10
78/78 [==============================] - 234s 3s/step - loss: 0.0802 - accur
acy: 0.9717 - val_loss: 0.0500 - val_accuracy: 0.9892
Epoch 4/10
78/78 [==============================] - 228s 3s/step - loss: 0.0719 - accur
acy: 0.9759 - val_loss: 0.0404 - val_accuracy: 0.9906
Epoch 4: early stopping
```

In [ ]:
```
#Function to plot train and validation accuracy
def plot_accuracy(history_5):
    N = len(history.history["accuracy"])
    plt.figure(figsize = (7, 7))
    plt.plot(np.arange(0, N), history.history["accuracy"],
             label = "train_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_accuracy"],
             label = "val_accuracy", ls = '--')
    plt.plot(np.arange(0, N), history.history["loss"],
             label = "loss", ls = '--')
    plt.plot(np.arange(0, N), history.history["val_loss"],
             label = "Validation Loss", ls = '--')
    plt.title("Accuracy vs Epoch")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc="upper left")

plot_accuracy(history_5)
```

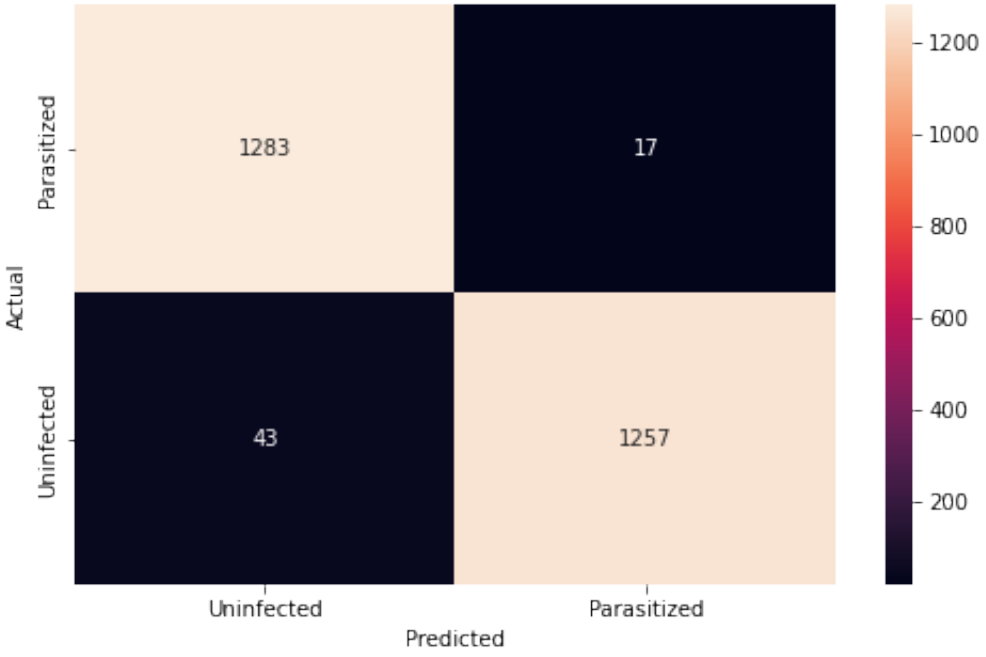


In [ ]:
```
# Evaluate the model to calculate the accuracy
accuracy_5=model_5.evaluate(test_images, test_label, verbose = 1)
print('\n', 'Test_Accuracy:-', accuracy[1])
```

```
82/82 [==============================] - 8s 97ms/step - loss: 0.0744 - accur
acy: 0.9769

 Test_Accuracy:- 0.9823076725006104
```

In [ ]:
```
#Generate the classification report and confusion matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

pred = model_5.predict(test_images)
pred = np.argmax(pred, axis = 1)
y_true = np.argmax(test_label, axis = 1)

# Printing the classification report
print(classification_report(y_true, pred))

# Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_true, pred)
plt.figure(figsize = (8, 5))
sns.heatmap(cm, annot = True,  fmt = '.0f',
            xticklabels = ['Uninfected', 'Parasitized'],
            yticklabels = ['Parasitized', 'Uninfected'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
82/82 [==============================] - 8s 98ms/step
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1300
           1       0.99      0.97      0.98      1300

    accuracy                           0.98      2600
   macro avg       0.98      0.98      0.98      2600
weighted avg       0.98      0.98      0.98      2600
```



Observations and Key Insights

Observations and Conclusions drawn from the final model:

- The base model forms a good foundation for the initial model demonstrating high recall, precision, and F1 score. This starting model gave a high degree of confidence for the sensitivity of the model and specificity to the parasite identification. Model one added a convolutional network, pooling, and dropout layers and revealed similar accuracy percentages but yielded eight false negatives. Adding batch normalization and leakyReLU to Model two where the precision, accuracy, recall and F1 score were 99% and the number of false negatives were reduced to three samples. Model three showed a reduction in accuracy, precision, recall, and F1 score with false positive count at 21 samples. Of these Models performed Model 2 yielded the best performance.
- The VGG16 model is a pre-trained model that has become a standard for use in computer vision. The model has successfully been applied to a variety of images within the medical field including tumor detection from digital mammography and detection of abnormal human cells for cancer detection in thin tissue biopsies. The VGG16 model performed with 99% precision, accuracy, recall, and F1 score. This correctly identified all infected cells in the validation samples with zero false positives and zero false negatives. The VGG16 model performs superior to Model 2 with higher sensitivity and specificity.
- This suggests the parameters to analyze the images for key features appreciates with repeated training. Combining fine-tuned parameters and utilizing the CNN architecture to maximize key features for the images allows a robust model to be created that is capable of visualizing parasites in blood smears. This includes augmented images that are rotated, flipped, and magnified. This suggests the model can be successful with low image quality and images that may have previous processing.

Insights

- False negative samples could have serious health outcomes including mortality thus precision and accuracy must match or exceed the accuracy rate of traditional light microscopy.
- The VGG16 model demonstrated the importance of well selected layers each working to improve the ability to discern features in image data. Using the VGG16 model following the proposed Model 2 could improve overall precision, exceeding the precision of traditional light microscopy. The VGG16 model performs with similar accuracy to traditional techniques used to diagnose Malaria.

Works Cited

1. *Detection of Blood Parasites*. Clinical Microbiology Procedures Handbook, 3 Volume Set, 4th Edition Amy L. Leber (Editor-in-Chief). doi:10.1128/9781555818814.ch.9.8.1
2. *Giemsa Stain*. Clinical Microbiology Procedures Handbook, 3 Volume Set, 4th Edition Amy L. Leber (Editor-in-Chief). doi:10.1128/9781555818814.ch.9.8.1
3. Pillow Python Imaging Library. PIL Fork Documentation.
4. OpenCV. OpenCV: Smoothing Images API Documentation.
5. WHO Guidelines for malaria, 25 November 2022. Geneva: World Health Organization; 2022 (WHO/UCN/GMP/2022.01 Rev.3). License: CC BY-NC-SA 3.0 IGO.
6. Center for Disease Control and Prevention Division of Parasitic Diseases and Malaria (DPDM). DPDx - Laboratory Identification of Parasites of Public Health Concern
7. A. A. Alonso-Ramírez et al., *Classifying Parasitized and Uninfected Malaria Red Blood Cells Using Convolutional-Recurrent Neural Networks*. IEEE Access, vol. 10, pp. 97348-97359, 2022, doi: 10.1109/ACCESS.2022.3206266.
8. Jdey, I., Hcini, G., & Ltifi, H. (2022). Deep learning and machine learning for Malaria detection: overview, challenges and future directions. arXiv preprint arXiv:2209.13292.
9. *How to Configure Image Data Augmentation in Keras*. Deep Learning for Computer Vision. Brownlee J. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
10. *Understanding Data Augmentation | What is Data Augmentation & how it works?* Great Learning Blog AI and Machine Learning. Arun K. https://www.mygreatlearning.com/blog/understanding-data-augmentation/
11. *Everything you need to know about VGG16*. Medium.com Blog. Rohini G. September 23, 2021. Everything you need to know about VGG16.
12. *Step by step VGG16 implementation in Keras for beginners*. Towards Data Science Blog. Thakur, Rohit. August 6, 2019. Step by step VGG16 implementation in Keras for beginners.
13. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Simonyan, K. and Zisserman, A. https://arxiv.org/abs/1409.1556. September 4, 2014.

Additional Datasets of Interest

Peripheral Blood Smear Red and White Blood Cell Detection: https://user-images.githubusercontent.com/3878466/80433866-dd0f6380-8900-11ea-8018-ecd4e8a7afe8.png" alt="Blood Cell Detection Dataset"

Malaria Stage Classifier dataset: https://datasetsearch.research.google.com/search?src=2&query=Blood%20Cell%20Detection%20Dataset&docid=L2cvMTF0ZDk2eGs3eg%3D

Malaria Cell Images Dataset (Infected and uninfected) https://datasetsearch.research.google.com/search?src=2&query=Blood%20Cell%20Detection%20Dataset&docid=L2cvMTFqbnl2eHB2MA%3D

World Malaria Statistics by country: https://datasetsearch.research.google.com/search?src=2&query=malaria&docid=L2cvMTFzZ3ZtcXpibA%3D%3D