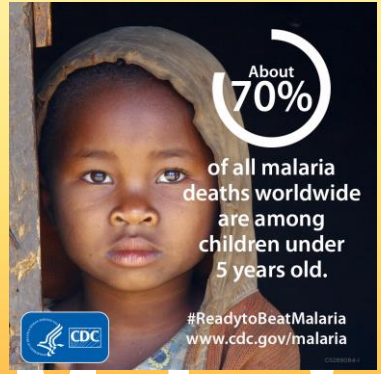


Lionel Tambe Eyong Nkongoh
Deep Learning Project

Malaria Cells Image Detection

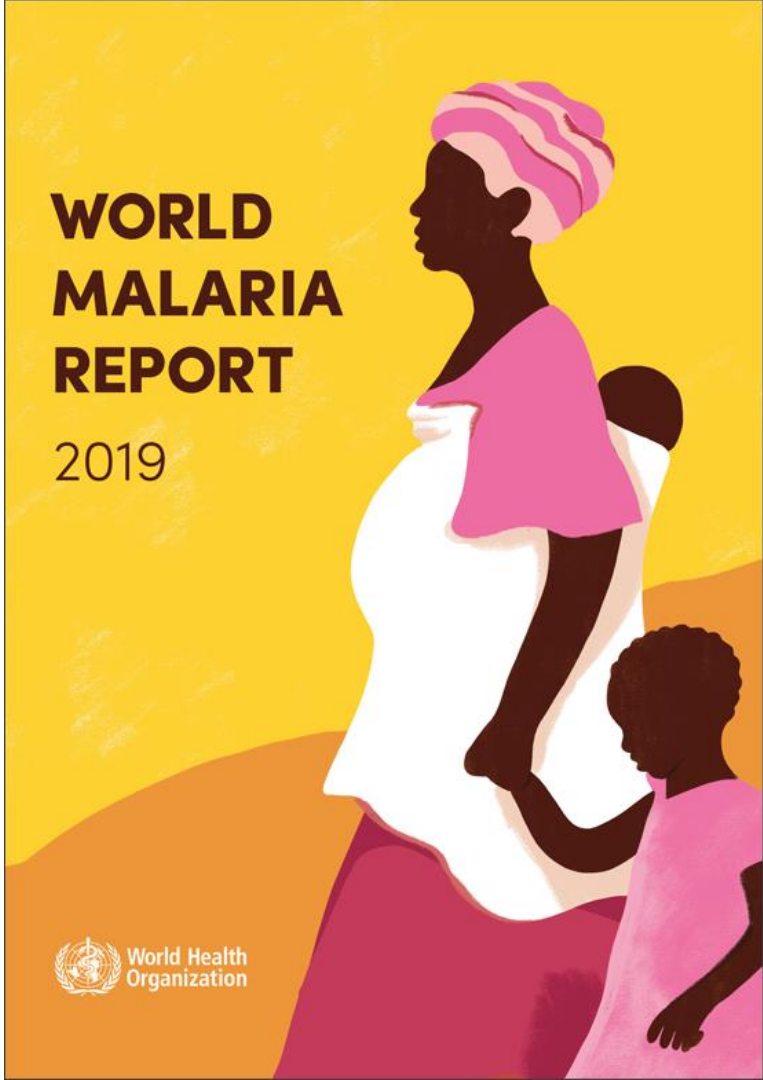
Mit CNN Algorithm

Inhalte

- Motivation
- Datensatz erklären
- Code erklären
- Visualisierungen erklären
- Interpretation
- Fazit

Malaria Cells Image Detection

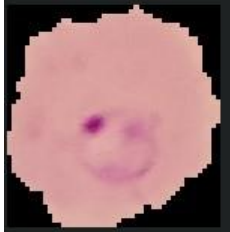
- Die Problematik Malaria, ist immernoch ein Thema in Tropische und Sud Sahara Länder.
- Malaria ist ein wichtiger Faktor für die höhe Sterberate in Afrika beispielweise.
- Jede effektive Methode Malaria infizierte Zellen zu identifizieren, hilft bei der Bekämpfung



```python

# Import Libraries

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import cv2
import os
import PIL
import pathlib
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")
from matplotlib.pyplot import figure




data = pathlib.Path('C:/Users/alfa/Desktop/Project/cell_images')  # Naming the path of the dataset



image_count = len(list(data.glob('*/*.png')))  # Counting the dataset
print(image_count)


parasitized = list(data.glob('Parasitized/*'))      # Visualizing the morphology of a parasitized cell
PIL.Image.open(str(parasitized[0]))

uninfected = list(data.glob('Uninfected/*'))        #Visualizing the morphology of an uninfected cell
PIL.Image.open(str(uninfected[0]))

```

```
2022-08-04 22:27:41.945688: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2022-08-04 22:27:41.945907: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror
if you do not have a GPU set up on your machine.
27558
Found 27558 files belonging to 2 classes.
Using 22047 files for training.
```

```
In [2]: parasitized = list(data.glob('Parasitized/*'))      # Visualizing the morphology of a parasitized cell
   ...: PIL.Image.open(str(parasitized[0]))
Out[2]:
```

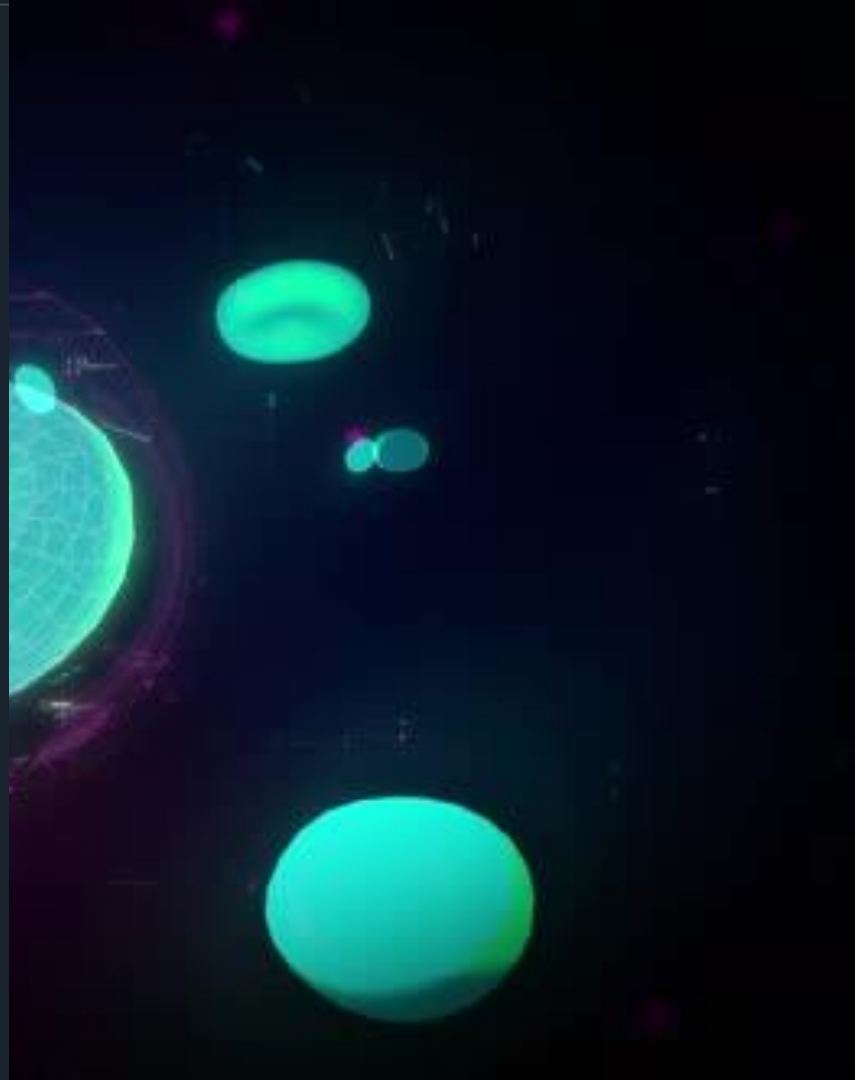


```
In [3]: uninfected = list(data.glob('Uninfected/*'))        #Visualizing the morphology of an uninfected cell
   ...: PIL.Image.open(str(uninfected[0]))
Out[3]:
```

```python
# Data Loading
# Creating Dataset

batch_size = 200
img_height = 75
img_width = 75

train_ds = tf.keras.utils.image_dataset_from_directory(
  data,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)




val_ds = tf.keras.utils.image_dataset_from_directory(
  data,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)


class_names = train_ds.class_names
print(class_names)
```

```
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Found 27558 files belonging to 2 classes.
Using 5511 files for validation.
['Parasitized', 'Uninfected']
Model: "sequential"
```

```python


# Defining the model architecture

model = keras.Sequential()

model.add(keras.layers.Conv2D(16, kernel_size=3,strides=1, activation=None,
                              input_shape=(75,75,3)))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Dropout(0.2))

model.add(keras.layers.Conv2D(32, kernel_size=3,strides=1, activation=None))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))

model.add(keras.layers.Conv2D(64, kernel_size=3,strides=1, activation=None))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Dropout(0.3))

model.add(keras.layers.Conv2D(32, kernel_size=3,strides=1, activation=None))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))

model.add(keras.layers.Conv2D(16, kernel_size=3,strides=1, activation=None))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Dropout(0.3))

model.add(keras.layers.Conv2D(8, kernel_size=3,strides=1, activation=None))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation('relu'))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(64,activation='relu'))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(units=1))
model.add(keras.layers.Activation('sigmoid'))

```



```python
# Summary of the model


model.summary()


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



# Training the model on training dataset and validating it on validation data

history = model.fit(train_ds,
                    epochs=20,
                    validation_data=val_ds,
                    batch_size=batch_size)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 73, 73, 16)        448

 batch_normalization (BatchN  (None, 73, 73, 16)       64
 ormalization)

 activation (Activation)     (None, 73, 73, 16)        0

 max_pooling2d (MaxPooling2D  (None, 36, 36, 16)       0
 )

 dropout (Dropout)           (None, 36, 36, 16)        0

 conv2d_1 (Conv2D)           (None, 34, 34, 32)        4640

 batch_normalization_1 (Batc  (None, 34, 34, 32)       128
 hNormalization)

 activation_1 (Activation)   (None, 34, 34, 32)        0

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        18496

 batch_normalization_2 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 activation_2 (Activation)   (None, 32, 32, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 64)        0

 conv2d_3 (Conv2D)           (None, 14, 14, 32)        18464

 batch_normalization_3 (Batc  (None, 14, 14, 32)       128
 hNormalization)

 activation_3 (Activation)   (None, 14, 14, 32)        0

 conv2d_4 (Conv2D)           (None, 12, 12, 16)        4624

 batch_normalization_4 (Batc  (None, 12, 12, 16)       64
 hNormalization)

 activation_4 (Activation)   (None, 12, 12, 16)        0
```

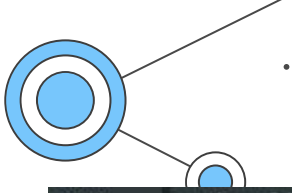
```
 activation_4 (Activation)   (None, 12, 12, 16)        0

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 16)         0
 2D)

 dropout_2 (Dropout)         (None, 6, 6, 16)          0

 conv2d_5 (Conv2D)           (None, 4, 4, 8)           1160

 batch_normalization_5 (Batc  (None, 4, 4, 8)          32
 hNormalization)

 activation_5 (Activation)   (None, 4, 4, 8)           0

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

 activation_6 (Activation)   (None, 1)                 0

=================================================================
Total params: 56,825
Trainable params: 56,489
Non-trainable params: 336
```

```
_____________________________________________________
Epoch 1/20
111/111 [==============================] - 122s 1s/step - loss: 0.5730 - accuracy: 0.7006 - val_loss: 0.7713 -
val_accuracy: 0.6788
Epoch 2/20
111/111 [==============================] - 116s 1s/step - loss: 0.2544 - accuracy: 0.9060 - val_loss: 0.1823 -
val_accuracy: 0.9363
Epoch 3/20
111/111 [==============================] - 113s 1s/step - loss: 0.1770 - accuracy: 0.9405 - val_loss: 0.1843 -
val_accuracy: 0.9316
Epoch 4/20
111/111 [==============================] - 112s 1s/step - loss: 0.1632 - accuracy: 0.9460 - val_loss: 0.1520 -
val_accuracy: 0.9505
Epoch 5/20
111/111 [==============================] - 112s 1s/step - loss: 0.1547 - accuracy: 0.9493 - val_loss: 0.1851 -
val_accuracy: 0.9365
Epoch 6/20
111/111 [==============================] - 113s 1s/step - loss: 0.1472 - accuracy: 0.9528 - val_loss: 0.1735 -
val_accuracy: 0.9443
Epoch 7/20
111/111 [==============================] - 124s 1s/step - loss: 0.1432 - accuracy: 0.9531 - val_loss: 0.1324 -
val_accuracy: 0.9563
Epoch 8/20
111/111 [==============================] - 123s 1s/step - loss: 0.1389 - accuracy: 0.9542 - val_loss: 0.1329 -
val_accuracy: 0.9550
Epoch 9/20
111/111 [==============================] - 113s 1s/step - loss: 0.1362 - accuracy: 0.9563 - val_loss: 0.1361 -
val_accuracy: 0.9550
Epoch 10/20
111/111 [==============================] - 110s 990ms/step - loss: 0.1331 - accuracy: 0.9566 - val_loss: 0.2308 -
val_accuracy: 0.9385
Epoch 11/20
111/111 [==============================] - 113s 1s/step - loss: 0.1309 - accuracy: 0.9566 - val_loss: 0.1422 -
val_accuracy: 0.9510
Epoch 12/20
111/111 [==============================] - 129s 1s/step - loss: 0.1273 - accuracy: 0.9590 - val_loss: 0.1992 -
val_accuracy: 0.9452
Epoch 13/20
111/111 [==============================] - 152s 1s/step - loss: 0.1261 - accuracy: 0.9587 - val_loss: 0.1266 -
val_accuracy: 0.9552
Epoch 14/20
111/111 [==============================] - 147s 1s/step - loss: 0.1247 - accuracy: 0.9586 - val_loss: 0.1384 -
val_accuracy: 0.9552
Epoch 15/20
111/111 [==============================] - 152s 1s/step - loss: 0.1223 - accuracy: 0.9595 - val_loss: 0.2261 -
val_accuracy: 0.9392
Epoch 16/20
111/111 [==============================] - 149s 1s/step - loss: 0.1204 - accuracy: 0.9604 - val_loss: 0.1372 -
val_accuracy: 0.9559
```

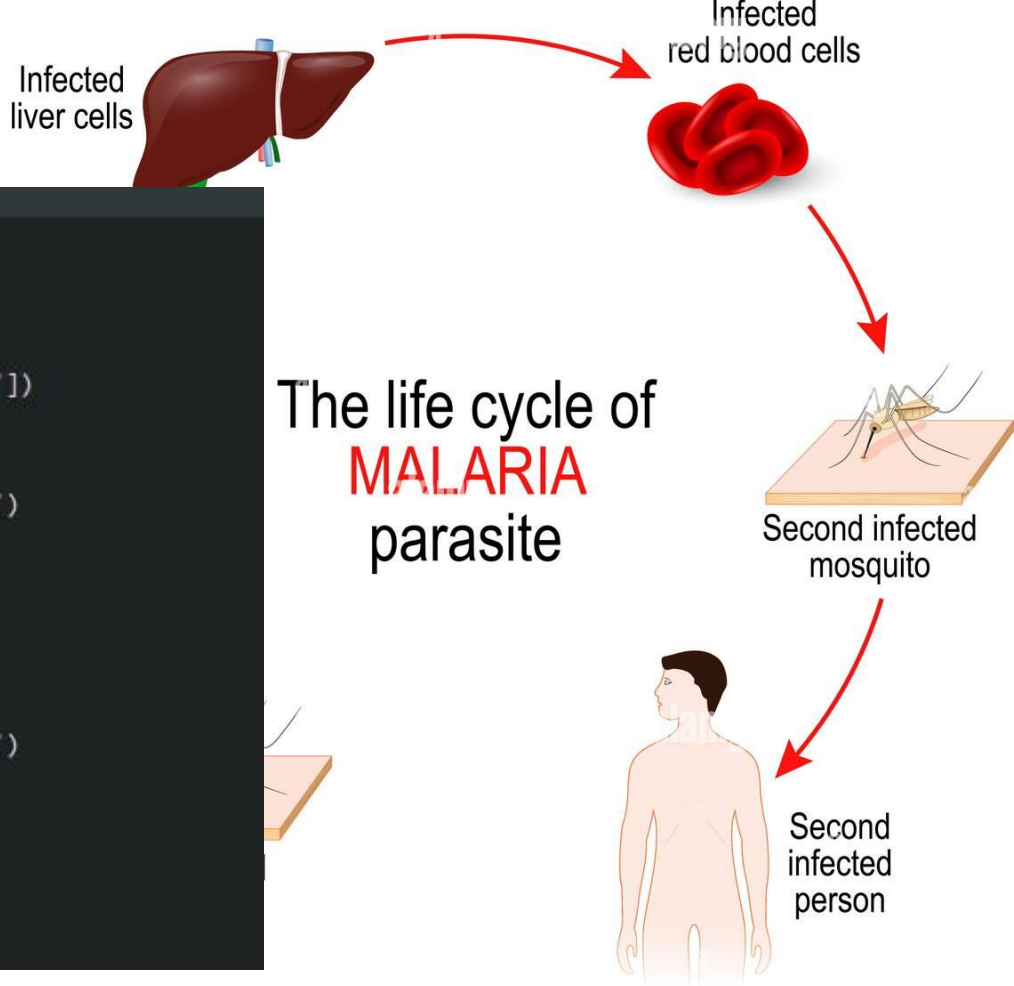
```
Epoch 16/20
111/111 [==============================] - 149s 1s/step - loss: 0.1204 - accuracy: 0.9604 - val_loss: 0.1372 -
val_accuracy: 0.9559
Epoch 17/20
111/111 [==============================] - 150s 1s/step - loss: 0.1202 - accuracy: 0.9594 - val_loss: 0.1620 -
val_accuracy: 0.9510
Epoch 18/20
111/111 [==============================] - 149s 1s/step - loss: 0.1137 - accuracy: 0.9611 - val_loss: 0.1572 -
val_accuracy: 0.9525
Epoch 19/20
111/111 [==============================] - 154s 1s/step - loss: 0.1191 - accuracy: 0.9601 - val_loss: 0.1618 -
val_accuracy: 0.9459
Epoch 20/20
111/111 [==============================] - 151s 1s/step - loss: 0.1151 - accuracy: 0.9609 - val_loss: 0.1333 -
val_accuracy: 0.9574
```

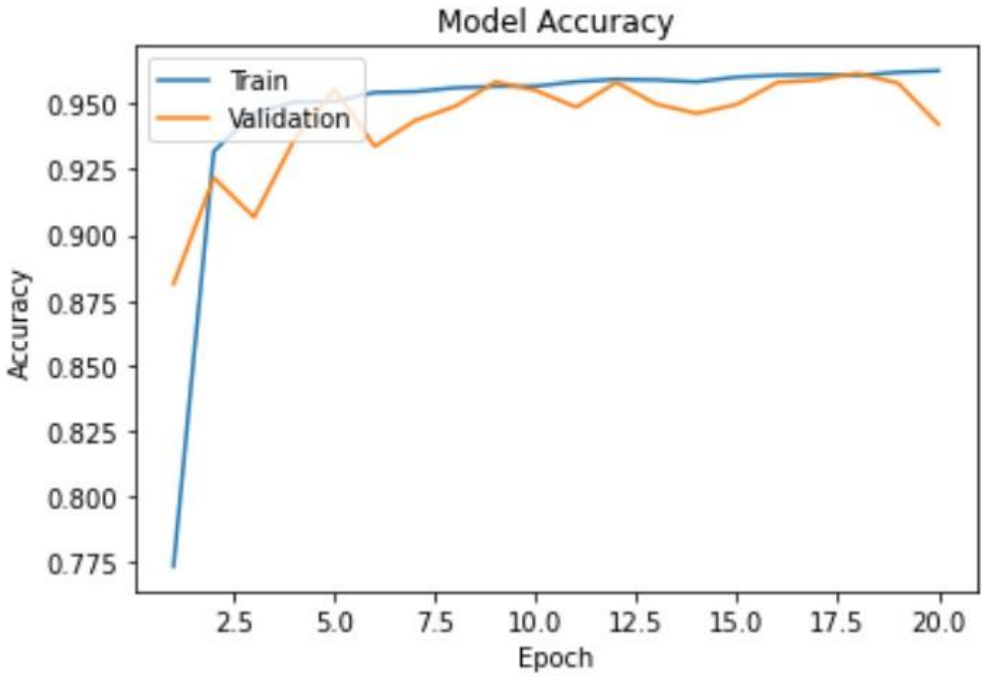
Warning

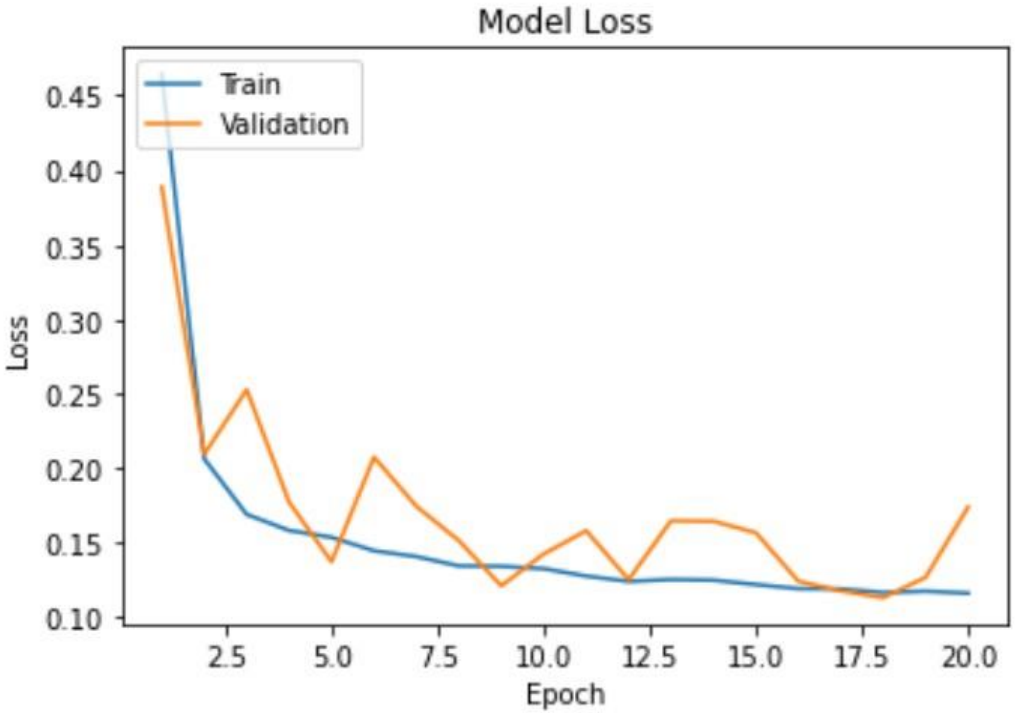
Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.



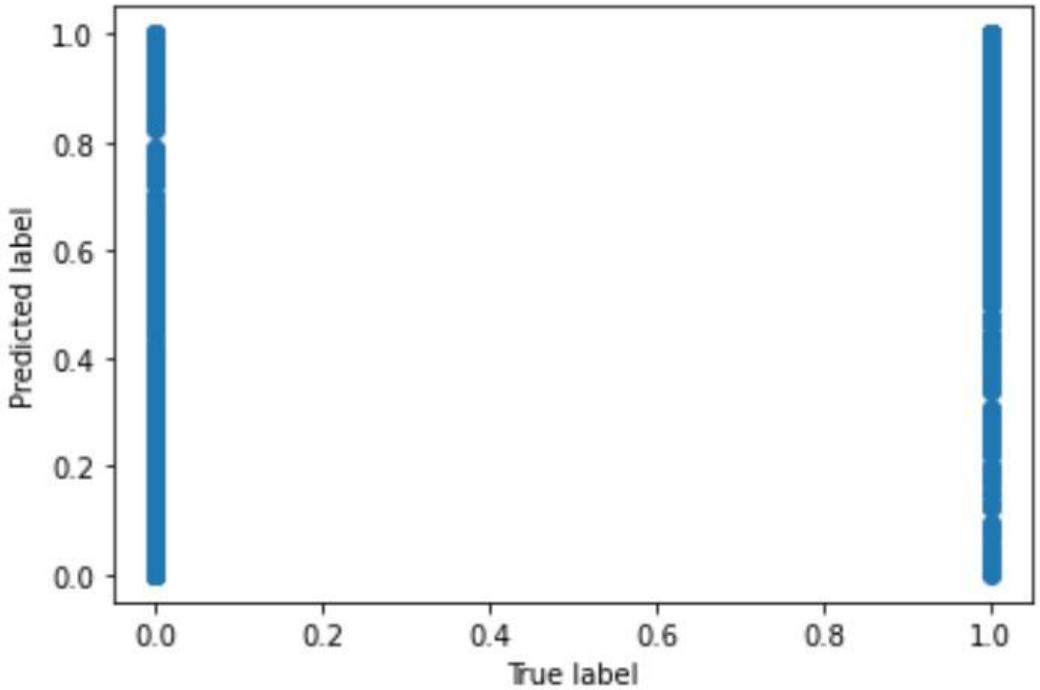
```python
# Visualizing the Learning Curve


def plotLearningCurve(history,epochs):
  epochRange = range(1,epochs+1)
  plt.plot(epochRange,history.history['accuracy'])
  plt.plot(epochRange,history.history['val_accuracy'])
  plt.title('Model Accuracy')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend(['Train','Validation'],loc='upper left')
  plt.show()

  plt.plot(epochRange,history.history['loss'])
  plt.plot(epochRange,history.history['val_loss'])
  plt.title('Model Loss')
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.legend(['Train','Validation'],loc='upper left')
  plt.show()

plotLearningCurve(history,20)


# Confusion Matrix

```

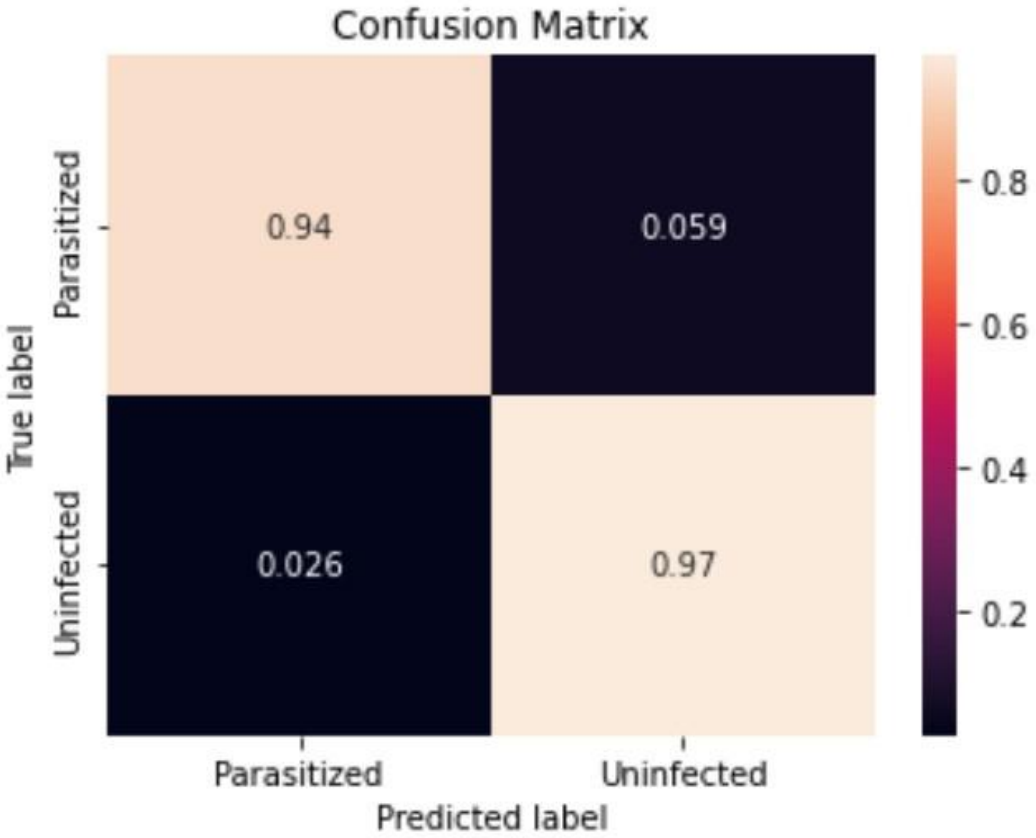



Model Accuracy
Train
Validation
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch


Model Loss
Train
Validation
Loss
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch

```python
# Confusion Matrix


all_labels = []
all_y_preds = []

for images, labels in val_ds:
    print('images.shape:', images.shape)
    print('labels.shape:', labels.shape)

    y_test_pred = model.predict(images)
    all_labels.extend(labels)
    all_y_preds.extend(y_test_pred)

plt.scatter(all_labels, all_y_preds)
plt.xlabel('True label')
plt.ylabel('Predicted label')
plt.show()


y_test_pred = model.predict(val_ds)
y_test = np.concatenate([y for x, y in val_ds], axis=0)

y_test_pred_class = np.round(all_y_preds).reshape(-1,1)
y_test_class      = np.round(all_labels).reshape(-1,1)



# Creates a confusion matrix
cm = confusion_matrix(y_test_class, y_test_pred_class, normalize='true')

# Transform to df for easier plotting
cm_df = pd.DataFrame(cm,
                     index = ['Parasitized','Uninfected'],
                     columns = ['Parasitized','Uninfected'])

plt.figure(figsize=(5.5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
Predicted label
0.0
0.2
0.4
0.6
0.8
1.0
True label


Confusion Matrix
0.94
0.059
0.026
0.97
Parasitized
Uninfected
True label
Parasitized
Uninfected
Predicted label
0.8
0.6
0.4
0.2

```
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 1s 50ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 43ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 45ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 43ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 45ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 43ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 44ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 44ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 44ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 46ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 44ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 46ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 43ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 42ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 47ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
```

```
labels.shape: (200,)
7/7 [==============================] - 0s 47ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 48ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 50ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 46ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 48ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 44ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 48ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 54ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 50ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 48ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 47ms/step
images.shape: (200, 75, 75, 3)
labels.shape: (200,)
7/7 [==============================] - 0s 46ms/step
images.shape: (111, 75, 75, 3)
labels.shape: (111,)
4/4 [==============================] - 0s 49ms/step
28/28 [==============================] - 11s 364ms/step

In [16]:
```

Interpretation:

- Aus der Confusion Matrix, kann man erkennen wo TP, FP, TN, FN liegen:

TP	**FN**
FP	**TN**

- Die Precision kann ausgerechnet werden: Precision = TP/(TP+FP)

$\Rightarrow$ 0.94 / (0.94+0.026) = 0.9731

Entspricht ein Precision von 97.3%

- Der Recall kann auch ausgerechnet werden: Recall = TP /(TP+FN)

$\Rightarrow$ 0.94/(0.94+0.059) = 0.9409

Entspricht ein Recall von 94.1%

Fazit

- Ziel der vorliegenden Projekt war Malaria infizierte Zellen von nicht infizierte
- Zellen zu unterscheiden/erkennen (Binary Image Classification of Malaria cells).
- Die Ergebnisse des Projekts zeigen dass CNN (Convolutional Neural Networks), ein sehr guter Modell ist für die Klassifikation von Cell Images (Mikroskopische Bilder).
- Die Convolutional und Pooling Layers helfen bei der Data Preprocessing.
- Die Ergebnisse der Binary Image Classification: accuracy von 0.9609, val_loss: 0.1333, val_accuracy : 0.9574
- Entspricht ein Prozentsatz von 95.7%.
- Ein Vergleich mit anderen Modellen ist mit der erzielten 95.7%, nicht nötig.
- Außerdem habe ich leider ein Mangel an Hardware Memory Leistung gehabt.
- Versuche mit weiteren Modellen hätte größe Einfluss auf das Zeitmanagement gehabt.

Vielen Dank für Ihre Aufmerksamkeit

Quelle Datensatz:
https://www.kaggle.com/