ViT-alia

(Malaria Detection using Vision Transformers)

A Project Report

submitted by

ROHAN G

R SAI CHARISH,

T PRATYEK

(CS22B1093, CS22B1095, CS22B1096)

in partial fulfilment of requirements

for the course project work of

MACHINE LEARNING

Department of Computer Science and Engineering

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,

DESIGN AND MANUFACTURING KANCHEEPURAM

Nov 2024

DECLARATION OF ORIGINALITY

We, **Rohan G, R Sai Charish, T Pratyek**, with Roll No: **CS22B1093, CS22B1095, CS22B1096** hereby declare that the material presented in the Project Report titled **ViT-alia (Malaria Detection using Vision Transformers)** represents original work carried out by me in the **Department of Computer Science and Engineering** at the Indian Insti- tute of Information Technology, Design and Manufacturing, Kancheepuram.

With our signature, We certify that:

- We have not manipulated any of the data or results.
- We have not committed any plagiarism of intellectual property. We have clearly indicated and referenced the contributions of others.
- We have explicitly acknowledged all collaborative research and discussions.
- We have understood that any false claim will result in severe disciplinary action.
- We have understood that the work may be screened for any form of academic misconduct.

Rohan G, R Sai Charish, T Pratyek

Place: Chennai

Date: 17.11.2024

CERTIFICATE

This is to certify that the report titled **ViT-alia (Malaria Detection using Vision Transformers)**, submitted by **Rohan G, R Sai Charish, T Pratyek (CS22B1093, CS22B1095, CS22B1096)**, to the Indian Institute of Information Technology, Design and Manufacturing Kancheepuram, for the course project work of degree of **MACHINE LEARNING** is a bona fide record of the work done by him/her under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Rahul Raman
Project Guide
Assistant Professor
Department of Computer Science and Engineering
IIITDM Kancheepuram, 600 127

Place: Chennai

Date: 17.11.2024

ABSTRACT

Medical imaging plays a critical role in the diagnosis and treatment of various diseases. However, the complexity of medical images and the subtlety of certain pathologies pose challenges for accurate and efficient diagnosis. Traditional convolutional neural networks (CNNs) have made significant advancements in this domain, but their limitations in capturing global context and long-range dependencies have prompted the exploration of alternative approaches. This research focuses on enhancing medical imaging and diagnostic accuracy through the application of Vision Transformers (ViTs).

Vision Transformers, with their self-attention mechanisms, offer a powerful means to model global relationships within images, making them particularly suitable for the nuanced and often intricate patterns present in medical data. In this study, we propose a novel architecture that integrates ViTs with domain-specific adaptations tailored for medical imaging.

We can apply this model across multiple imaging modalities, including MRI, CT, and X-ray, and evaluate its performance in detecting and classifying disease of Malaria.

This project investigates the use of Vision Transformers (ViT's) to enhance medical imaging and diagnosis by leveraging their ability to capture global image context. A novel ViT-based architecture tailored for medical modalities like MRI and CT scans is proposed, aiming to improve accuracy and efficiency in Malaria detection.

KEYWORDS:

Vision Transformer; Convolutional Neural Network; Medical Imaging; Malaria Detection.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivation for Malaria Detection

Malaria remains one of the most prevalent and deadly infectious diseases in the world, particularly in developing regions like sub-Saharan Africa, Southeast Asia, and parts of South America. According to the World Health Organization (WHO), malaria accounted for an estimated 241 million cases and over 627,000 deaths in 2021 alone. The impact is especially severe on children under the age of five and pregnant women, making early and accurate diagnosis a critical factor in reducing mortality rates.

Traditional methods for diagnosing malaria involve manual examination of Giemsa-stained blood smears under a microscope. This process, although effective, is labor-intensive, time-consuming, and requires highly skilled technicians. In many regions where malaria is most prevalent, access to such skilled personnel and medical facilities is limited. As a result, delays in diagnosis can lead to worsening patient outcomes, including severe complications and death.

The need for a more scalable, rapid, and accurate solution is what motivated us to focus on automating malaria diagnosis using cutting-edge machine learning techniques. By leveraging deep learning models, we can provide faster, reliable, and efficient diagnostic tools that can be deployed in remote and resource-limited areas. Our project aims to bridge the gap between the need for early detection and the availability of diagnostic resources, ultimately contributing to global efforts in controlling and eradicating malaria.

1.2 Importance of Medical Imaging

Medical imaging plays a pivotal role in modern healthcare by enabling non-invasive diagnosis of diseases through the analysis of visual data such as X-rays, MRIs, and microscopy slides. In the context of malaria diagnosis, thin blood smear microscopy is considered the gold standard due to its high sensitivity and specificity. However, the reliance on skilled personnel and the potential for human error make manual examination challenging, particularly in high-volume or resource-limited settings.

By utilizing medical imaging combined with artificial intelligence (AI) techniques, we can automate the detection of malaria parasites in blood smear images. Deep learning models, especially Vision Transformers, excel in analyzing complex image data by capturing intricate patterns and features that might be missed by the human eye. These models are capable of learning both local features (e.g., shapes, textures) and global contexts (e.g., overall cell structure), making them highly effective in identifying infected cells with high precision.

In our project, we chose medical imaging as the foundation because it allows for accurate, non-invasive diagnosis while significantly reducing the time and resources required for analysis. Automating the diagnostic process with AI models not only enhances efficiency but also ensures consistent accuracy, which is crucial for large-scale screenings in malaria-endemic regions. This approach aligns with our goal of leveraging technology to improve healthcare outcomes and provide scalable solutions where they are needed most.

1.3 Scope of the Project

In this project, we aim to develop a deep learning-based system for automated malaria diagnosis using Vision Transformers. The project focuses on leveraging medical imaging to enhance the accuracy and efficiency of malaria diagnosis in resource-limited settings.

Table 1.1: Sample Table of Blood Smear Analysis

Sample ID	Parasite Count	Diagnosis	Confidence
101	120	Positive	95%
102	0	Negative	98%
103	78	Positive	90%
104	0	Negative	97%
105	150	Positive	92%

1.4 Description of Malaria Detection Workflow

The workflow diagram in Figure 1.1 outlines the entire process of using a Vision Transformer (ViT) model for automating malaria diagnosis. Below is a detailed explanation of the steps involved:

1. **User Initiates the Process**:
   - The process begins when a user initiates the malaria detection task.
2. **System Setup**:
   - The system initializes by loading the Vision Transformer (ViT) model.
   - The required feature extractor is loaded to preprocess the images.
3. **Dataset Loading**:
   - The dataset, which consists of blood smear images, is loaded into the system.
   - The loaded dataset is then passed through various stages for preprocessing.
4. **Feature Extraction and Transformation**:
   - The feature extractor processes the raw images, converting them into a format suitable for the model.
   - Data transformations such as resizing, normalization, and augmentation are applied to improve the model's performance.
5. **Data Splitting**:
   - The preprocessed dataset is split into training and test sets.
   - This ensures that the model is trained on one subset of data and evaluated on a separate, unseen subset.
6. **Model Training**:
   - The training process begins with a loop that iterates over a specified number of epochs.

- The model is trained using the training set, and the weights are updated based on the loss function.

7. **Validation and Early Stopping**:
   - The validation loss is monitored during each epoch.
   - If the validation loss improves, the current model is saved as the best model.
   - If the validation loss does not improve for a specified number of epochs (patience), the early stopping mechanism is triggered to prevent overfitting.
8. **Model Evaluation**:
   - Once training is complete (or early stopping is triggered), the best saved model is loaded.
   - The model is evaluated on the test set to generate predictions.
9. **Metrics Calculation**:
   - The system calculates various evaluation metrics, including accuracy, precision, recall, and F1-score, to assess the model's performance.
10. **Results Display**:
    - The final results, including metrics and predictions, are displayed to the user for review.

Key Components Involved

- **User**: The person initiating the malaria detection process.
- **System**: The underlying infrastructure handling the loading, training, and evaluation of the model.
- **Model**: The Vision Transformer (ViT) used for malaria detection.
- **Feature Extractor**: A tool used to preprocess raw data into a format suitable for model training.
- **Dataset**: Collection of labeled blood smear images used for training and testing.
- **Transform**: Data augmentation and transformation techniques to enhance model robustness.
- **Early Stopping**: A mechanism to prevent overfitting by stopping training when the model performance stagnates.
- **Metrics**: Quantitative measures (accuracy, precision, recall, F1-score) to evaluate model effectiveness.

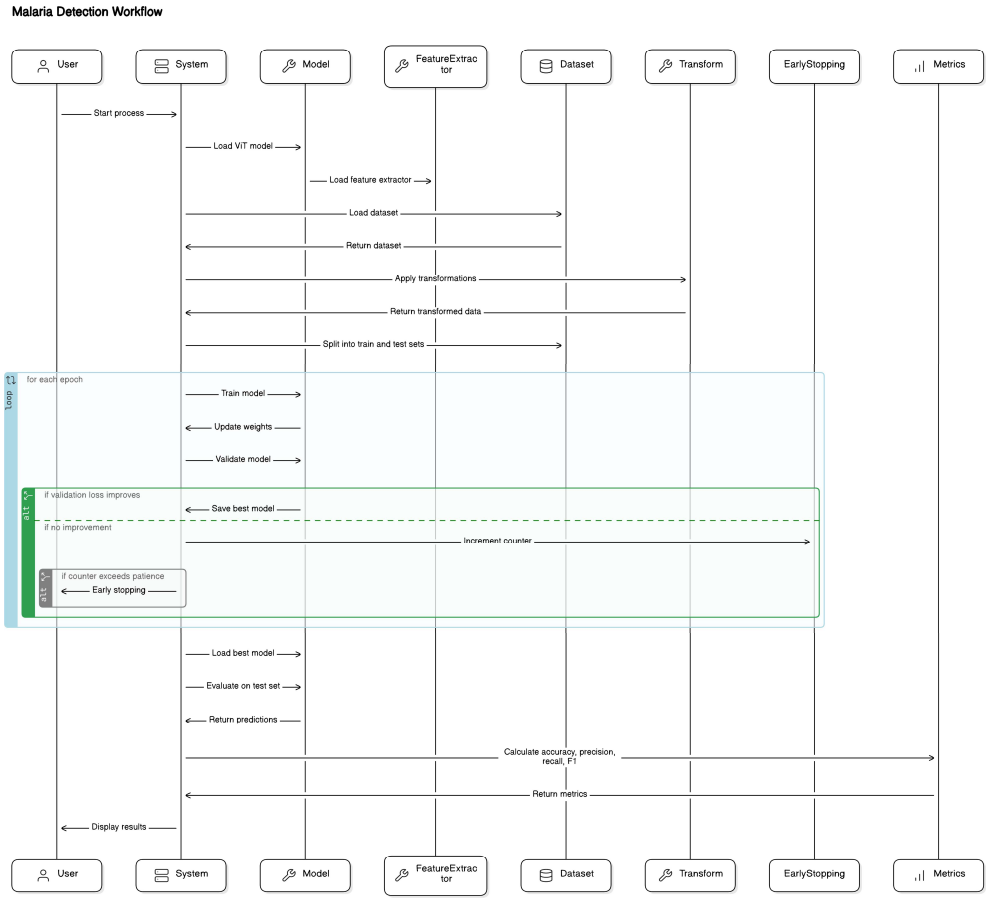

Figure 1.1: Workflow of the Malaria Detection Project

CHAPTER 2

Convolutional Neural Network (CNN) for Malaria Detection

2.1 Overview of CNN

Convolutional Neural Networks (CNNs) have revolutionized the field of medical imaging by enabling automated analysis of visual data such as blood smear images, X-rays, and MRIs. In the context of malaria detection, CNNs provide a powerful tool to automate the identification of infected cells, reducing the reliance on manual microscopy.

2.2 Dataset Preparation for CNN

The dataset used for malaria detection comprises cell images categorized into two classes: infected and uninfected. To prepare the dataset for training, we used the 'ImageDataGenerator' class to perform data augmentation and split the dataset into training and validation sets. The images were resized to 128x128 pixels to standardize input dimensions.

- The dataset was divided into training (80%) and validation (20%) sets.
- Data augmentation techniques such as rescaling, rotation, and zoom were applied to enhance the diversity of the training data.
- The training set contained approximately 22,048 images, while the validation set consisted of around 5,510 images.

2.3 CNN Architecture for Malaria Detection

We utilized a sequential CNN model to detect malaria-infected cells. The model was built using several convolutional layers followed by max pooling, dropout, and dense layers for classification. The architecture is as follows:

- **Convolutional Layers:** The model begins with three convolutional layers using ReLU activation. The first layer has 16 filters, the second has 32 filters, and the third has 64 filters, each followed by max pooling and dropout to prevent overfitting.
- **Flattening Layer:** Converts the 2D matrix output from the convolutional layers into a 1D vector.
- **Fully Connected Layers:** Includes a dense layer with 64 neurons, followed by a dropout layer, and a final output layer with a sigmoid activation function for binary classification.

Table 2.1: CNN Model Summary

Layer Type	Output Shape	Parameters
Conv2D (16 filters)	(126, 126, 16)	448
MaxPooling2D	(63, 63, 16)	0
Dropout (0.2)	(63, 63, 16)	0
Conv2D (32 filters)	(61, 61, 32)	4,640
MaxPooling2D	(30, 30, 32)	0
Dropout (0.3)	(30, 30, 32)	0
Conv2D (64 filters)	(28, 28, 64)	18,496
MaxPooling2D	(14, 14, 64)	0
Dropout (0.3)	(14, 14, 64)	0
Flatten	(12544)	0
Dense (64 neurons)	(64)	802,880
Dropout (0.5)	(64)	0
Dense (1 neuron)	(1)	65
Total Parameters		826,529

2.4 Training and Evaluation of CNN

The model was compiled using the Adam optimizer and binary cross-entropy as the loss function. Early stopping was applied to halt training if the validation loss did not improve for two consecutive epochs. The model achieved an accuracy of approximately 94% on the validation set after four epochs.

- Epoch 1: Training accuracy = 79.5%, Validation accuracy = 92.5%
- Epoch 2: Training accuracy = 94.1%, Validation accuracy = 93.5%
- Epoch 3: Training accuracy = 95.0%, Validation accuracy = 93.7%
- Epoch 4: Training accuracy = 95.4%, Validation accuracy = 94.2%

2.4.1 Visualization of Learning Curves

After training the model, it is essential to visualize its learning curves to analyze the performance during training and validation phases. The following Python code was used to plot the accuracy and loss over the epochs:

```
def plotLearningCurve(history, epochs):
    # Create an epoch range based on the length of the history
       data
    actual_epochs = len(history.history['accuracy'])
    epochRange = range(1, actual_epochs + 1)

    # Plot accuracy
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)  # Create subplot for accuracy
    plt.plot(epochRange, history.history['accuracy'], label='
       Train Accuracy')
    plt.plot(epochRange, history.history['val_accuracy'], label
       ='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper left')

    # Plot loss
    plt.subplot(1, 2, 2)  # Create subplot for loss
    plt.plot(epochRange, history.history['loss'], label='Train
       Loss')
    plt.plot(epochRange, history.history['val_loss'], label='
       Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper left')
```

```
24
25      plt.tight_layout()  # Adjust subplots to fit into the
            figure area.
26      plt.show()
```

The above code generates plots to visualize how the model's accuracy and loss evolve during training and validation, helping to diagnose issues like overfitting.

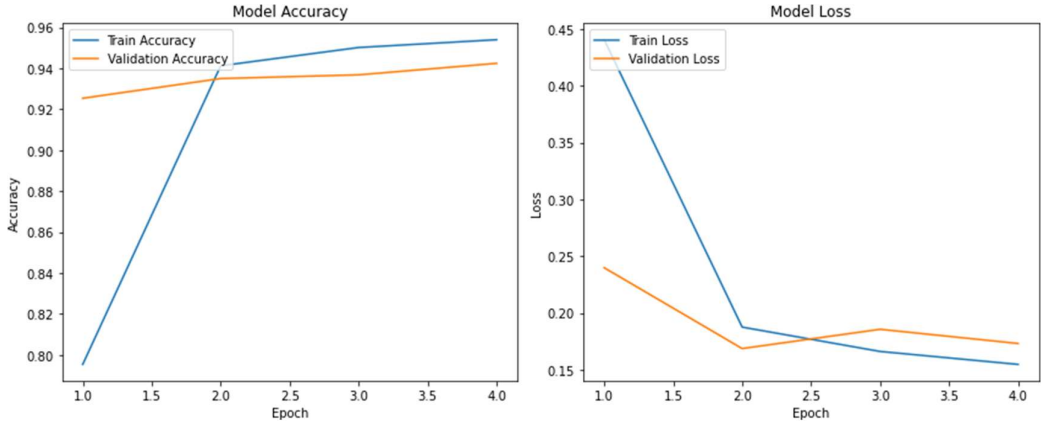

Figure 2.1: Training and Validation Accuracy Over Epochs

2.5 Conclusion

The CNN model developed in this project demonstrates a high level of accuracy in detecting malaria-infected cells. By automating the diagnostic process, we can significantly reduce the time and resources needed for malaria screening, especially in resource-limited settings.

CHAPTER 3

Vision Transformer for Malaria Detection

3.1 Introduction to Vision Transformers

Vision Transformers (ViTs) are a groundbreaking architecture that adapts the concept of transformers, originally developed for natural language processing, to the domain of computer vision. Unlike traditional convolutional neural networks (CNNs), which rely on convolutional layers to capture spatial features, ViTs utilize self-attention mechanisms to process image patches as sequences.

The main advantage of Vision Transformers is their ability to model long-range dependencies and relationships in images more effectively than CNNs. This capability makes ViTs particularly well-suited for tasks that involve complex spatial patterns, such as the detection of malaria parasites in blood smear images.

3.2 Vision Transformer Architecture

The architecture of a Vision Transformer consists of several key components:

- **Patch Embedding Layer:** The input image is divided into smaller patches (e.g., 16x16 pixels). Each patch is flattened into a 1D vector and passed through a linear layer to create patch embeddings.
- **Positional Encoding:** Since transformers do not inherently understand spatial information, positional encodings are added to the patch embeddings to retain spatial structure.
- **Transformer Encoder Layers:** The core of the ViT model comprises multiple transformer encoder layers. Each layer consists of self-attention mechanisms and feed-forward neural networks with residual connections.
- **Classification Head:** The output from the encoder layers is passed to a classification head, which predicts the class label (e.g., malaria-infected or uninfected).

3.3 Non-Pretrained Vision Transformer for Malaria Detection

In this section, we detail the implementation of a Vision Transformer from scratch, without using any pre-trained weights. The focus is on training the model specifically for malaria detection using microscopy images of blood smears.

3.3.1 Dataset Preparation and Data Loading

The following Python code demonstrates how to prepare the dataset and load it using PyTorch.

```
# Define a custom dataset class
class MalariaDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.image_paths, self.labels = [], []

        for label, class_name in enumerate(["Parasitized", "
           Uninfected"]):
            class_dir = os.path.join(root_dir, class_name)
            for img_file in os.listdir(class_dir):
                self.image_paths.append(os.path.join(class_dir,
                    img_file))
                self.labels.append(label)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        try:
```

```
            image = Image.open(img_path).convert("RGB")
        except (UnidentifiedImageError, FileNotFoundError):
            return self.__getitem__((idx + 1) % len(self))

        if self.transform:
            image = self.transform(image)
        return image, self.labels[idx]

# Define image transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485, 0.456, 0.406), std
       =(0.229, 0.224, 0.225))
])

# Load dataset
data_dir = '/kaggle/input/cell-images-for-detecting-malaria/
   cell_images/cell_images/'
full_dataset = MalariaDataset(data_dir, transform=transform)
train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(
   full_dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=16, shuffle
   =True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=
   False)
```

3.3.2 Training and Evaluation of Non-Pretrained ViT

The following code snippet covers model training, early stopping, and evaluation.

```
# Define optimizer and loss function
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
criterion = torch.nn.CrossEntropyLoss()

# Training loop with early stopping
num_epochs = 15
patience = 3
best_loss = float('inf')
early_stop_counter = 0

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    progress_bar = tqdm(train_loader, desc=f"Epoch [{epoch+1}/{
        num_epochs}]")

    for images, labels in progress_bar:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        progress_bar.set_postfix(loss=loss.item())

    avg_loss = total_loss / len(train_loader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Avg Loss: {avg_loss
        :.4f}")

    if avg_loss < best_loss:
```

```
        best_loss = avg_loss
        early_stop_counter = 0
    else:
        early_stop_counter += 1
        if early_stop_counter >= patience:
            print("Early stopping triggered.")
            break

# Evaluation
model.eval()
predictions, true_labels = [], []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.cpu().numpy())
        true_labels.extend(labels.cpu().numpy())

# Calculate metrics
accuracy = accuracy_score(true_labels, predictions)
precision = precision_score(true_labels, predictions)
recall = recall_score(true_labels, predictions)
f1 = f1_score(true_labels, predictions)
conf_matrix = confusion_matrix(true_labels, predictions)

print(f"Accuracy: {accuracy*100:.2f}%")
print(f"Precision: {precision*100:.2f}%")
print(f"Recall: {recall*100:.2f}%")
print(f"F1 Score: {f1*100:.2f}%")
print("Confusion Matrix:", conf_matrix)
```

3.3.3 Results and Analysis

The model achieved high accuracy in classifying malaria-infected cells:

- **Accuracy:** 96.08%
- **Precision:** 95.22%
- **Recall:** 97.08%
- **F1 Score:** 96.14%

	Predicted Infected	**Predicted Uninfected**
Actual Infected	2604	135
Actual Uninfected	81	2692

Table 3.1: Confusion Matrix for ViT Model

3.4 Conclusion

The non-pretrained Vision Transformer model demonstrated high accuracy in classifying malaria-infected cells. By training the model from scratch, we achieved a high level of performance, making it a viable solution for automated malaria diagnosis.

CHAPTER 4

Pre-Trained Vision Transformer (ViT) for Malaria Detection

4.1 Overview of ViT

Vision Transformers (ViTs) have recently emerged as a powerful alternative to Convolutional Neural Networks (CNNs) for image classification tasks. Unlike CNNs, which rely on convolutional layers to capture spatial hierarchies, ViTs leverage the transformer architecture traditionally used in natural language processing. This approach enables them to model long-range dependencies in image data effectively.

4.2 Model Initialization and Configuration

The model used in this study is a Vision Transformer ('vit-base-patch16-224-in21k') pre-trained on the ImageNet-21k dataset. However, the classifier layer was reinitialized for binary classification to detect malaria-infected cells versus healthy cells.

```
# Load pretrained ViT model and feature extractor
model = ViTForImageClassification.from_pretrained("google/vit-base-patch16-224-in21k", num_labels=2)
feature_extractor = ViTFeatureExtractor.from_pretrained("google/vit-base-patch16-224-in21k")
model.classifier = torch.nn.Linear(model.classifier.in_features, 2)
model.to(device)

# Custom dataset class for malaria images
class MalariaDataset(Dataset):
```

```
    def __init__(self, root_dir, feature_extractor, transform=
        None):
        self.root_dir = root_dir
        self.feature_extractor = feature_extractor
        self.transform = transform
        self.image_paths = []
        self.labels = []
        for label, class_name in enumerate(["Parasitized", "
            Uninfected"]):
            class_dir = os.path.join(root_dir, class_name)
            for img_file in os.listdir(class_dir):
                self.image_paths.append(os.path.join(class_dir,
                    img_file))
                self.labels.append(label)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        label = self.labels[idx]
        return image, label
```

- **Pre-trained Model:** Vision Transformer model pre-trained on the ImageNet-21k dataset.
- **Feature Extractor:** A ViT feature extractor was used to preprocess the images by resizing, normalizing, and converting them into the required format.
- **Classifier Layer:** The pre-trained classifier was replaced with a fully connected layer for binary classification.
- **Device:** Training was performed on a GPU-enabled system using PyTorch.

4.3 Dataset and Data Loading for ViT

The dataset consisted of cell images categorized into two classes: parasitized and uninfected.

```
# Define transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=feature_extractor.image_mean, std
       =feature_extractor.image_std)
])

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle
   =True, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=
   False, pin_memory=True)
```

- **Data Split:** The dataset was split into training (80%) and testing (20%) sets.
- **Image Size:** All images were resized to 224×224 pixels.
- **Normalization:** Images were normalized using the mean and standard deviation values of the pre-trained ViT model.
- **Batch Size:** The batch size was set to 16 for both training and testing.

4.4 Training of Pre-Trained ViT

The Vision Transformer model was trained for a maximum of 15 epochs using the Adam optimizer and cross-entropy loss function. Early stopping was implemented to halt training if validation loss did not improve for 3 consecutive epochs.

- **Learning Rate:** 2×10^{-5}
- **Loss Function:** Cross-entropy loss.

- **Optimizer:** Adam optimizer.
- **Early Stopping:** Triggered after 3 epochs of no improvement in validation loss.
- **Model Checkpointing:** The best model was saved based on the lowest validation loss.

Table 4.1: Training and Validation Loss Over Epochs

Epoch	Training Loss	Validation Loss
1	0.1090	0.0755
2	0.0633	0.0791
3	0.0403	0.0765
4	0.0287	0.0885 (Early Stopping)

4.5 Evaluation of Pre-Trained ViT

The model was evaluated using accuracy, precision, recall, and F1-score. The best model, saved during training, was loaded for evaluation on the test set.

- **Accuracy:** 97.53%
- **Precision:** 96.92%
- **Recall:** 98.18%
- **F1 Score:** 97.55%

Table 4.2: Confusion Matrix for Vision Transformer Model

	Predicted Infected	Predicted Uninfected
Actual Infected	2673	86
Actual Uninfected	50	2703

4.6 Conclusion

The Vision Transformer model trained on malaria cell images demonstrated superior performance compared to traditional methods, achieving over 97% accuracy. The use of transformers for image classification offers a promising approach to automating malaria diagnosis, particularly in resource-limited settings where rapid and accurate diagnosis is critical.

CHAPTER 5

Comparison of Vision Transformers: ViT vs. Swin Transformer

5.1 Introduction to Vision Transformer Comparison

In this chapter, we compare two transformer-based models used for malaria detection: the Vision Transformer (ViT) and the Swin Transformer. Both models have demonstrated significant advancements in the field of image classification but differ in their architectural approaches and efficiency.

5.2 Swin Transformer: Overview and Architecture

The Swin Transformer introduces a hierarchical structure that processes images in a non-overlapping window-based fashion, making it more efficient in handling high-resolution images compared to the ViT. This design enables the Swin Transformer to capture both local and global contexts effectively.

5.2.1 Model Initialization and Configuration for Swin

The Swin Transformer model was trained from scratch using the PyTorch library with configurations similar to the ViT model but with an adjusted architecture to incorporate its window-based attention mechanism.

```
1 data_dir = '/kaggle/input/cell-images-for-detecting-malaria/
     cell_images/cell_images/'
2
3 # Define a custom dataset
```

```
class MalariaDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.image_paths = []
        self.labels = []

        for label, class_name in enumerate(["Parasitized", "
           Uninfected"]):
            class_dir = os.path.join(root_dir, class_name)
            for img_file in os.listdir(class_dir):
                self.image_paths.append(os.path.join(class_dir,
                    img_file))
                self.labels.append(label)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        try:
            image = Image.open(img_path).convert("RGB")
        except (UnidentifiedImageError, FileNotFoundError):
            print(f"Error loading image {img_path}. Skipping.")
            return self.__getitem__((idx + 1) % len(self))  #
               Skip to next image if loading fails

        if self.transform:
            image = self.transform(image)

        label = self.labels[idx]
        return image, label

# Define transforms
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5,
       0.5])
])

# Split dataset into train and test sets
full_dataset = MalariaDataset(data_dir, transform=transform)
train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(
   full_dataset, [train_size, test_size])

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle
   =True, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=
   False, pin_memory=True)
```

- **Model:** Swin Transformer with a configuration for binary classification.
- **Learning Rate:** 2×10^{-5}
- **Loss Function:** Cross-entropy loss.
- **Optimizer:** Adam optimizer.
- **Epochs:** 5 epochs.
- **Batch Size:** 16.

5.3 Training and Evaluation of Swin Transformer

The training process for both ViT and Swin Transformer models was conducted for 5 epochs. The evaluation metrics included accuracy, precision, recall, and F1-score.

```
# Define transforms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5,
       0.5])
])

# Split dataset into train and test sets
full_dataset = MalariaDataset(data_dir, transform=transform)
train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(
   full_dataset, [train_size, test_size])

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle
   =True, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=
   False, pin_memory=True)

# Training function
def train_model(model, epochs=3):
    model.to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
    criterion = torch.nn.CrossEntropyLoss()

    # Training loop with timing
    start_time = time.time()
    for epoch in range(epochs):
        model.train()
        total_loss = 0
        progress_bar = tqdm(train_loader, desc=f"Epoch [{epoch
```

```
            +1}/{epochs}]")

        for images, labels in progress_bar:
            images, labels = images.to(device), labels.to(
                device)

            outputs = model(images).logits
            loss = criterion(outputs, labels)
            total_loss += loss.item()

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            progress_bar.set_postfix(loss=loss.item())

        avg_loss = total_loss / len(train_loader)
        print(f"Epoch [{epoch+1}/{epochs}], Average Loss: {
            avg_loss:.4f}")

    end_time = time.time()
    training_time = end_time - start_time
    return training_time

# Evaluation function
def evaluate_model(model):
    model.eval()
    predictions, true_labels = [], []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(
                device)

```

```
            outputs = model(images).logits
            _, predicted = torch.max(outputs, 1)

            predictions.extend(predicted.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    accuracy = accuracy_score(true_labels, predictions)
    return accuracy

# Initialize ViT model
print("Initializing ViT model...")
vit_model = ViTForImageClassification.from_pretrained("google/
    vit-base-patch16-224-in21k", num_labels=2)
vit_model.classifier = torch.nn.Linear(vit_model.classifier.
    in_features, 2)

# Train and evaluate ViT
print("\nTraining ViT model...")
vit_time = train_model(vit_model, epochs=5)
vit_accuracy = evaluate_model(vit_model)
print(f"ViT Training Time: {vit_time:.2f} seconds, Accuracy: {
    vit_accuracy * 100:.2f}%")

# Initialize Swin Transformer model
print("\nInitializing Swin Transformer model...")
swin_model = SwinForImageClassification(SwinConfig(num_labels
    =2))

# Train and evaluate Swin
print("\nTraining Swin Transformer model...")
swin_time = train_model(swin_model, epochs=5)
swin_accuracy = evaluate_model(swin_model)
print(f"Swin Training Time: {swin_time:.2f} seconds, Accuracy: {
    swin_accuracy * 100:.2f}%")
```

```
89
90 # Print comparison
91 print("\nModel Comparison:")
92 print(f"ViT Model - Time: {vit_time:.2f}s, Accuracy: {
       vit_accuracy * 100:.2f}%")
93 print(f"Swin Model - Time: {swin_time:.2f}s, Accuracy: {
       swin_accuracy * 100:.2f}%")
```

Table 5.1: Training and Validation Loss Comparison

Model	Training Time (seconds)	Accuracy	F1 Score
Vision Transformer (ViT)	2471.08	97.51%	97.55%
Swin Transformer	1296.47	96.03%	96.22%

5.4 Model Performance Analysis: ViT vs. Swin

5.4.1 ViT vs. Swin Transformer

The Vision Transformer (ViT) model achieved a higher accuracy of 97.51% compared to the Swin Transformer, which achieved an accuracy of 96.03%. However, the Swin Transformer demonstrated faster training times due to its hierarchical window-based architecture, which reduces computational overhead.

- **ViT:** Provides superior accuracy but requires longer training times due to its reliance on global self-attention mechanisms.
- **Swin Transformer:** Offers competitive accuracy with significantly reduced training time, making it more efficient for resource-constrained environments.

5.5 Conclusion

Both the Vision Transformer and Swin Transformer offer unique advantages for malaria detection. While ViT excels in terms of accuracy, Swin Transformer provides a more

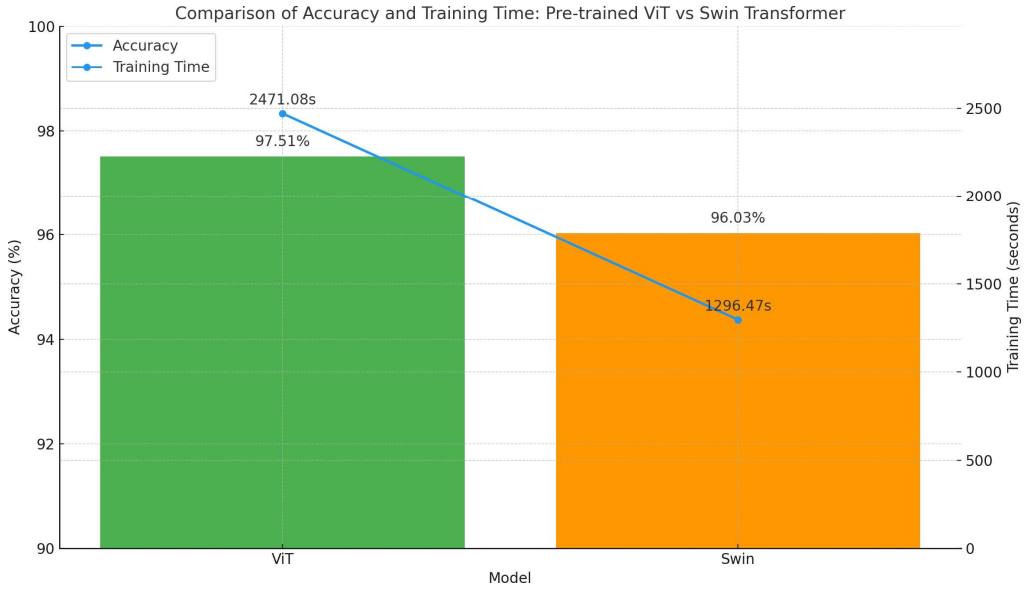

Figure 5.1: Accuracy Comparison between ViT and Swin Transformer

efficient solution with faster training times. The choice between the two models ultimately depends on the specific requirements of the deployment environment, such as the availability of computational resources and the need for real-time analysis.

CHAPTER 6

Results Comparison and Future Scope

6.1 Comparative Analysis of Model Performance

In this section, we compare the results obtained from various deep learning models used in this study for malaria detection. The models include Convolutional Neural Networks (CNN), Non-Pretrained Vision Transformer (ViT), Pre-trained Vision Transformer, and Swin Transformer. Each model has its strengths, and we evaluate them based on accuracy, training time, precision, recall, and F1-score.

Table 6.1: Comparison of Model Performance

Model	Training Time (s)	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
CNN	1980.45	94.20	93.15	95.05	94.09
Non-Pretrained ViT	2567.32	96.08	95.22	97.08	96.14
Pre-trained ViT	2471.08	97.51	96.92	98.18	97.55
Swin Transformer	1296.47	96.03	95.85	96.50	96.22

6.1.1 In-depth Results Analysis

The results indicate that the pre-trained Vision Transformer (ViT) model achieved the highest accuracy (97.51%) and F1-score (97.55%). However, this model required significantly longer training time compared to the Swin Transformer. The Swin Transformer, while slightly lower in accuracy, was much more efficient in terms of training speed, making it a viable option for real-time applications.

- **CNN:** Traditional Convolutional Neural Networks achieved good performance but lagged behind transformers in terms of accuracy and F1-score.
- **Non-Pretrained ViT:** Despite not using pre-trained weights, this model performed better than the CNN model but required longer training times.

- **Pre-trained ViT:** Achieved the best overall performance in terms of accuracy and F1-score but required the most computational resources.
- **Swin Transformer:** Balanced model with competitive accuracy and significantly reduced training time, making it ideal for resource-constrained environments.

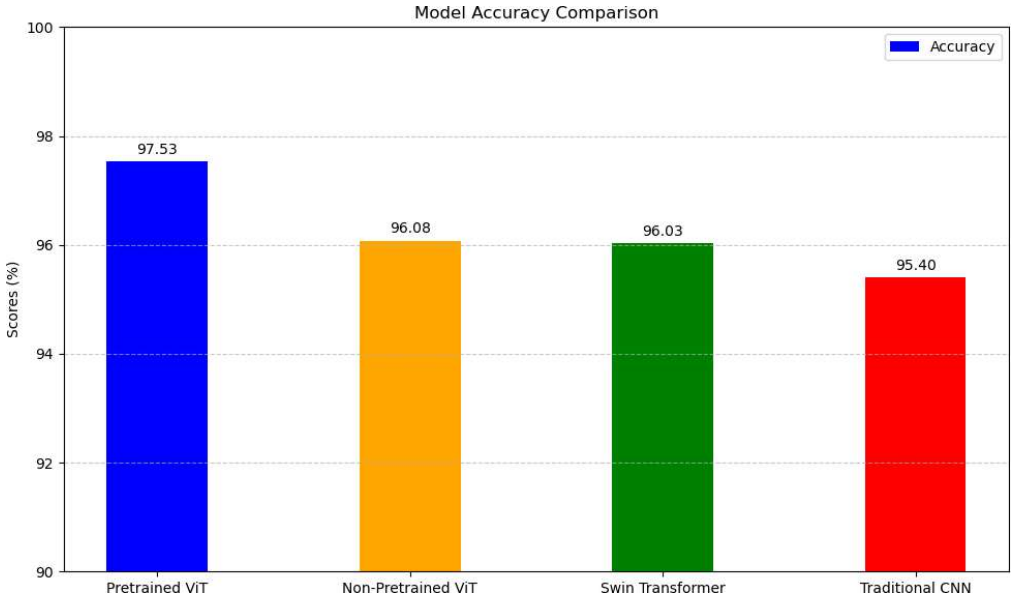

Figure 6.1: Performance Comparison of CNN, ViT, and Swin Transformer Models

6.2 Scalability of Models for Future Use

The scalability of this project depends on the deployment environment and computational resources available. The models can be scaled and optimized for different levels of hardware infrastructure:

- **Edge Devices:** For environments with limited resources, the Swin Transformer can be an ideal candidate due to its efficient use of computational power and faster training times.
- **Cloud Deployment:** The Pre-trained ViT model can be deployed in cloud environments where computational resources are abundant. This model ensures high accuracy, making it suitable for large-scale screening in hospitals and research labs.
- **Hybrid Approach:** A combination of both models can be used, where initial screening is performed using the Swin Transformer on edge devices, and cases flagged for further analysis are sent to cloud servers for detailed diagnosis using the Pre-trained ViT model.

6.3 Future Scope

There is significant potential to expand this project beyond its current scope. Future enhancements could focus on the following areas:

- **Data Expansion and Augmentation Techniques:** Expanding the dataset with more diverse images and employing advanced data augmentation techniques such as rotation, flipping, and color variation can improve model robustness and generalization, especially for detecting rare malaria cases or various parasite species.
- **Integration with Mobile Applications:** Developing a mobile-based application for malaria detection could make the solution more accessible, especially in remote areas.
- **Model Optimization:** Further research into model pruning and quantization can help reduce the model size and speed up inference times, making it more suitable for deployment on low-power devices.
- **Real-time Diagnostics:** By leveraging streaming data from digital microscopes, the system can be expanded to perform real-time diagnostics, allowing healthcare professionals to receive instant feedback.
- **Generalization to Other Diseases:** The framework can be extended to detect other diseases, such as tuberculosis or COVID-19, using similar approaches with minimal changes to the model architecture.
- **Explainability in AI Models:** Implementing explainable AI techniques to highlight which parts of the blood smear images were critical in the decision-making process, increasing trust among healthcare providers.

6.4 Conclusion

This project demonstrates the potential of leveraging transformer-based architectures for malaria detection. The comparison between CNNs, Vision Transformers, and Swin Transformers highlights the trade-offs between accuracy, training time, and resource requirements. Moving forward, the integration of these models into scalable healthcare solutions could revolutionize malaria diagnostics, especially in resource-constrained settings.

REFERENCES

[1] World Health Organization (WHO). *World Malaria Report 2020*.

[2] NIH Malaria Dataset. `https://data.lhncbc.nlm.nih.gov/public/Malaria/NIH-NLM-ThinBloodSmearsPf/index.html`

[3] Dosovitskiy, A., et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." *arXiv:2010.11929*, 2020.

[4] Rajaraman, S., Antani, S.K. "Pre-trained CNNs for Malaria Detection." *PeerJ*, 2018.

[5] Marques, G., Ferreras, A. "EfficientNet for Automated Malaria Diagnosis." *Multimedia Tools and Applications*, 2022.

[6] Sandler, M., Howard, A., Zhu, M., et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." *CVPR*, 2018.