

```
In [185... import numpy as np
import pandas as pd
```

```
In [186... import os
print(os.listdir(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project'))
['cell images']
```

```
In [187... import cv2
import matplotlib.pyplot as plt
import seaborn as sns
import os
from PIL import Image
from tensorflow.keras.utils import img_to_array
from tensorflow.keras.utils import load_img
from keras.utils import np_utils
```

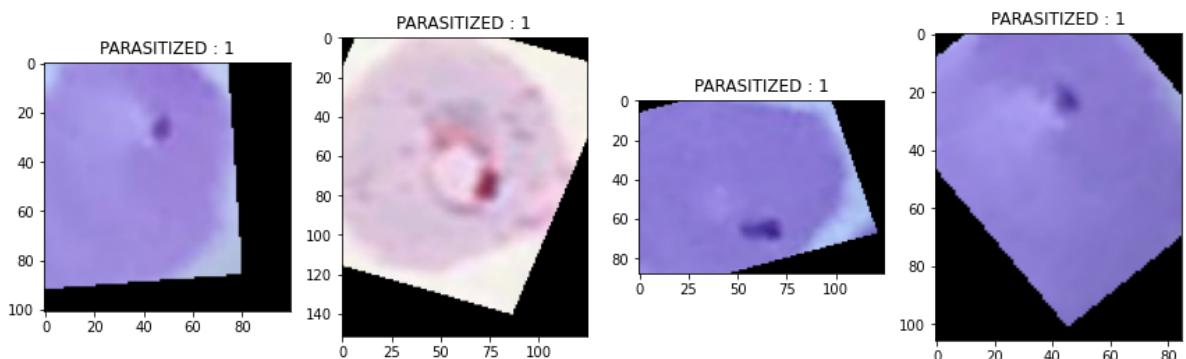
```
In [188... parasitized_data = os.listdir(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project')
print(parasitized_data[:10])

uninfected_data = os.listdir(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project\
print('\n')
print(uninfected_data[:10])

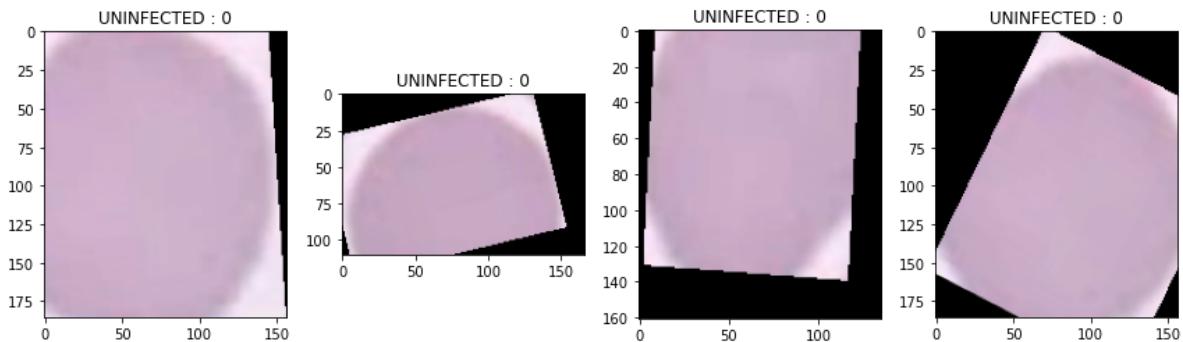
['aug_0_1.png', 'aug_0_1005.png', 'aug_0_1027.png', 'aug_0_1029.png', 'aug_0_104.p
ng', 'aug_0_1067.png', 'aug_0_1073.png', 'aug_0_1080.png', 'aug_0_1096.png', 'aug_
0_1105.png']

['aug_0_1000.png', 'aug_0_1004.png', 'aug_0_1038.png', 'aug_0_1039.png', 'aug_0_10
56.png', 'aug_0_1058.png', 'aug_0_1082.png', 'aug_0_1085.png', 'aug_0_1088.png',
'aug_0_1095.png']
```

```
In [189... plt.figure(figsize = (12,12))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project\cell ima
    plt.imshow(img)
    plt.title('PARASITIZED : 1')
    plt.tight_layout()
plt.show()
```

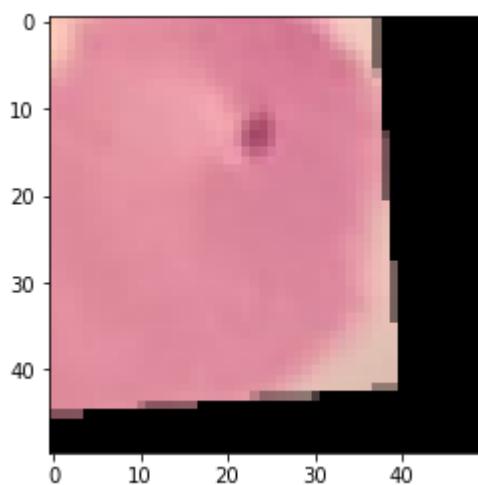


```
In [190... plt.figure(figsize = (12,12))
for i in range(4):
    plt.subplot(1, 4, i+1)
    img = cv2.imread(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project\cell ima
    plt.imshow(img)
    plt.title('UNINFECTED : 0')
    plt.tight_layout()
plt.show()
```



```
In [191...]: data = []
labels = []
for img in parasitized_data:
    try:
        img_read = plt.imread(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project')
        img_resize = cv2.resize(img_read, (50, 50))
        img_array = img_to_array(img_resize)
        data.append(img_array)
        labels.append(1)
    except:
        None

for img in uninfected_data:
    try:
        img_read = plt.imread(r'C:\Users\blueb\OneDrive\Desktop\Final_year_project')
        img_resize = cv2.resize(img_read, (50, 50))
        img_array = img_to_array(img_resize)
        data.append(img_array)
        labels.append(0)
    except:
        None
plt.imshow(data[0])
plt.show()
```



```
In [192...]: image_data = np.array(data)
labels = np.array(labels)
idx = np.arange(image_data.shape[0])
np.random.shuffle(idx)
image_data = image_data[idx]
labels = labels[idx]
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(image_data, labels, test_size =
y_train = np_utils.to_categorical(y_train, num_classes = 2)
y_test = np_utils.to_categorical(y_test, num_classes = 2)
print(f'SHAPE OF TRAINING IMAGE DATA : {x_train.shape}')
```

```
print(f'SHAPE OF TESTING IMAGE DATA : {x_test.shape}')
print(f'SHAPE OF TRAINING LABELS : {y_train.shape}')
print(f'SHAPE OF TESTING LABELS : {y_test.shape}')
```

```
SHAPE OF TRAINING IMAGE DATA : (1736, 50, 50, 3)
SHAPE OF TESTING IMAGE DATA : (307, 50, 50, 3)
SHAPE OF TRAINING LABELS : (1736, 2)
SHAPE OF TESTING LABELS : (307, 2)
```

In [193...]

```
import keras
from keras.layers import Dense, Conv2D
from keras.layers import Flatten
from keras.layers import MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Activation
from keras.layers import BatchNormalization
from keras.layers import Dropout
from keras.models import Sequential
from keras import backend as K

from keras import optimizers
```

In [194...]

```
def CNNbuild(height, width, classes, channels):
    model = Sequential()

    inputShape = (height, width, channels)
    chanDim = -1

    if K.image_data_format() == 'channels_first':
        inputShape = (channels, height, width)
    model.add(Conv2D(32, (3,3), activation = 'relu', input_shape = inputShape))
    model.add(MaxPooling2D(2,2))
    model.add(BatchNormalization(axis = chanDim))
    model.add(Dropout(0.2))

    model.add(Conv2D(32, (3,3), activation = 'relu'))
    model.add(MaxPooling2D(2,2))
    model.add(BatchNormalization(axis = chanDim))
    model.add(Dropout(0.2))

    model.add(Conv2D(32, (3,3), activation = 'relu'))
    model.add(MaxPooling2D(2,2))
    model.add(BatchNormalization(axis = chanDim))
    model.add(Dropout(0.2))

    model.add(Flatten())

    model.add(Dense(512, activation = 'relu'))
    model.add(BatchNormalization(axis = chanDim))
    model.add(Dropout(0.5))
    model.add(Dense(classes, activation = 'softmax'))

    return model

height = 50
width = 50
classes = 2
channels = 3
model = CNNbuild(height = height, width = width, classes = classes, channels = channels)
model.summary()
```

Model: "sequential_9"

| Layer (type) | Output Shape | Param # |
|---|--------------------|---------|
| <hr/> | | |
| conv2d_27 (Conv2D) | (None, 48, 48, 32) | 896 |
| max_pooling2d_27 (MaxPooling2D) | (None, 24, 24, 32) | 0 |
| batch_normalization_36 (BatchNormalization) | (None, 24, 24, 32) | 128 |
| dropout_36 (Dropout) | (None, 24, 24, 32) | 0 |
| conv2d_28 (Conv2D) | (None, 22, 22, 32) | 9248 |
| max_pooling2d_28 (MaxPooling2D) | (None, 11, 11, 32) | 0 |
| batch_normalization_37 (BatchNormalization) | (None, 11, 11, 32) | 128 |
| dropout_37 (Dropout) | (None, 11, 11, 32) | 0 |
| conv2d_29 (Conv2D) | (None, 9, 9, 32) | 9248 |
| max_pooling2d_29 (MaxPooling2D) | (None, 4, 4, 32) | 0 |
| batch_normalization_38 (BatchNormalization) | (None, 4, 4, 32) | 128 |
| dropout_38 (Dropout) | (None, 4, 4, 32) | 0 |
| flatten_9 (Flatten) | (None, 512) | 0 |
| dense_18 (Dense) | (None, 512) | 262656 |
| batch_normalization_39 (BatchNormalization) | (None, 512) | 2048 |
| dropout_39 (Dropout) | (None, 512) | 0 |
| dense_19 (Dense) | (None, 2) | 1026 |
| <hr/> | | |
| Total params: 285,506 | | |
| Trainable params: 284,290 | | |
| Non-trainable params: 1,216 | | |

In [195...]

```
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ['acc'])
h = model.fit(x_train, y_train, epochs = 8, batch_size = 16)
```

```

Epoch 1/8
109/109 [=====] - 4s 24ms/step - loss: 0.3605 - accuracy: 0.8940
Epoch 2/8
109/109 [=====] - 3s 26ms/step - loss: 0.1986 - accuracy: 0.9332
Epoch 3/8
109/109 [=====] - 3s 24ms/step - loss: 0.1313 - accuracy: 0.9603
Epoch 4/8
109/109 [=====] - 3s 27ms/step - loss: 0.0870 - accuracy: 0.9712
Epoch 5/8
109/109 [=====] - 3s 25ms/step - loss: 0.0759 - accuracy: 0.9729
Epoch 6/8
109/109 [=====] - 3s 24ms/step - loss: 0.0893 - accuracy: 0.9706
Epoch 7/8
109/109 [=====] - 3s 27ms/step - loss: 0.0523 - accuracy: 0.9821
Epoch 8/8
109/109 [=====] - 3s 27ms/step - loss: 0.0650 - accuracy: 0.9821

```

In [197]:

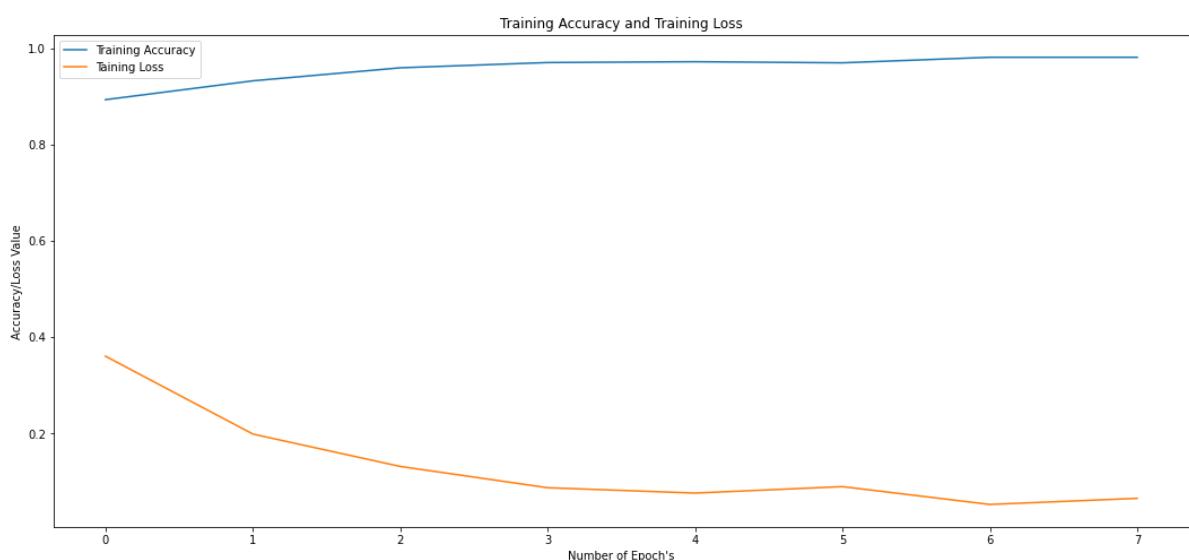
```

plt.figure(figsize = (18,8))
plt.plot(range(8), h.history['accuracy'], label = 'Training Accuracy')
plt.plot(range(8), h.history['loss'], label = 'Training Loss')

plt.xlabel("Number of Epoch's")
plt.ylabel('Accuracy/Loss Value')
plt.title('Training Accuracy and Training Loss')
plt.legend(loc = "best")

```

Out[197]:



In [198]:

```

predictions = model.evaluate(x_test, y_test)

10/10 [=====] - 0s 10ms/step - loss: 0.8998 - accuracy: 0.7752

```

In [199]:

```

print(f'LOSS : {predictions[0]}')
print(f'ACCURACY : {predictions[1]}')

```

LOSS : 0.899788498878479
ACCURACY : 0.7752442955970764

In []: