National College of Ireland

Configuration Manual

Research Project
MSc Data Analytics

Teny Jose
Student ID: x23189371

School of Computing
National College of Ireland

Supervisor: Prof. Anu Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Teny Jose
Student ID:	x23189371
Programme:	MSc Data Analytics **Year:** 2025
Module:	Research Project
Lecturer:	Prof. Anu Sahni
Submission Due Date:	24 April 2025
Project Title:	An Ensemble Deep Learning Approach For Breast Cancer Classification Using Vision Transformers and CNNs
Word Count:	1324 **Page Count:** 8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Teny Jose

Date: 24 April 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Teny Jose
Student ID: x23189371

1 Hardware And Software Requirements

1.1 Hardware Requirements

The following hardware configuration is considered ideal for running the experiment smoothly.

Specification	Details
Operating System	Linux (Amazon EC2 Ubuntu-based AMI)
RAM	32 GB
GPU	NVIDIA A10G GPU (Amazon EC2 g5.2xlarge instance)
Storage	Minimum 100 GB SSD
Processor	Intel Xeon processor

NOTE: The project was executed remotely on an Amazon Web Services (AWS) EC2 instance equipped with deep learning capabilities.

1.2 Software Requirements

Software/Tool	Version	Purpose
Python	3.12	Programming Language
PyTorch	2.6.0+cu126	Deep Learning Framework (InceptionV4, EfficientNet, ViT)
scikit-learn	1.4.2	Evaluation metrics (accuracy, confusion matrix, ROC curve)
Amazon EC2 instance	Deep Learning AMI	Compute environment
Jupyter Notebook	Server version 7+	Development Environment

2 Environment Setup

2.1 AWS EC2 Instance Setup

1. Launch an EC2 instance with the Deep Learning AMI in AWS console.
2. Select g5.2xlarge instance type for GPU access.
3. Configure Security Groups: Allow inbound rules for SSH (port 22) and Custom TCP (port 8888) for Jupyter Notebook access.
4. Create and download a Key Pair (.pem) for secure connection.

2.2 Connecting to the EC2 Instance

For connecting the EC2 Instance the SSH client used was PuTTY.

Connection Steps:

1. Convert .pem key file to .ppk format using PuTTYgen.
2. Open PuTTY and connect to the instance's Public IPv4 address using SSH on port 22.

Figure 1 shows the PuTTY configuration settings used to establish a secure SSH connection to the AWS EC2 instance.

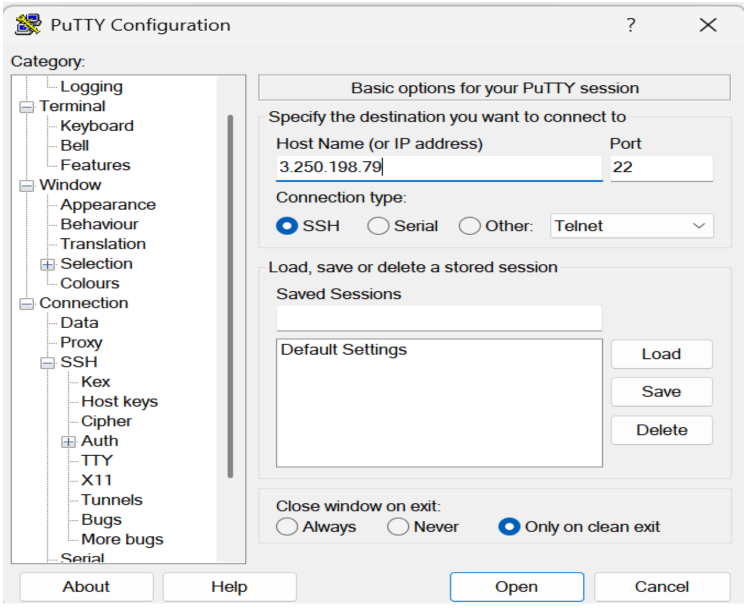

Figure 1: PuTTY Configuration for EC2 SSH Connection

2.3 Setting up Jupyter Notebook Server

After setting up the EC2 instance and SSH connection, a Jupyter Notebook server was started using port 8888. The notebook was accessed through the EC2 public IP in a browser (http://3.250.198.79:8889/tree). As shown in Figure 2, the Research_Project folder is in the home directory, containing the project files and datasets.

Figure 2: Jupyter Notebook Home Page Showing Project Directory

3 Data Preparation

The project used two publicly available histopathological datasets: **BreakHis** and **BACH**. The BreakHis dataset contained 7,930 images divided into two primary classes, *Benign* and *Malignant*, covering different magnifications (40X, 100X, 200X, 400X). The BACH dataset consisted of 400 high-resolution images, which were reorganized similarly for external evaluation. Figure 3 and Figure 4 shows the organized folder structure used for training and testing.

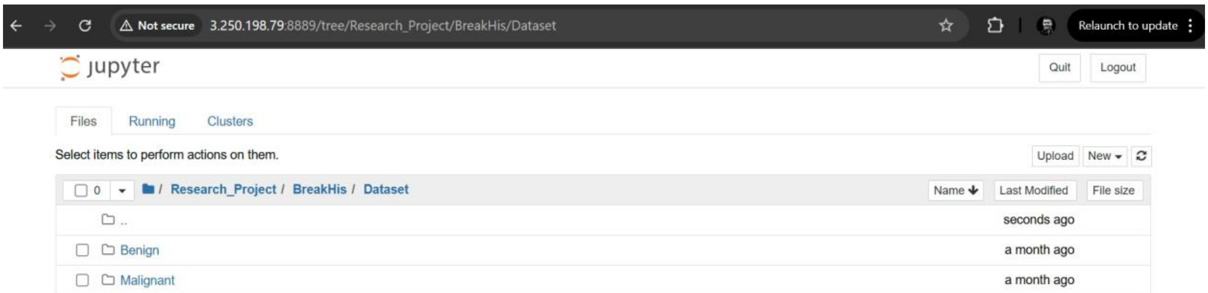

Figure 3: Organized Folder Structure of BreakHis

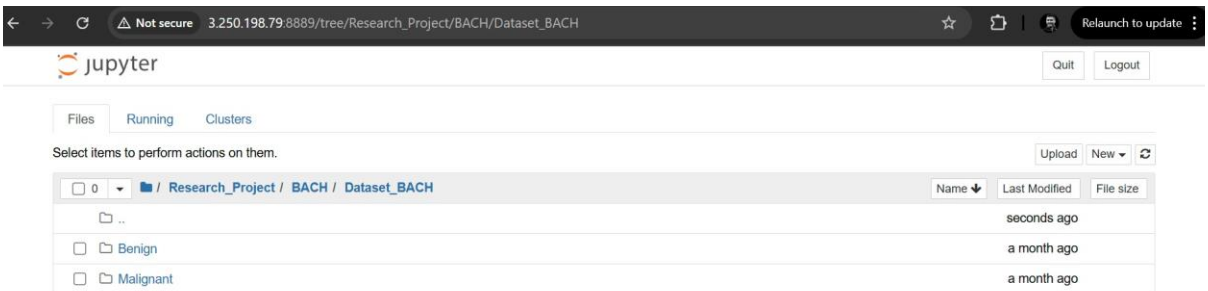

Figure 4: Organized Folder Structure of BACH

To prepare the data, images were sorted into two separate folders (*Benign* and *Malignant*) under both datasets, merging all magnifications into their respective classes. Standard preprocessing steps were applied, including resizing images to 299×299 pixels for CNN models (InceptionV4, EfficientNet) and 224×224 pixels for ViT and normalizing using ImageNet mean and standard deviation values. Data augmentations such as random rotations, flips, and colour jitter were applied during training to improve generalization and model robustness. Figure 4 shows the image preprocessing and augmentation pipeline used during training of Inception model.

Inception-v4

In [18]:

```
# Data Transforms and Loading
train_transform = transforms.Compose([
    transforms.Resize((299, 299)),
    transforms.RandomResizedCrop(299, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=imagenet_mean,
                         std=imagenet_std),
])

val_transform = transforms.Compose([
    transforms.Resize((299, 299)),
    transforms.ToTensor(),
    transforms.Normalize(mean=imagenet_mean,
                         std=imagenet_std),
])
```

Figure 4: Data Augmentation

4 Model Configuration

Three deep learning models were implemented in the study. All three models, Incepion-V4, EfficientNet, and ViT are pre-trained on ImageNet and are then fine-tuned in BreakHis dataset.

Training was conducted in two phases:

- **Phase 1 (Transfer Learning):** Only the final classification layers were trained while freezing the other feature extractors.
- **Phase 2 (Fine-tuning):** All model parameters were unfrozen and trained with a lower learning rate to further optimize performance.

4.1 InceptionV4 Configuration

Setting	Value
Model	InceptionV4
Input Image Size	299 × 299 pixels
Final Layer	Replaced with 2-output fully connected layer
Optimizer	Adam
Learning Rate	1e-3 (Phase 1), 1e-5 (Phase 2)

Setting	Value
Loss Function	Cross-Entropy Loss
Batch Size	32
Total Epochs	20 (10 + 10)
Device	NVIDIA GPU

4.2 EfficientNet-B0 Configuration

Setting	Value
Model	EfficientNet-B0
Input Image Size	299 × 299 pixels
Final Layer	Replaced with 2-output fully connected layer
Optimizer	Adam
Learning Rate	1e-3 (Phase 1), 1e-5 (Phase 2)
Loss Function	Cross-Entropy Loss
Batch Size	32
Total Epochs	20 (10 + 10)
Device	NVIDIA GPU

4.3 Vision Transformer (ViT) Configuration

Setting	Value
Model	Vision Transformer
Input Image Size	224 × 224 pixels
Final Layer	Modified classification head for 2 classes
Optimizer	Adam
Learning Rate	1e-3 (Phase 1), 1e-5 (Phase 2)
Loss Function	Cross-Entropy Loss
Batch Size	32
Total Epochs	20 (10 + 10)
Device	NVIDIA GPU

Figure 5 presents the core training loop implemented for all base models. This modular function handles forward pass, loss computation, and backpropagation across a single training epoch. The snippet shows the code for ViT, similar code is used for other two base models as well.

```
In [83]: # Training

def vit_train_one_epoch(model, dataloader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
    return running_loss / total, correct / total
```

Figure 5: Training Loop

4.4 Ensemble Setup

After individually fine-tuning InceptionV4, EfficientNet-B0, and Vision Transformer models, an ensemble model was created to further improve classification performance. The ensemble method used was **stacking,** where:

- The outputs (predicted probabilities) from each of the three base models were used as features.
- Random Forest Classifier used as the meta classifier was trained on these features to make the final predictions.

Component	**Value**
Base Models	InceptionV4, EfficientNet-B0, Vision Transformer
Meta-classifier	Random Forest Classifier
Training Data for Ensemble	Validation predictions from base models
Input to Meta-classifier	Predicted class probabilities (softmax outputs)
Output	Final binary classification (Benign / Malignant)

5 Training and Evaluation Setup

The dataset was divided as follows:

- **Training Set:** 80% of BreakHis images
- **Validation Set:** 20% of BreakHis images
- **External Test Set:** 400 images from the BACH dataset

For evaluation, the following metrics were calculated:

- Accuracy
- Precision
- Recall
- F1-Score

The final evaluation was performed on the external test dataset after fine-tuning the individual models. For example, Figure 6 shows the classification report for the EfficientNet-B0 model, summarizing its performance across the benign and malignant classes after external evaluation and after fine tuning.

```
In [74]: accuracy = accuracy_score(all_labels, all_preds)
         print(f"External Test Set Accuracy: {accuracy:.4f}")

         print("\nExternal Test Set Evaluation After Fine-Tuning:")
         print(classification_report(all_labels, all_preds, target_names=external_dataset.classes))

External Test Set Accuracy: 0.6875

External Test Set Evaluation After Fine-Tuning:
              precision    recall  f1-score   support

      Benign       0.84      0.45      0.59       157
   Malignant       0.63      0.91      0.75       163

    accuracy                           0.69       320
   macro avg       0.73      0.68      0.67       320
weighted avg       0.73      0.69      0.67       320
```

Figure 6: EfficientNet-B0 Classification Report on External Test Set

6 Other Software Used

For visualization of the evaluation part, Microsoft Excel was used. Microsoft Word was used for writing, formatting, and organizing the research project report as shown in Figure 6.

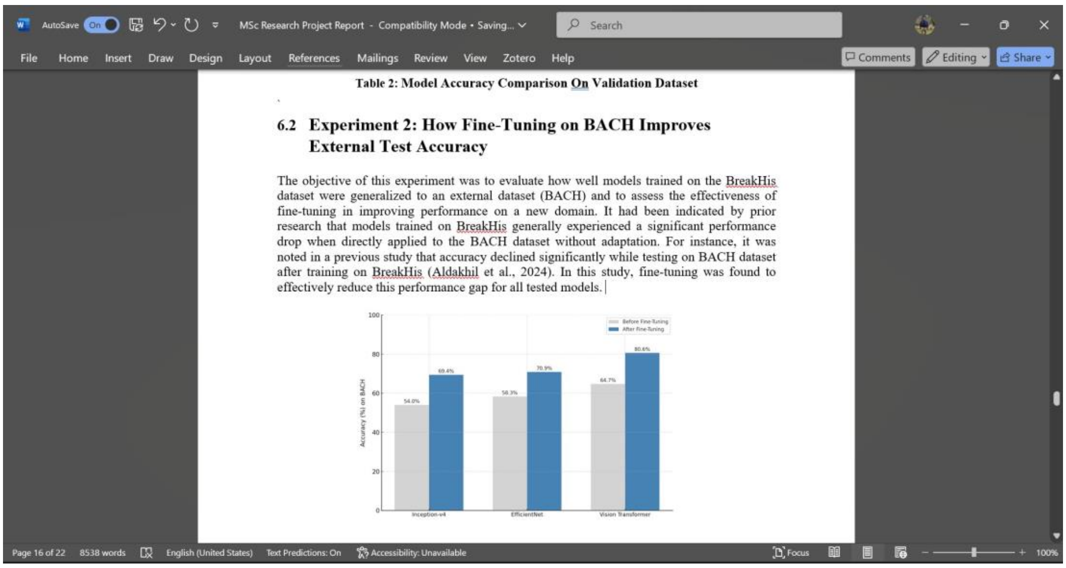

Figure 5: Word Project

7 Conclusion

This configuration manual documents the complete setup, environment configuration, model training, and evaluation method for the ensemble-based breast cancer classification project. It make sure that the project is fully reproducible and future research could be done on it.

References

- https://pytorch.org/docs/stable/index.html
  (PyTorch documentation used for model building and training)
- *https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics*
  (scikit-learn documentation used for evaluation)
- https://matplotlib.org/stable/contents.html
  (Matplotlib documentation used for generating plots and visualizations)