Comprehensive Report on MOSQUITO-NET Implementation and Understanding

Prepared by:
📌 *Revanth Puvaneswaren*
📌 *Imthias Abubakkar*

🏥 Introduction

This report outlines the implementation of the **MOSQUITO-NET** system, inspired by the research paper:
📝 *"MOSQUITO-NET: A Deep Learning-Based CADx System for Malaria Diagnosis Along with Model Interpretation Using Grad-CAM and Class Activation Maps"*.
The objective of this notebook is to develop a **lightweight deep learning model** for **accurate malaria detection** while incorporating interpretability features such as **Grad-CAM** for visual insights.

📚 Research Paper Overview

The **MOSQUITO-NET framework** employs **Convolutional Neural Networks (CNNs)** for malaria detection, integrating **Grad-CAM** for **hotspot visualizations** that highlight areas influential to predictions. This interpretability enhances **diagnostic trust** and facilitates deployment in **resource-constrained areas**.

📂 Notebook Workflow

1. Downloading and Extracting the Data

📥 The dataset was obtained from the **National Library of Medicine** (Download Link).
📌 It contains labeled blood smear images of infected and uninfected cells, categorized into:

- ✅ **Infected cells**
- ❌ **Uninfected cells**

🏗️ Model Architecture

The **MOSQUITO-NET** architecture is a **lightweight CNN** for efficient malaria cell classification. Below is a detailed breakdown:

🔹 1. Input Layer

- **Input Dimensions**: `75x75` images with 3 channels `(RGB format)`.
- **Purpose**: Standardizes the input shape for computations.

🔹 2. Convolutional Layers

🔍 These layers extract spatial features such as **edges, textures, and patterns**:

- **Conv2D Layers:**
  - 🔹 **First Layer:** `16 filters, kernel size 5x5, stride 1, padding = 'same'`
  - 🔹 **Second Layer:** `32 filters, same configuration`
  - 🔹 **Third Layer:** `64 filters, kernel size 5x5, padding = 'same' (named "rr" for Grad-CAM integration)`
- **Purpose:** Layers learn progressively complex features.

🔹 3. Batch Normalization

📌 **Function:** Normalizes activations to stabilize learning & improve convergence speed.

🔹 4. ReLU Activation

✅ **Non-Linearity:** Helps the network learn complex patterns.

🔹 5. Max Pooling Layers

- **Kernel Size:** 2x2, stride of 2.
- **Purpose:** Reduces spatial dimensions while **retaining essential features**.

🔹 6. Flatten Layer

🔄 Converts multi-dimensional outputs into a one-dimensional array for dense layers.

🔹 7. Fully Connected (Dense) Layers

- 🔹 **First Dense Layer:** `512 neurons` with **ReLU activation**.
- 🔹 **Dropout:** `0.2` (Prevents overfitting).
- 🔹 **Second Dense Layer:** `128 neurons`.
- 🔹 **Output Layer:** `2 neurons with Sigmoid activation` for binary classification.

🔹 8. Sigmoid Activation for Classification

📌 Produces probability scores for **each class (infected/uninfected)**.

📌 **MODEL STRUCTURE**

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 120, 120, 16)	1,216
batch_normalization (BatchNormalization)	(None, 120, 120, 16)	64
re_lu (ReLU)	(None, 120, 120, 16)	0
max_pooling2d (MaxPooling2D)	(None, 60, 60, 16)	0
conv2d_1 (Conv2D)	(None, 60, 60, 32)	12,832
batch_normalization_1 (BatchNormalization)	(None, 60, 60, 32)	128
re_lu_1 (ReLU)	(None, 60, 60, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
rr (Conv2D)	(None, 30, 30, 64)	51,264
batch_normalization_2 (BatchNormalization)	(None, 30, 30, 64)	256
re_lu_2 (ReLU)	(None, 30, 30, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 64)	0
flatten (Flatten)	(None, 14400)	0
dense (Dense)	(None, 512)	7,373,312
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65,664
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	258

Total params: 7,504,994 (28.63 MB)
Trainable params: 7,504,770 (28.63 MB)
Non-trainable params: 224 (896.00 B)

:

🎯 Training and Validation

✅ **Trained using TensorFlow/Keras**, with **accuracy & loss** monitored.
📈 Training and validation accuracy **plots** provided insights into **overfitting & generalization**.

📊 Training Visualization

📌 Accuracy comparisons were visualized using **Matplotlib**, generating interpretable **performance graphs**.

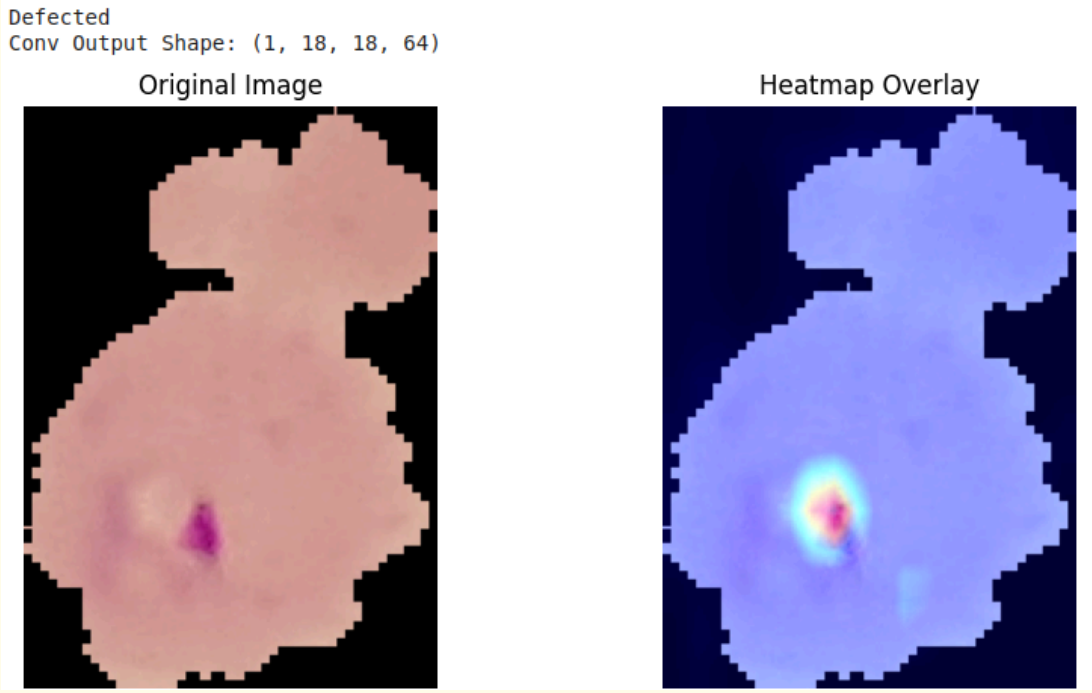
🧪 Predictions and Interpretability

Making Predictions

The trained model was tested on **unseen validation data**. Below is a **visualization**:

```
fig, axes = plt.subplots(5,5, figsize=(15,15))
for i, ax in enumerate(axes.flat):
    r = np.random.randint(X_val.shape[0])
    ax.imshow(X_val[r])
    ax.grid(False)
    ax.axis('off')
    ax.set_title('Original: {} Predicted: {}'.format(np.argmax(y_val[r]),
np.argmax(model.predict(X_val[r].reshape(1, 75, 75, 3)))))
```

📌 **OUTPUT IMAGE:**



📊 Confusion Matrix

📌 **Purpose**: Evaluates model performance by analyzing: ✅ **True Positives (TP)**, ❌ **False Positives (FP)**, ✅ **True Negatives (TN)**, ❌ **False Negatives (FN)**.

- 🔹 Helps **diagnose classification errors** & improve model refinement.

🎓 Learnings

🛠️ Technical Skills

✅ **Model Design & Implementation:** Lightweight CNN, Batch Normalization, Activation Functions.
✅ **Grad-CAM for Interpretability:** Understanding **hotspot visualizations**.
✅ **Data Augmentation:** Using `ImageDataGenerator` to improve generalization.
✅ **Visualization:** Creating **confusion matrices & performance plots**.

🌍 Real-World Applications

✅ **Resource Efficiency:** AI tailored for **edge devices** in **remote regions**.
✅ **Scalability:** Optimizing **performance vs. computational constraints**.

🧩 Problem-Solving

✅ **Data Challenges:** Handling **large datasets**, **preprocessing techniques**.
✅ **Error Analysis:** **Confusion matrix insights** for **classification improvements**.

📢 Big-Picture Insights

✅ **AI in Healthcare:** Transformative potential of AI in **global health diagnostics**.
✅ **Model Transparency:** Importance of **interpretability & trust** in AI models.

✅ Conclusion

📌 The **MOSQUITO-NET** model was successfully implemented, demonstrating **malaria detection** capabilities using **deep learning**.
📈 **Future Work:**

- **Expand the dataset**.
- **Refine model architecture**.
- **Real-world deployment** for field testing.