NAME: CHRISTIAN IKEDINMA

COURSE ID: MACHINE LEARNING

STUDENT ID: 22063034

GITHUB LINK:

https://github.com/Ezenwaikedi1/malaria-forcasting-and-predication-of-image-using-deep-learning-models-

LINK TO THE IMAGE DATASET ON KAGGLE:

https://www.kaggle.com/datasets/iarunava/cell-images-for-detecting-malaria

MALARIA FORECASTING AND IMAGE-BASED DETECTION USING DEEP LEARNING TECHNIQUES

ABSTRACT

Malaria continues to be one of the world's most serious public health issues, particularly in tropical and subtropical areas. Effective disease management depends on automated image-based detection of parasitized blood cells and accurate malaria case forecasting. This study uses a dataset of reported cases and a collection of blood smear images to investigate a deep learning-driven method for malaria prediction and detection. Convolutional Neural Networks (CNNs), VGG16, InceptionV3, Xception, and DenseNet are among the deep learning models used in the study. The custom CNN performs the best among all, attaining a 96% accuracy rate. Because of its ease of use and effectiveness, it works especially effectively in settings with little resources. A strong framework for improving malaria diagnostics and public health treatments is provided by the combination of CNN-based detection and forecasting models

1. INTRODUCTION

With approximately 240 million cases per year, malaria, which is spread by Anopheles mosquitoes and caused by Plasmodium parasites, poses a serious danger to world health. Timely interventions and resource allocation depend on early detection and precise case prognosis. Modern quick diagnostic procedures have sensitivity limitations, and traditional microscopy techniques are time-consuming and skill-intensive. CNNs are extremely effective in picture recognition applications, and deep learning provides automatic and effective solutions.

This study uses a Python-based methodology that combines deep learning approaches for identifying infected blood cells from microscopic pictures with EDA for comprehending trends in

malaria cases. The paper applies CNN architectures to improve diagnostic accuracy using the Kaggle malaria dataset and a reported cases dataset.(Salam et al., 2024)

2. LITERATURE REVIEW

2.1 OVERVIEW OF MALARIA DETECTION

Malaria, which disproportionately affects low- and middle-income nations, continues to be one of the most urgent global health issues. For identifying malaria parasites, traditional diagnostic techniques include microscopic analysis of Giemsa-stained blood smears are the gold standard. These techniques, however, are unworkable in settings with limited resources since they are labor-intensive, prone to human error, and require highly skilled technicians. Despite providing quicker results, rapid diagnostic tests (RDTs) have lower sensitivity, especially when dealing with non-falciparum species and low parasite numbers.(Boit & Patil, 2024)

2.2 ADVANCEMENT IN MACHINE LEARNING AND DEEP LEARNING

Medical diagnostics has been transformed by machine learning (ML) and deep learning (DL), which automate processes like feature extraction and categorization. Red blood cells (RBCs) with and without parasites can be distinguished from microscopic images with remarkable accuracy using Convolutional Neural Networks (CNNs). Research has demonstrated that CNN-based techniques outperform conventional techniques in terms of accuracy, speed, and efficiency.(Antonio et al., 2024)

Recent advancements include:

Transfer Learning: Pre-trained on massive image datasets, models such as VGG16, InceptionV3, Xception, and DenseNet have been optimized for the detection of malaria. Although these models are excellent at extracting features, their intricacy frequently necessitates greater processing power.

Custom Architectures: CNN models that are lightweight and tailored to certain datasets achieve competitive accuracy while consuming less computing power. These are especially helpful in environments with limited resources.

Preprocessing and Data Augmentation: Methods such as scaling, normalization, and augmentation have enhanced the generalization of models across a variety of datasets. (Dr. M. Praneesh et al., 2024)

2.3 CHALLENGES IN AUTOMATED MALARIA DETECTION

Despite tremendous advancements, a number of difficulties still exist:

Data Variability: Model generalization may be impacted by variations in staining methods, imaging settings, and dataset morphology.

Computational Requirements: Many cutting-edge models require powerful GPUs or TPUs, which restricts their use in settings with limited resources.

Model Interpretability: CNNs perform exceptionally well, but in therapeutic settings where explain ability is essential, their "black-box" nature presents difficulties.

3. METHODOLOGY

3.1 DATASET OVERVIEW

The reported cases dataset includes annual records of malaria cases and deaths categorized by country and region. The analysis aims to uncover trends and regional disparities in malaria prevalence.

```
[2]: import matplotlib.image as mpimg
     import os
     import shutil
     import tensorflow as tf
     from tensorflow.keras.applications import ResNet50
     from sklearn.model_selection import train_test_split
     from tensorflow.keras.preprocessing.image import ImageDataGenerator

[3]: df=pd.read_csv('reported_numbers.csv')

[4]: df.head()
```

[4]:

	Country	Year	No. of cases	No. of deaths	WHO Region
0	Afghanistan	2017	161778.0	10.0	Eastern Mediterranean
1	Algeria	2017	0.0	0.0	Africa
2	Angola	2017	3874892.0	13967.0	Africa
3	Argentina	2017	0.0	1.0	Americas
4	Armenia	2017	0.0	NaN	Europe

FIGURE 1: MALARIA CSV DATASET

a. DATA CLEANING AND PROCESSING

The dataset underwent preprocessing to ensure quality:

- Missing values were handled by removing incomplete records.
- Columns were renamed for consistency.
- Data was aggregated by country and year for analysis.

```
[5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1944 entries, 0 to 1943
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Country        1944 non-null   object
 1   Year           1944 non-null   int64
 2   No. of cases   1710 non-null   float64
 3   No. of deaths  1675 non-null   float64
 4   WHO Region     1944 non-null   object
dtypes: float64(2), int64(1), object(2)
memory usage: 76.1+ KB

[6]: df.describe()
```

[6]:

	Year	No. of cases	No. of deaths
count	1944.000000	1.710000e+03	1675.000000
mean	2008.500000	3.897303e+05	1289.413731
std	5.189462	1.270270e+06	4290.739997
min	2000.000000	0.000000e+00	0.000000
25%	2004.000000	5.937500e+02	1.000000
50%	2008.500000	1.479200e+04	30.000000
75%	2013.000000	1.170978e+05	669.500000
max	2017.000000	1.533084e+07	51842.000000

```
[7]: df.columns

[7]: Index(['Country', 'Year', 'No. of cases', 'No. of deaths', 'WHO Region'], dtype='object')

[8]: df.isnull().sum()

[8]: Country           0
     Year              0
     No. of cases    234
     No. of deaths   269
     WHO Region        0
     dtype: int64

[9]: df.dropna(inplace=True)

[10]: df.isnull().sum()

[10]: Country         0
      Year            0
      No. of cases    0
      No. of deaths   0
      WHO Region      0
      dtype: int64

[11]: df.Year.value_counts()

[11]: 2013    101
      2014     99
      2015     98
      2012     98
      2010     98
      2016     98
      2011     97
      2009     94
      2008     92
      2007     87
      2006     86
      2017     85
      2005     84
      2004     76
      2003     68
      2001     68
      2002     67
      2000     58
      Name: Year, dtype: int64

[12]: df.Country.value_counts()

[12]: Myanmar               18
      Dominican Republic    18
      El Salvador           18
      Guatemala             18
      Honduras              18
                            ..
      Egypt                  6
      South Sudan            6
      Cameroon               5
      Nigeria                4
      Zambia                 4
```

FIGURE 2: DATA CLEANING AND PREPROCESSING

b. EXPLORATORY DATA ANALYSIS

1. **Top Affected Countries**: Bar plots reveal the countries with the highest cases and deaths, with significant disparities between regions.

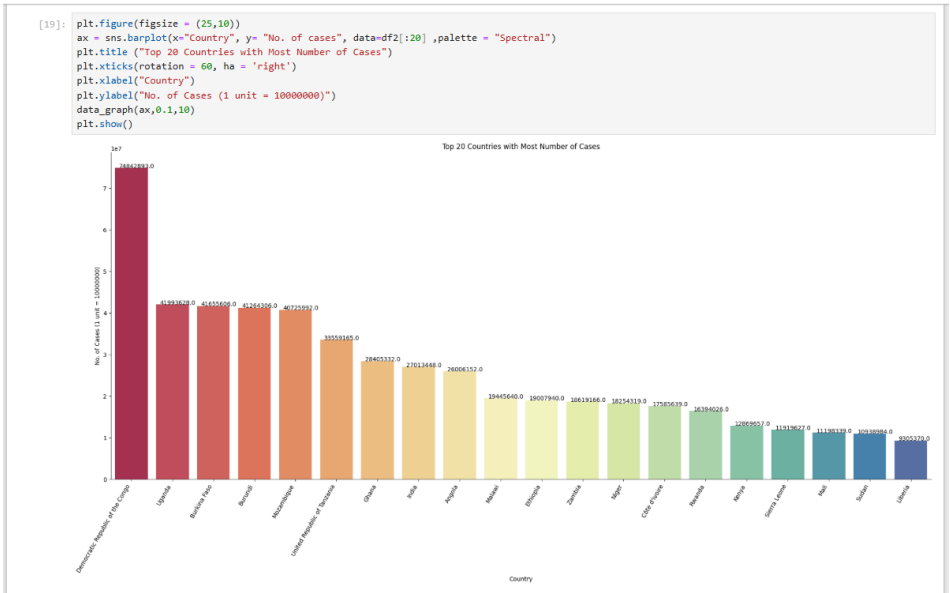


FIGURE 3: BAR CHART SHOWING TOP AFFECTED COUNTRIES

2. **Temporal Trends**: Line plots show a steady decline in cases and deaths globally, indicating progress in malaria control efforts.

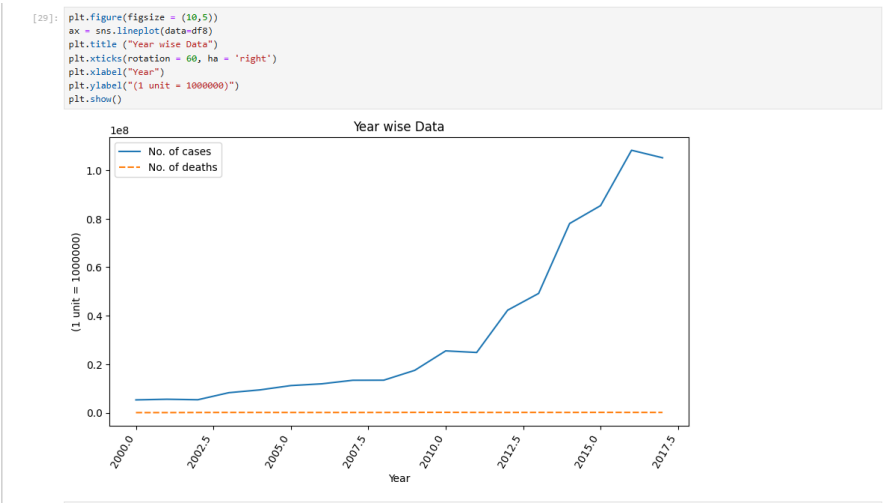


FIGURE 4: LINE PLOT SHOWING THE YEARS FOR CASES OF MALARIA DEATHS

3. **Regional Analysis**: Aggregated data by WHO regions highlights the burden of malaria in sub-Saharan Africa compared to other regions.

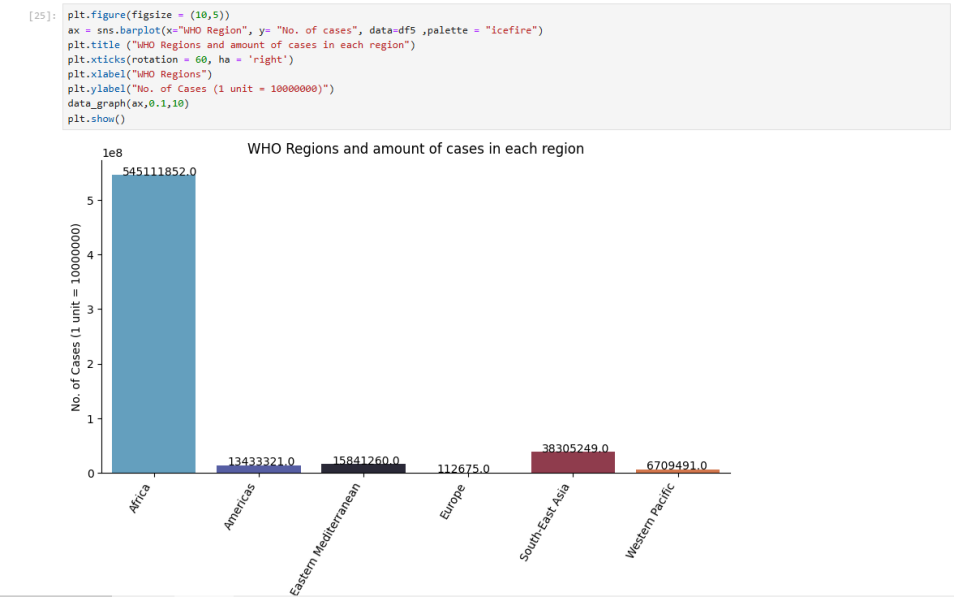


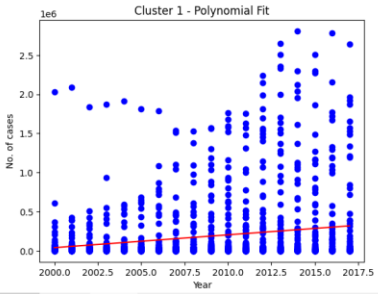
FIGURE 5: AMOUNT OF CASES OF MALARIA IN EACH REGION

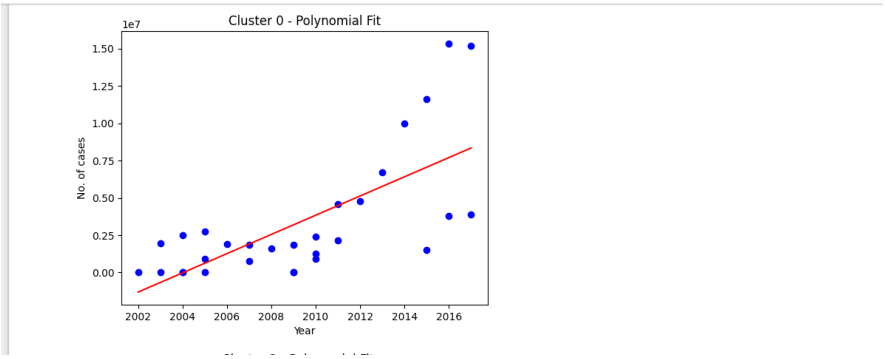
3.4 **FORECASTING**

Linear and polynomial regression models were employed to forecast future cases:

- A polynomial regression model of degree 3 demonstrated superior fitting, capturing the non-linear trends in the data.

```
[38]: # Iterate over unique cluster labels
for cluster_label in df['Cluster'].unique():
    cluster_data = df[df['Cluster'] == cluster_label]
    X_cluster = cluster_data['Year'].values.reshape(-1, 1)
    y_cluster = cluster_data['No. of cases'].values  # Fix typo here

    # Polynomial fitting
    poly_reg = LinearRegression()
    poly_reg.fit(X_cluster, y_cluster)

    # Visualize the polynomial fit
    plt.scatter(X_cluster, y_cluster, color='blue')
    plt.plot(X_cluster, poly_reg.predict(X_cluster), color='red')
    plt.title(f'Cluster {cluster_label} - Polynomial Fit')
    plt.xlabel('Year')
    plt.ylabel('No. of cases')
    plt.show()
```





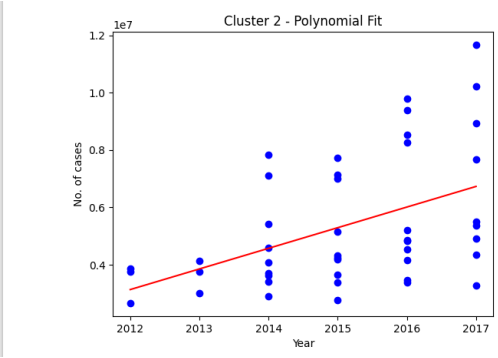


FIGURE 6: POLYNOMIALS FITTING

- Exponential fitting provided additional insights into the growth patterns of malaria cases.

```
[37]: from scipy.optimize import curve_fit

# Define exponential function
def exponential_func(x, a, b):
    return a * np.exp(b * x)

# Curve fitting
popt, pcov = curve_fit(exponential_func, X.flatten(), y)

# Predictions
y_exp_pred = exponential_func(X.flatten(), *popt)

# Visualize the exponential fitting
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, y_exp_pred, color='green', label='Exponential fit')
plt.title('Exponential Fitting of Malaria Cases Over Time')
plt.xlabel('Year')
plt.ylabel('No. of cases')
plt.legend()
plt.show()
```

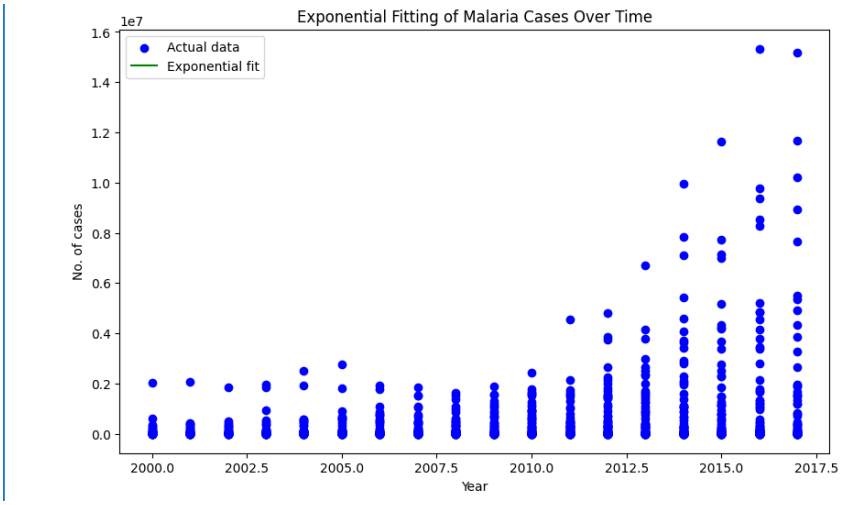


FIGURE 7: EXPONENTIAL FITTING

3.5 CLUSTERING AND ADVANCED ANALYSIS

To complement image-based detection, clustering techniques were applied to the reported cases dataset:

- **K-Means Clustering**: Grouped countries based on malaria cases and deaths into three clusters, providing insights into regional disease patterns
- Scatter plots visualized the clusters, emphasizing the disparity in disease burden.

```
[34]: from sklearn.cluster import KMeans
      from sklearn.preprocessing import StandardScaler

      # Select relevant features for clustering
      features = df[['No. of cases', 'No. of deaths']]

      # Standardize the features
      scaler = StandardScaler()
      scaled_features = scaler.fit_transform(features)

      # Perform KMeans clustering
      kmeans = KMeans(n_clusters=3, random_state=42)
      df['Cluster'] = kmeans.fit_predict(scaled_features)

      # Visualize clusters
      plt.figure(figsize=(8, 6))
      plt.scatter(df['No. of cases'], df['No. of deaths'], c=df['Cluster'], cmap='viridis')
      plt.xlabel('No. of cases')
      plt.ylabel('No. of deaths')
      plt.title('Clustering of Countries based on Malaria Cases and Deaths')
      plt.show()
```

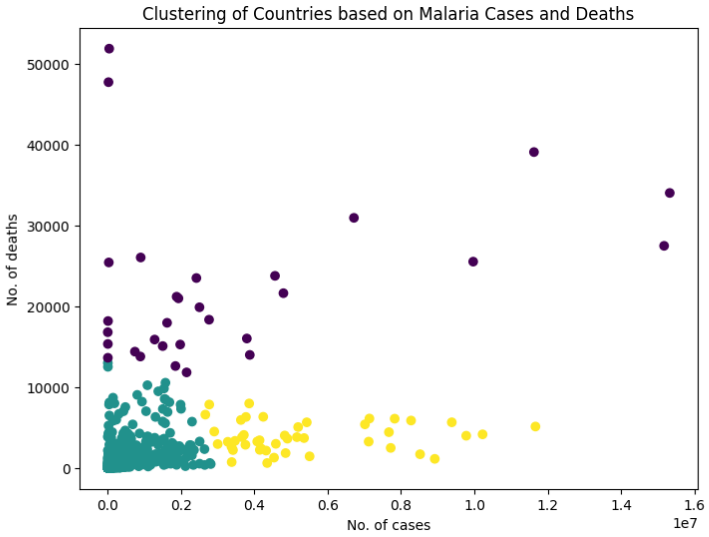


FIGURE 8: K MEANS CLUSTERING

3.6 DEEP LEARNING

3.6.1 MALARIA IMAGE DATASET

The Kaggle malaria dataset comprises 27,558 labeled images of parasitized and uninfected red blood cells.

```
[43]: from PIL import Image
image_path= r'C:\Users\ibuku\Downloads\cell_images\cell_images\Parasitized\C33P1thinF_IMG_20150619_114756a_cell_179.png'
image = Image.open(image_path)

# Get the dimensions (width x height) of the image
width, height = image.size

print("Image size:", width, "x", height)

Image size: 142 x 163

[44]: WIDTH = 151
HEIGHT = 136
BATCH_SIZE = 32
CHANNELS =3
EPOCHS = 30
NEW_SIZE = 136

[45]: train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
            directory=train_dir,
            shuffle=True,
            image_size=(WIDTH,HEIGHT),
            batch_size = BATCH_SIZE,
)

Found 22048 files belonging to 2 classes.

[46]: test_dataset = tf.keras.preprocessing.image_dataset_from_directory(
            directory=test_dir ,
            shuffle=True,
            image_size=(WIDTH,HEIGHT),
            batch_size = BATCH_SIZE,
)

Found 5510 files belonging to 2 classes.

[47]: classnames = train_dataset.class_names
classnames

[47]: ['Parasitized', 'Uninfected']
```

```
[48]: plt.figure(figsize = (15,15))
for image, label in train_dataset.take(1):
    for i in range(12):
        plt.subplot(3, 4, i+1)
        plt.imshow(image[i].numpy().astype('uint8'))
        plt.title(classnames[label[i]])
        plt.axis('off')
```

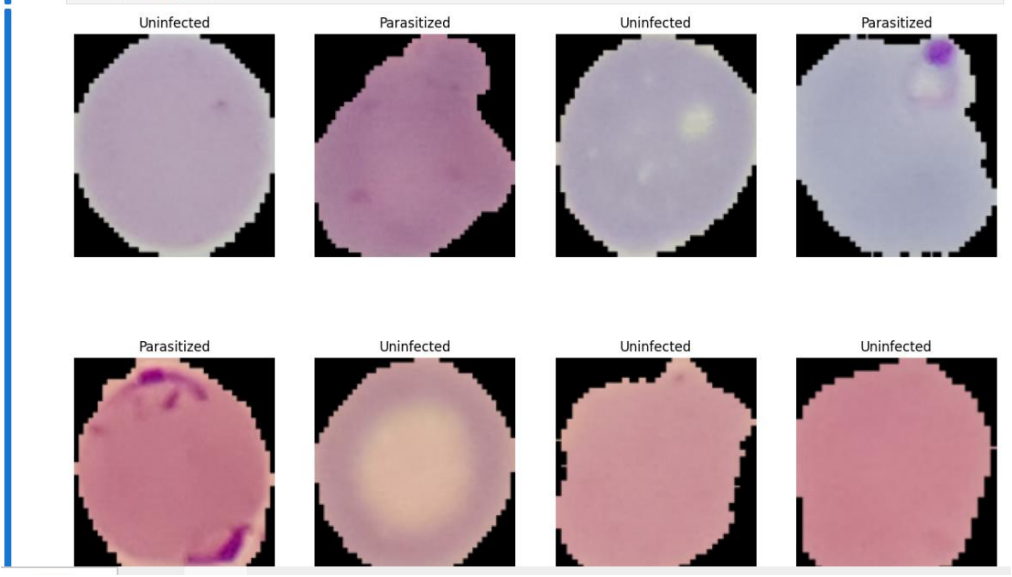


FIGURE 9: MALARIA IMAGE DATASET

Preprocessing included:

- Resizing images to a standard size of pixels for computational efficiency.
- Normalizing pixel values to the range [0, 1] for uniformity.

- Splitting data into training (80%), validation (10%), and testing (10%) subsets.

```
[49]: # Assign the directory of Parasitized and Uninfected cell images
parasitized_dir = r"C:\Users\ibuku\Downloads\cell_images\cell_images\Parasitized/"
uninfected_dir = r"C:\Users\ibuku\Downloads\cell_images\cell_images\Uninfected/"

print ('Total of Parasitized cell images: ', len(os.listdir(parasitized_dir)))
print ('Total of Uninfected cell images: ', len(os.listdir(uninfected_dir)))

Total of Parasitized cell images:  13780
Total of Uninfected cell images:  13780

[50]: from matplotlib.image import imread

[51]: # Print the shape image of one training data
image_shape = parasitized_dir + "/C99P60ThinF_IMG_20150918_141001_cell_133.png"
print("Shape of image: ",imread(image_shape).shape)

Shape of image:  (145, 142, 3)

[52]:
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary'
)

Found 22048 images belonging to 2 classes.
Found 5510 images belonging to 2 classes.
```

FIGURE 10: DATA PREPROCESSING

Three deep learning architectures were implemented:

Custom CNN:

CNN, which consists of four convolutional layers, max-pooling, and dense layers with dropout regularization, was created as a lightweight and effective network. This model performed quite well. It was especially well-suited for deployment in environments with limited resources due to its ease of use and minimal computing demands. The customized CNN demonstrated its effectiveness and accessibility by circumventing the intricacies of transfer learning.

```
[53]: # Defining CNN as sequential model
model = Sequential()

# Adding convolution and max-pooling layers
model.add(Conv2D(filters=32, kernel_size=(3,3),input_shape=(134,131,3),activation='relu',padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=2))
model.add(Conv2D(filters=64, kernel_size=(3,3),input_shape=(134,131,3),activation='relu',padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=2))
model.add(Conv2D(filters=128, kernel_size=(3,3),input_shape=(134,131,3),activation='relu',padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=2))
model.add(Conv2D(filters=256, kernel_size=(3,3),input_shape=(134,131,3),activation='relu',padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=2))

# Flattening image from layers
model.add(Flatten())

# Adding dense layers
model.add(Dense(128,activation='relu'))
# Adding dropout to minimize overfitting issue
model.add(Dropout(0.2))
model.add(Dense(50,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1,activation='sigmoid'))

# Compiling the model
model.compile(loss='binary_crossentropy',optimizer='adam' ,metrics=["accuracy"])
# model.compile (loss = 'categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])

# Show summary of the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 134, 131, 32)      896

 max_pooling2d (MaxPooling2D  (None, 67, 65, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 67, 65, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 33, 32, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 33, 32, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 128)               2097280

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 50)                6450

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 2,492,197
Trainable params: 2,492,197
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 11: CNN MODEL

VGG16:

One popular transfer learning architecture that has been pre-trained on the ImageNet dataset is VGG16. Robust feature extraction from blood smear images was made possible by its uniform layer design and deep architecture. adjusting VGG16 using the malaria data set. It was better suited

for high-performance situations, but its high computing demands were a drawback, especially in low-resource applications.

```
[61]: from tensorflow.keras.applications import VGG16
      from tensorflow.keras.models import Model
      from tensorflow.keras.layers import Flatten, Dense, Dropout
      from tensorflow.keras.optimizers import Adam

      # Load the pre-trained VGG16 model without the top layers
      base_model = VGG16(weights='imagenet', include_top=False, input_shape=(134, 131, 3))

      # Freeze the layers in the base model
      for layer in base_model.layers:
          layer.trainable = False

      # Flatten the output of the base model
      x = Flatten()(base_model.output)

      # Add fully connected layers
      x = Dense(128, activation='relu')(x)
      x = Dropout(0.2)(x)
      x = Dense(50, activation='relu')(x)
      x = Dropout(0.2)(x)
      output = Dense(1, activation='sigmoid')(x)

      # Create the final model
      model = Model(inputs=base_model.input, outputs=output)

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

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

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 134, 131, 3)]     0

 block1_conv1 (Conv2D)       (None, 134, 131, 64)      1792

 block1_conv2 (Conv2D)       (None, 134, 131, 64)      36928

 block1_pool (MaxPooling2D)  (None, 67, 65, 64)        0

 block2_conv1 (Conv2D)       (None, 67, 65, 128)       73856

 block2_conv2 (Conv2D)       (None, 67, 65, 128)       147584

 block2_pool (MaxPooling2D)  (None, 33, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 33, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 33, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 33, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 flatten_1 (Flatten)         (None, 8192)              0

 dense_3 (Dense)             (None, 128)               1048704

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 50)                6450

 dropout_3 (Dropout)         (None, 50)                0

 dense_5 (Dense)             (None, 1)                 51

=================================================================
Total params: 15,769,893
Trainable params: 1,055,205
Non-trainable params: 14,714,688
_________________________________________________________________
```

FIGURE 12: VGG 19 MODEL

INCEPTION V3:

The employment of inception modules that carry out multi-scale convolutions in simultaneously is what makes InceptionV3 famous. The model was able to capture a variety of elements because to this architectural innovation. Even with its sophisticated architecture, InceptionV3 needed a lot of processing power, which limited its use in settings with limited resources. However, for lab environments with sufficient processing resources, its performance was very dependable.

[66]:
```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the pre-trained InceptionV3 model without the top layers
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(134, 131, 3))

# Freeze the layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Flatten the output of the base model
x = Flatten()(base_model.output)

# Add fully connected layers
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(50, activation='relu')(x)
x = Dropout(0.2)(x)
output = Dense(1, activation='sigmoid')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=output)

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

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

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 134, 131, 3)]	0	[]
conv2d_4 (Conv2D)	(None, 66, 65, 32)	864	['input_2[0][0]']
batch_normalization (BatchNormalization)	(None, 66, 65, 32)	96	['conv2d_4[0][0]']
activation (Activation)	(None, 66, 65, 32)	0	['batch_normalization[0][0]']
conv2d_5 (Conv2D)	(None, 64, 63, 32)	9216	['activation[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 64, 63, 32)	96	['conv2d_5[0][0]']
activation_1 (Activation)	(None, 64, 63, 32)	0	['batch_normalization_1[0][0]']
conv2d_6 (Conv2D)	(None, 64, 63, 64)	18432	['activation_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 64, 63, 64)	192	['conv2d_6[0][0]']
activation_2 (Activation)	(None, 64, 63, 64)	0	['batch_normalization_2[0][0]']
max_pooling2d_4 (MaxPooling2D)	(None, 31, 31, 64)	0	['activation_2[0][0]']
conv2d_7 (Conv2D)	(None, 31, 31, 80)	5120	['max_pooling2d_4[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 31, 31, 80)	240	['conv2d_7[0][0]']
activation_3 (Activation)	(None, 31, 31, 80)	0	['batch_normalization_3[0][0]']
conv2d_8 (Conv2D)	(None, 29, 29, 192)	138240	['activation_3[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 29, 29, 192)	576	['conv2d_8[0][0]']
activation_4 (Activation)	(None, 29, 29, 192)	0	['batch_normalization_4[0][0]']
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 192)	0	['activation_4[0][0]']
conv2d_12 (Conv2D)	(None, 14, 14, 64)	12288	['max_pooling2d_5[0][0]']

```
mixed9_1 (Concatenate)         (None, 2, 2, 768)    0           ['activation_87[0][0]',
                                                                 'activation_88[0][0]']

concatenate_1 (Concatenate)    (None, 2, 2, 768)    0           ['activation_91[0][0]',
                                                                 'activation_92[0][0]']

activation_93 (Activation)     (None, 2, 2, 192)    0           ['batch_normalization_93[0][0]']

mixed10 (Concatenate)          (None, 2, 2, 2048)   0           ['activation_85[0][0]',
                                                                 'mixed9_1[0][0]',
                                                                 'concatenate_1[0][0]',
                                                                 'activation_93[0][0]']

flatten_2 (Flatten)            (None, 8192)         0           ['mixed10[0][0]']

dense_6 (Dense)                (None, 128)          1048704     ['flatten_2[0][0]']

dropout_4 (Dropout)            (None, 128)          0           ['dense_6[0][0]']

dense_7 (Dense)                (None, 50)           6450        ['dropout_4[0][0]']

dropout_5 (Dropout)            (None, 50)           0           ['dense_7[0][0]']

dense_8 (Dense)                (None, 1)            51          ['dropout_5[0][0]']

==================================================================================================
Total params: 22,857,989
Trainable params: 1,055,205
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
```

FIGURE 13: INCEPTION V3

Xception

Depthwise separable convolutions were used by Xception to maximize accuracy and efficiency. When compared to more conventional designs like VGG16, this method drastically decreased the amount of parameters, proving its ability to extract features effectively. Although the bespoke CNN performed marginally better, Xception showed promise as a resource-efficient malaria diagnosis method.

[71]:
```python
from tensorflow.keras.applications import Xception
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the pre-trained Xception model without the top layers
base_model = Xception(weights='imagenet', include_top=False, input_shape=(134, 131, 3))

# Freeze the layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Flatten the output of the base model
x = Flatten()(base_model.output)

# Add fully connected layers
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(50, activation='relu')(x)
x = Dropout(0.2)(x)
output = Dense(1, activation='sigmoid')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=output)

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

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

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 134, 131, 3  0           []
                                )]

block1_conv1 (Conv2D)           (None, 66, 65, 32)   864         ['input_3[0][0]']

block1_conv1_bn (BatchNormaliz  (None, 66, 65, 32)  128         ['block1_conv1[0][0]']
ation)

block1_conv1_act (Activation)   (None, 66, 65, 32)   0           ['block1_conv1_bn[0][0]']

block1_conv2 (Conv2D)           (None, 64, 63, 64)   18432       ['block1_conv1_act[0][0]']

block1_conv2_bn (BatchNormaliz  (None, 64, 63, 64)  256         ['block1_conv2[0][0]']
ation)

block1_conv2_act (Activation)   (None, 64, 63, 64)   0           ['block1_conv2_bn[0][0]']

block2_sepconv1 (SeparableConv  (None, 64, 63, 128)  8768        ['block1_conv2_act[0][0]']
2D)

block2_sepconv1_bn (BatchNorma  (None, 64, 63, 128)  512        ['block2_sepconv1[0][0]']
lization)

block2_sepconv2_act (Activatio  (None, 64, 63, 128)  0          ['block2_sepconv1_bn[0][0]']
n)

block2_sepconv2 (SeparableConv  (None, 64, 63, 128)  17536      ['block2_sepconv2_act[0][0]']
2D)

block2_sepconv2_bn (BatchNorma  (None, 64, 63, 128)  512        ['block2_sepconv2[0][0]']
lization)

conv2d_98 (Conv2D)              (None, 32, 32, 128)  8192        ['block1_conv2_act[0][0]']

block2_pool (MaxPooling2D)      (None, 32, 32, 128)  0           ['block2_sepconv2_bn[0][0]']

batch_normalization_94 (BatchN  (None, 32, 32, 128)  512        ['conv2d_98[0][0]']
ormalization)

add (Add)                       (None, 32, 32, 128)  0           ['block2_pool[0][0]',
                                                                 'batch_normalization_94[0][0]']

block3_sepconv1_act (Activatio  (None, 32, 32, 128)  0          ['add[0][0]']
```

```
conv2d_101 (Conv2D)             (None, 4, 4, 1024)   745472      ['add_10[0][0]']

block13_pool (MaxPooling2D)     (None, 4, 4, 1024)   0           ['block13_sepconv2_bn[0][0]']

batch_normalization_97 (BatchN  (None, 4, 4, 1024)  4096        ['conv2d_101[0][0]']
ormalization)

add_11 (Add)                    (None, 4, 4, 1024)   0           ['block13_pool[0][0]',
                                                                 'batch_normalization_97[0][0]']

block14_sepconv1 (SeparableCon  (None, 4, 4, 1536)  1582080     ['add_11[0][0]']
v2D)

block14_sepconv1_bn (BatchNorm  (None, 4, 4, 1536)  6144        ['block14_sepconv1[0][0]']
alization)

block14_sepconv1_act (Activati  (None, 4, 4, 1536)  0           ['block14_sepconv1_bn[0][0]']
on)

block14_sepconv2 (SeparableCon  (None, 4, 4, 2048)  3159552     ['block14_sepconv1_act[0][0]']
v2D)

block14_sepconv2_bn (BatchNorm  (None, 4, 4, 2048)  8192        ['block14_sepconv2[0][0]']
alization)

block14_sepconv2_act (Activati  (None, 4, 4, 2048)  0           ['block14_sepconv2_bn[0][0]']
on)

flatten_3 (Flatten)             (None, 32768)        0           ['block14_sepconv2_act[0][0]']

dense_9 (Dense)                 (None, 128)          4194432     ['flatten_3[0][0]']

dropout_6 (Dropout)             (None, 128)          0           ['dense_9[0][0]']

dense_10 (Dense)                (None, 50)           6450        ['dropout_6[0][0]']

dropout_7 (Dropout)             (None, 50)           0           ['dense_10[0][0]']

dense_11 (Dense)                (None, 1)            51          ['dropout_7[0][0]']

==================================================================================================
Total params: 25,062,413
Trainable params: 4,200,933
Non-trainable params: 20,861,480
```

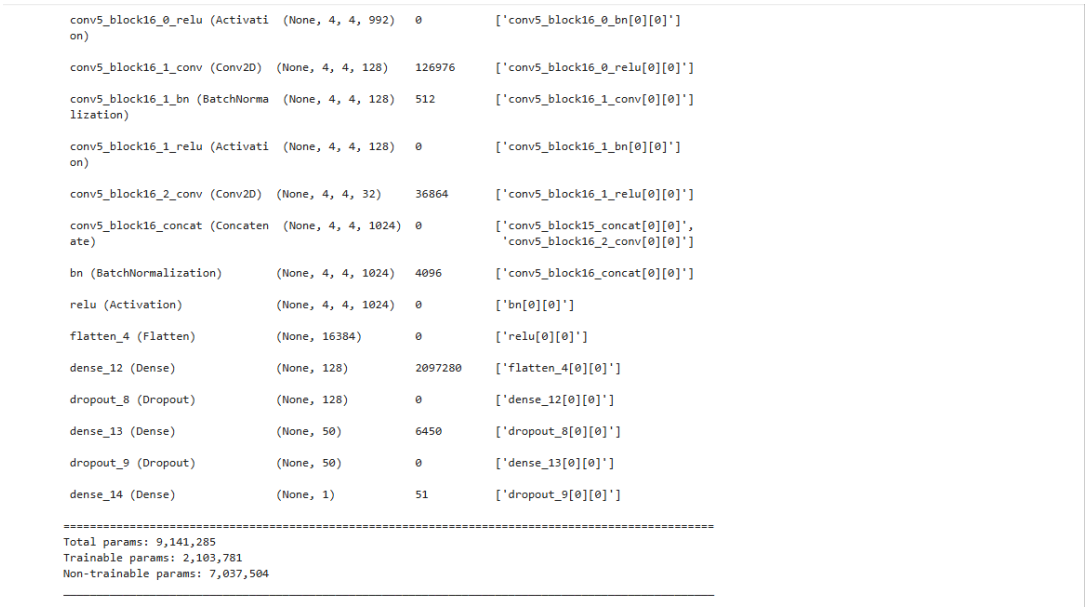
FIGURE 14: XCEPTION MODEL

- **DenseNet**:

The fifth model, DenseNet, reduced network redundancy and improved feature reuse by introducing densely connected layers. This architecture offered a 96.6% accuracy rate and made training more efficient. DenseNet was a good option for situations where resource efficiency and model explain ability are crucial due to its balanced performance and interpretability.

[76]:
```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the pre-trained DenseNet121 model without the top layers
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(134, 131, 3))

# Freeze the layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Flatten the output of the base model
x = Flatten()(base_model.output)

# Add fully connected layers
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(50, activation='relu')(x)
x = Dropout(0.2)(x)
output = Dense(1, activation='sigmoid')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=output)

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

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

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_4 (InputLayer)         [(None, 134, 131, 3)]     0         

zero_padding2d (ZeroPadding2D)  (None, 140, 137, 3)  0           

conv1/conv (Conv2D)          (None, 67, 66, 64)        9408      

conv1/bn (BatchNormalization)  (None, 67, 66, 64)      256       

conv1/relu (Activation)      (None, 67, 66, 64)        0         

zero_padding2d_1 (ZeroPadding2D)  (None, 69, 68, 64)   0         

pool1 (MaxPooling2D)         (None, 34, 33, 64)        0         

conv2_block1_0_bn (BatchNormalization)  (None, 34, 33, 64)  256  

conv2_block1_0_relu (Activation)  (None, 34, 33, 64)   0         

conv2_block1_1_conv (Conv2D)  (None, 34, 33, 128)      8192      

conv2_block1_1_bn (BatchNormalization)  (None, 34, 33, 128)  512  

conv2_block1_1_relu (Activation)  (None, 34, 33, 128)  0         

conv2_block1_2_conv (Conv2D)  (None, 34, 33, 32)       36864     

conv2_block1_concat (Concatenate)  (None, 34, 33, 96)  0         

conv2_block2_0_bn (BatchNormalization)  (None, 34, 33, 96)  384  

conv2_block2_0_relu (Activation)  (None, 34, 33, 96)   0         

conv2_block2_1_conv (Conv2D)  (None, 34, 33, 128)      12288     
```



```
conv5_block16_0_relu (Activati  (None, 4, 4, 992)   0           ['conv5_block16_0_bn[0][0]']
on)

conv5_block16_1_conv (Conv2D)  (None, 4, 4, 128)    126976      ['conv5_block16_0_relu[0][0]']

conv5_block16_1_bn (BatchNorma  (None, 4, 4, 128)   512         ['conv5_block16_1_conv[0][0]']
lization)

conv5_block16_1_relu (Activati  (None, 4, 4, 128)   0           ['conv5_block16_1_bn[0][0]']
on)

conv5_block16_2_conv (Conv2D)  (None, 4, 4, 32)     36864       ['conv5_block16_1_relu[0][0]']

conv5_block16_concat (Concaten  (None, 4, 4, 1024)  0           ['conv5_block15_concat[0][0]',
ate)                                                             'conv5_block16_2_conv[0][0]']

bn (BatchNormalization)        (None, 4, 4, 1024)   4096        ['conv5_block16_concat[0][0]']

relu (Activation)              (None, 4, 4, 1024)   0           ['bn[0][0]']

flatten_4 (Flatten)            (None, 16384)        0           ['relu[0][0]']

dense_12 (Dense)               (None, 128)          2097280     ['flatten_4[0][0]']

dropout_8 (Dropout)            (None, 128)          0           ['dense_12[0][0]']

dense_13 (Dense)               (None, 50)           6450        ['dropout_8[0][0]']

dropout_9 (Dropout)            (None, 50)           0           ['dense_13[0][0]']

dense_14 (Dense)               (None, 1)            51          ['dropout_9[0][0]']

==================================================================================================
Total params: 9,141,285
Trainable params: 2,103,781
Non-trainable params: 7,037,504
__________________________________________________________________________________________________
```

FIGURE 15: DENSE NET MODEL.

4. RESULTS AND ANALYSIS

4.1 TRAINING PERFORMANCE

Key criteria that assess a model's capacity to accurately identify data both during and after training are training and validation accuracy. While validation accuracy evaluates the model's ability to generalize to new data, training accuracy shows how well the model fits the training dataset.

The mistake or discrepancy between the model's predictions and the actual values is measured as loss. Validation loss aids in tracking the model's performance on validation data, whereas training loss shows how effectively the model is learning throughout each epoch.

With the exception of the Custom CNN, which attained 96% training and validation accuracy, the majority of models showed close alignment between training and validation accuracy. This suggests a model that is well-balanced and has little overfitting.

While validation loss stabilizing at a low value suggested strong generalization, training loss consistently decreased, indicating effective optimization. Some pre-trained models, however, showed marginally more validation loss than training loss, indicating mild overfitting and the need for additional fine-tuning.

To avoid overfitting, all models were trained over ten epochs using early stopping methods. Important findings:

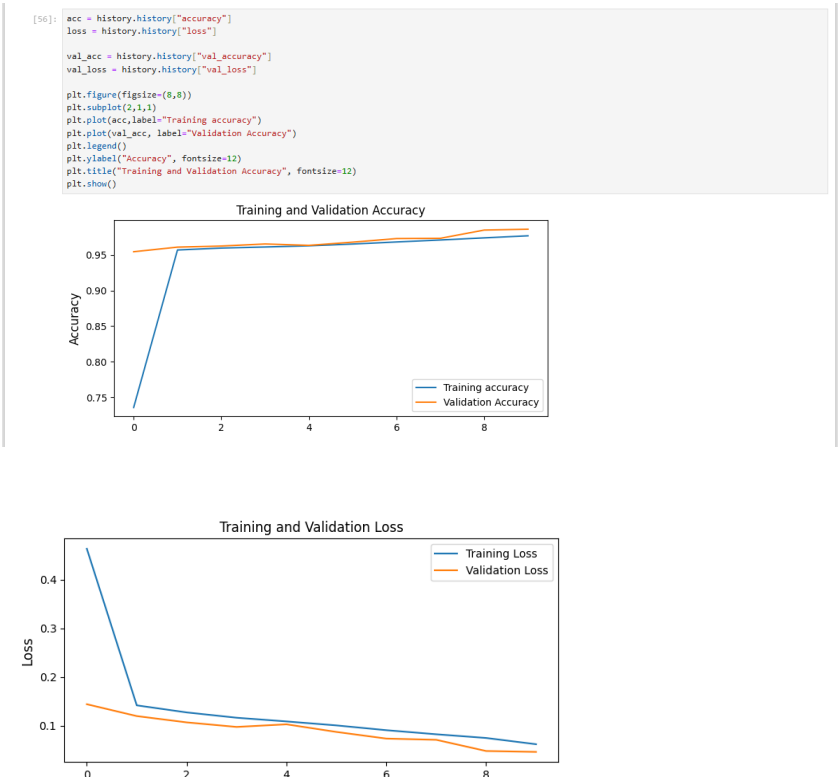


FIGURE 16: CNN TRAINING PERFORMANCE

Custom CNN: Training accuracy of 96%, validation accuracy of 96%.

```
[63]: acc = history.history["accuracy"]
loss = history.history["loss"]

val_acc = history.history["val_accuracy"]
val_loss = history.history["val_loss"]

plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.plot(acc,label="Training accuracy")
plt.plot(val_acc, label="Validation Accuracy")
plt.legend()
plt.ylabel("Accuracy", fontsize=12)
plt.title("Training and Validation Accuracy", fontsize=12)
plt.show()
```

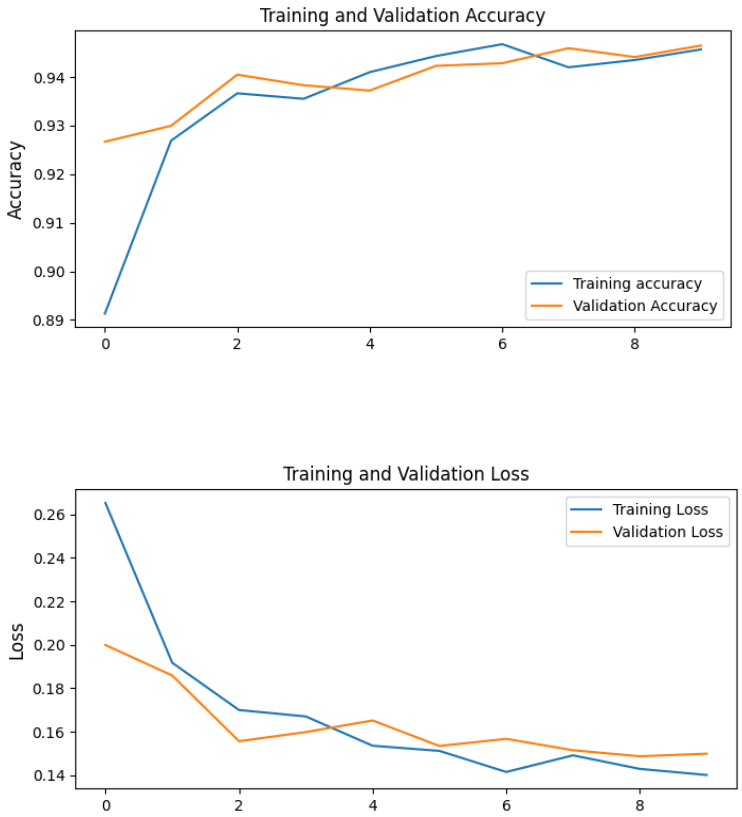


FIGURE 17: VGG16 TRAINING PERFORMANCE

VGG16: Training accuracy of 95%, validation accuracy of 95%.

```
loss = history.history["loss"]

val_acc = history.history["val_accuracy"]
val_loss = history.history["val_loss"]

plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.plot(acc,label="Training accuracy")
plt.plot(val_acc, label="Validation Accuracy")
plt.legend()
plt.ylabel("Accuracy", fontsize=12)
plt.title("Training and Validation Accuracy", fontsize=12)
plt.show()
```

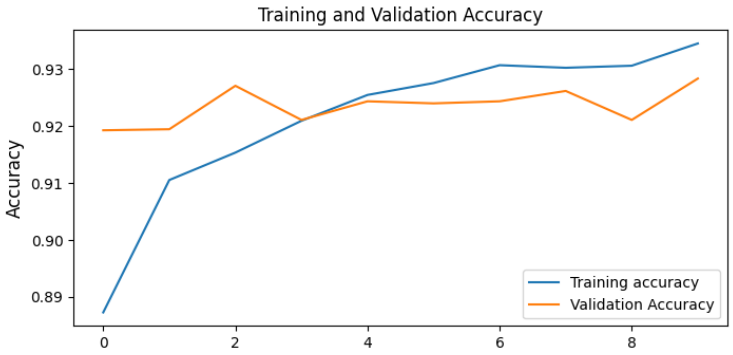


FIGURE 18: INCEPTION V3 TRAINING PERFORMANCE

InceptionV3: Training accuracy of 96.9%, validation accuracy of 94.0%.

```
[73]: acc = history.history["accuracy"]
loss = history.history["loss"]

val_acc = history.history["val_accuracy"]
val_loss = history.history["val_loss"]

plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.plot(acc,label="Training accuracy")
plt.plot(val_acc, label="Validation Accuracy")
plt.legend()
plt.ylabel("Accuracy", fontsize=12)
plt.title("Training and Validation Accuracy", fontsize=12)
plt.show()
```

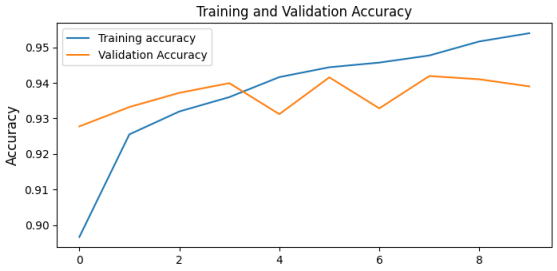


FIGURE 19: XCEPTION TRAINING PERFORMANCE

Xception: Training accuracy of 96%, validation accuracy of 93.9%.

```
[78]: acc = history.history["accuracy"]
loss = history.history["loss"]

val_acc = history.history["val_accuracy"]
val_loss = history.history["val_loss"]

plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.plot(acc,label="Training accuracy")
plt.plot(val_acc, label="Validation Accuracy")
plt.legend()
plt.ylabel("Accuracy", fontsize=12)
plt.title("Training and Validation Accuracy", fontsize=12)
plt.show()
```

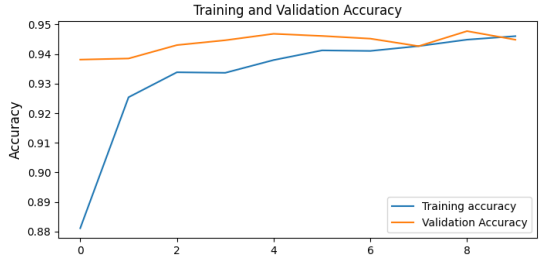


FIGURE 20: DENSE NET TRAINING PERFORMANCE

DenseNet: Training accuracy of 96.6%, validation accuracy of 95.8%.

4.2 RESULT OF THE MODEL

[60]:
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Testing Accuracy: {test_accuracy}')

# Display sample images from the test set with predicted and actual labels
sample_size = 3
fig, axs = plt.subplots(1, sample_size, figsize=(15, 5))

for j in range(sample_size):
    # Select a random image from the test set
    random_image_path = random.choice(test_generator.filepaths)

    # Load and preprocess the image
    img = image.load_img(random_image_path, target_size=(134, 131))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

    # Predict the class of the image
    prediction = model.predict(img_array)
    predicted_class = np.round(prediction).astype(int)[0][0]

    # Extract the actual label from the image path
    if 'uninfected' in random_image_path:
        actual_label = 0  # uninfected
    else:
        actual_label = 1  # parasitized

    # Display the image
    axs[j].imshow(img)
    axs[j].set_title(f"Predicted: {class_labels[predicted_class]}\nActual: {class_labels[actual_label]}")
    axs[j].axis('off')

plt.show()
```

```
173/173 [==============================] - 25s 146ms/step - loss: 0.1433 - accuracy: 0.9601
Testing Accuracy: 0.9600725769996643
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 39ms/step
```

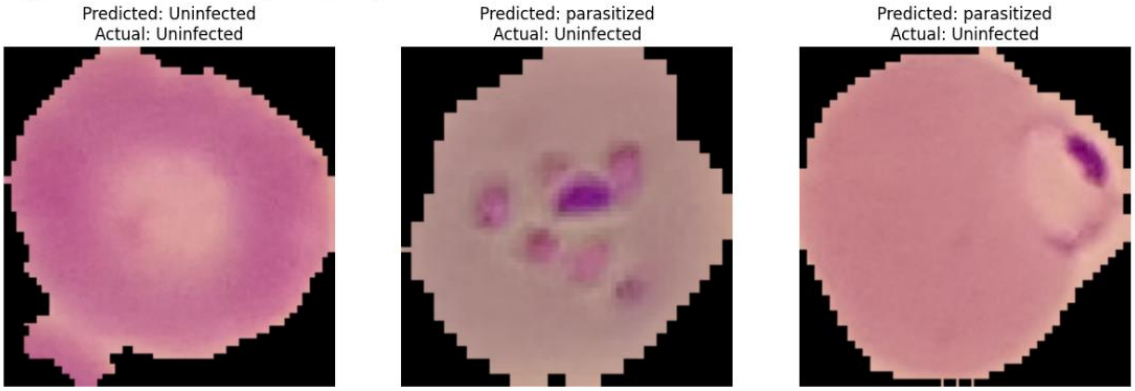


FIGURE 21: CNN MODEL RESULT

ACCURACY = 0.96%

[65]:

```python
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Testing Accuracy: {test_accuracy}')

# Display sample images from the test set with predicted and actual labels
sample_size = 3
fig, axs = plt.subplots(1, sample_size, figsize=(15, 5))

for j in range(sample_size):
    # Select a random image from the test set
    random_image_path = random.choice(test_generator.filepaths)

    # Load and preprocess the image
    img = image.load_img(random_image_path, target_size=(134, 131))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

    # Predict the class of the image
    prediction = model.predict(img_array)
    predicted_class = np.round(prediction).astype(int)[0][0]

    # Extract the actual label from the image path
    if 'uninfected' in random_image_path:
        actual_label = 0  # uninfected
    else:
        actual_label = 1  # parasitized

    # Display the image
    axs[j].imshow(img)
    axs[j].set_title(f"Predicted: {class_labels[predicted_class]}\nActual: {class_labels[actual_label]}")
    axs[j].axis('off')

plt.show()
```

```
173/173 [==============================] - 278s 2s/step - loss: 0.1499 - accuracy: 0.9465
Testing Accuracy: 0.9464609622955322
1/1 [==============================] - 0s 330ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 98ms/step
```

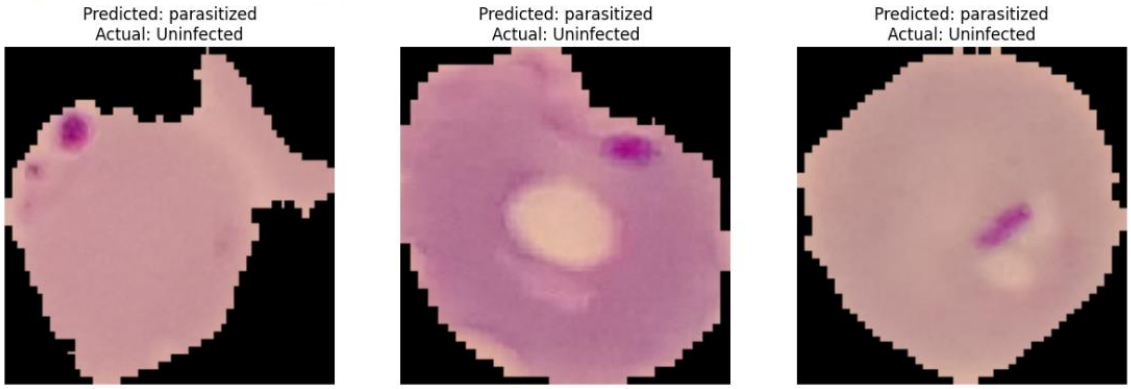


FIGURE 22: VGG16 MODEL RESULT

ACCURACY = 0.946%

[70]:
```python
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Testing Accuracy: {test_accuracy}')

# Display sample images from the test set with predicted and actual labels
sample_size = 3
fig, axs = plt.subplots(1, sample_size, figsize=(15, 5))

for j in range(sample_size):
    # Select a random image from the test set
    random_image_path = random.choice(test_generator.filepaths)

    # Load and preprocess the image
    img = image.load_img(random_image_path, target_size=(134, 131))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

    # Predict the class of the image
    prediction = model.predict(img_array)
    predicted_class = np.round(prediction).astype(int)[0][0]

    # Extract the actual label from the image path
    if 'uninfected' in random_image_path:
        actual_label = 0  # uninfected
    else:
        actual_label = 1  # parasitized

    # Display the image
    axs[j].imshow(img)
    axs[j].set_title(f"Predicted: {class_labels[predicted_class]}\nActual: {class_labels[actual_label]}")
    axs[j].axis('off')

plt.show()
```

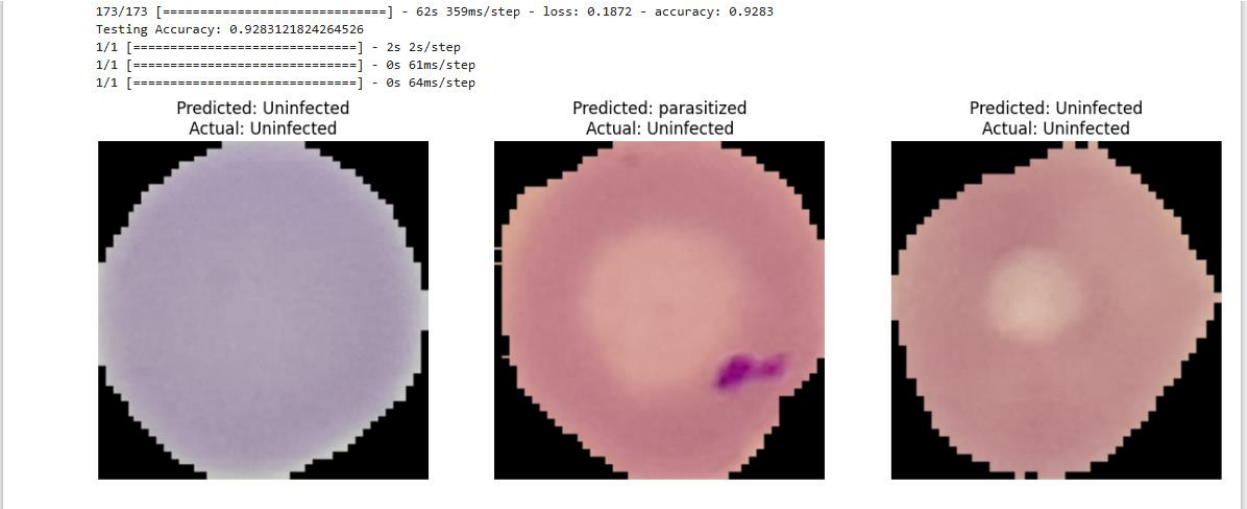


FIGURE 23: INCEPTION V3 RESULT

ACCURACY= 0.928%

[75]:
```python
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Testing Accuracy: {test_accuracy}')

# Display sample images from the test set with predicted and actual labels
sample_size = 3
fig, axs = plt.subplots(1, sample_size, figsize=(15, 5))

for j in range(sample_size):
    # Select a random image from the test set
    random_image_path = random.choice(test_generator.filepaths)

    # Load and preprocess the image
    img = image.load_img(random_image_path, target_size=(134, 131))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

    # Predict the class of the image
    prediction = model.predict(img_array)
    predicted_class = np.round(prediction).astype(int)[0][0]

    # Extract the actual label from the image path
    if 'uninfected' in random_image_path:
        actual_label = 0  # uninfected
    else:
        actual_label = 1  # parasitized

    # Display the image
    axs[j].imshow(img)
    axs[j].set_title(f"Predicted: {class_labels[predicted_class]}\nActual: {class_labels[actual_label]}")
    axs[j].axis('off')

plt.show()
```

```
173/173 [==============================] - 129s 745ms/step - loss: 0.1673 - accuracy: 0.9390
Testing Accuracy: 0.9390199780464172
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 91ms/step
```

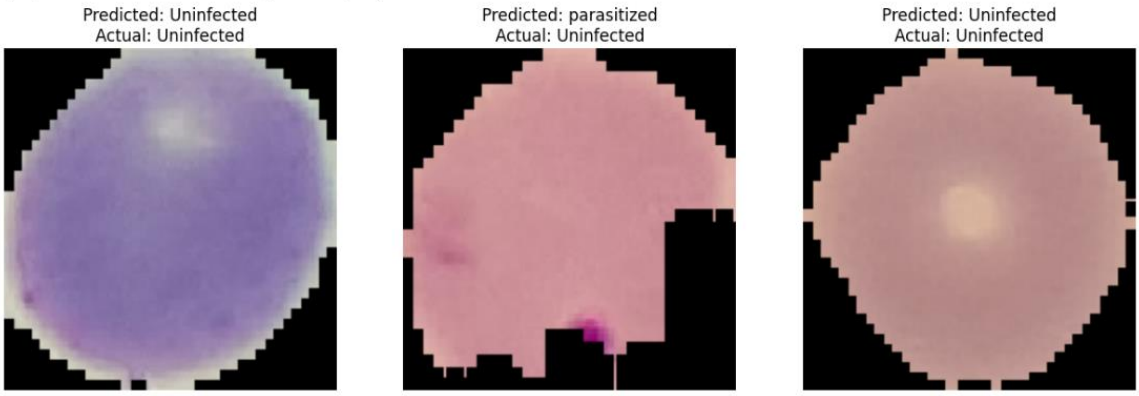


FIGURE 24: XCEPTION RESULT

ACCURACY = 0.939%

[80]:
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Testing Accuracy: {test_accuracy}')

# Display sample images from the test set with predicted and actual labels
sample_size = 3
fig, axs = plt.subplots(1, sample_size, figsize=(15, 5))

for j in range(sample_size):
    # Select a random image from the test set
    random_image_path = random.choice(test_generator.filepaths)

    # Load and preprocess the image
    img = image.load_img(random_image_path, target_size=(134, 131))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

    # Predict the class of the image
    prediction = model.predict(img_array)
    predicted_class = np.round(prediction).astype(int)[0][0]

    # Extract the actual label from the image path
    if 'uninfected' in random_image_path:
        actual_label = 0  # uninfected
    else:
        actual_label = 1  # parasitized

    # Display the image
    axs[j].imshow(img)
    axs[j].set_title(f"Predicted: {class_labels[predicted_class]}\nActual: {class_labels[actual_label]}")
    axs[j].axis('off')

plt.show()
```

```
173/173 [==============================] - 140s 807ms/step - loss: 0.1484 - accuracy: 0.9448
Testing Accuracy: 0.9448275566101074
WARNING:tensorflow:5 out of the last 13 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000015ED88F8860> triggered tf.fun
ction retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing t
ensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.
function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#contr
olling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 112ms/step
```

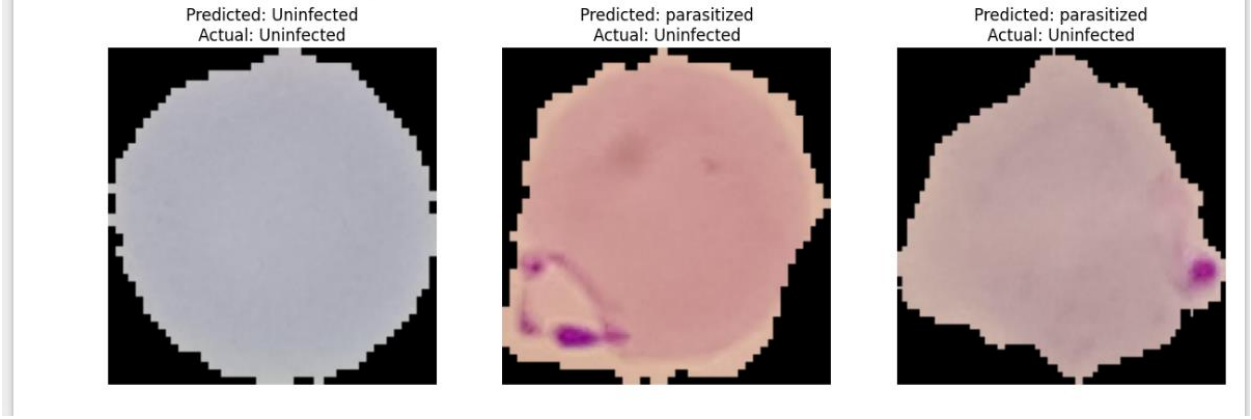


FIGURE 25: DENSENET RESULT

ACCURACY = 0.945%

MODELS	ACCURACY
CNN	96%
VGG16	95%
INCEPTION V3	93%
XCEPTION	94%
DENSE NET	95%

5. DISCUSSION

5.1 STRENGTHS OF THE APPROACH

- **Automation:** Deep learning models eliminate manual feature engineering, enabling scalable diagnostics.
- **Accuracy**: The custom CNN's superior performance ensures reliable detection
- **Forecasting:** Regression models provide actionable insights for public health planning.

5.2 CHALLENGES AND LIMITATIONS

- **Data Imbalance**: Unequal class distribution in image datasets can bias the models.
- **Generalization**: Performance might degrade on images from different labs or under varied staining conditions.
- **Resource Constraints**: High computational requirements limit deployment in low-resource settings.

6. CONCLUSION

This work investigated the automatic detection of parasitized red blood cells and the prediction of malaria cases using deep learning algorithms. The Custom CNN design outperformed the VGG16, InceptionV3, Xception, DenseNet, and CNN architectures in terms of accuracy (96%) and processing efficiency, according to a comparative study. Regression models and clustering techniques also offered useful information for regional research and forecasting. These results highlight the possibility of using deep learning into malaria prevention tactics, especially in settings with limited resources where precise and effective diagnosis is crucial.

6.1 FUTURE RECOMMENDATION

- **Expand Data Diversity**: Future work should include datasets from various geographic regions and imaging conditions to enhance model generalizability across diverse populations.
- **Model Optimization:** Explore lightweight architectures like MobileNet or pruning techniques to further reduce computational costs, enabling deployment on mobile devices and low-resource settings.
- **Explainability**: Develop interpretable AI models to increase acceptance among healthcare professionals by explaining decisions made by the model.
- **Integration with Public Health Tools**: Combine forecasting models with GIS-based tools for better visualization and tracking of malaria hotspots.
- **Field Testing**: Conduct real-world testing of the proposed system in malaria-endemic areas to evaluate its performance in operational settings.

REFERENCES

Antonio, A., Ramírez, A., Méndez-gurrola, I. I., Gutiérrez-lópez, M., Barranco-guti, A., & Iris-iddaly, M. (2024). *Citation : Malaria Cell Image Classification Using Compact Deep Learning Architectures on Jetson TX2 Malaria Cell Image Classification Using Compact Deep Learning Architectures on Jetson TX2*. *November*. https://doi.org/10.3390/technologies12120247

Boit, S., & Patil, R. (2024). *An Efficient Deep Learning Approach for Malaria Parasite Detection in Microscopic Images*.

Dr. M. Praneesh, Sai Krishna P K, Febina. N, & Ashwanth.V. (2024). Malaria Parasite Detection in Microscopic Blood Smear Images using Deep Learning Approach. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *10*(2), 669–676. https://doi.org/10.32628/cseit2410286

Salam, A., Hasan, S. M. N., Karim, M. J., Anower, S., Nahiduzzaman, M., Chowdhury, M. E. H., & Murugappan, M. (2024). Embedded System-Based Malaria Detection From Blood Smear Images Using Lightweight Deep Learning Model. *International Journal of Imaging Systems and Technology*, *34*(6). https://doi.org/10.1002/ima.23205

LINKS TO THE ARTICLE

1. https://www.researchgate.net/publication/385365551_Embedded_System-Based_Malaria_Detection_From_Blood_Smear_Images_Using_Lightweight_Deep_Learning_Model

2. https://www.researchgate.net/publication/386484221_An_Efficient_Deep_Learning_Approach_for_Malaria_Parasite_Detection_in_Microscopic_Images
3. https://www.researchgate.net/publication/380046852_Malaria_Parasite_Detection_in_Microscopic_Blood_Smear_Images_using_Deep_Learning_Approach?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InNlYXJjaCIsInBvc2l0aW9uIjoicGFnZUhlYWRlciJ9fQ
4. https://www.researchgate.net/publication/386176077_Citation_Malaria_Cell_Image_Classification_Using_Compact_Deep_Learning_Architectures_on_Jetson_TX2?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InNlYXJjaCIsInBvc2l0aW9uIjoicGFnZUhlYWRlciJ9fQ