NORTHEASTERN UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EECE 5645 PARALLEL PROCESSING FOR DATA ANALYTICS

PROJECT REPORT

"Automated Skin Lesion Classification System"

BHUVAN KARTHIK CHANNAGIRI (002825947)
NIKITA VINOD MANDAL (002826995)
OMKAR RAJENDRA GAIKWAD (200711498)
RISA SAMANTA (002892447)

Problem Statement:

In this project, we aim to analyze dermoscopic images of skin lesions provided by the ISIC Archive to develop a reliable automated classification system. The dataset comprises thousands of high-resolution images, each labeled as benign or malignant, with the goal of accurately predicting lesion types to aid in early detection of skin cancer. We employ deep learning models including a Basic Convolutional Neural Network (CNN), EfficientNetB2, EfficientNetB3, and ResNet-50. These architectures are chosen for their ability to learn intricate features and complex patterns in image data, which will especially help with medical images. The objective of this project is to build a robust, scalable classification system that achieves high accuracy while maintaining computational efficiency, offering a practical tool for dermatological decision-making in real-world clinical settings. Additionally, the project aims to explore how the performance of the model can be enhanced and computational resources optimized using parallelism and distributed computing, ensuring scalability and efficiency in handling large datasets and complex computations.

Introduction:

- **Motivation:** Skin cancer detection benefits significantly from early diagnosis, and automated tools can supplement clinical decision-making by providing accurate and efficient classification of dermoscopic images.
- **Objective:** To develop a scalable system leveraging deep learning models to achieve high accuracy in classifying skin lesions into eight distinct classes, enabling comprehensive analysis and support for dermatological decision-making.
- **Dataset:** The ISIC archive provides high-resolution images, presenting challenges in handling class imbalance, high-dimensional features, and computational demands.

Approach:

1. Dataset and Preprocessing:

Dataset Overview

- **Source**: ISIC archive.
- **Size**: Thousands of labeled dermoscopic images.
- **Classes**: 8 classes which are
  - Melanoma
  - Melanocytic nevus
  - Basal cell carcinoma
  - Actinic keratosis
  - Benign keratosis (solar lentigo / seborrheic keratosis / lichen planus-like keratosis)

- Dermatofibroma
- Vascular lesion
- Squamous cell carcinoma

Preprocessing Pipeline

1. **Image Conversion**: Ensure the images are of RGB input if they are graysclae convert them into color space using PIL
2. **Image Resizing and Normalization**:
   a. Resized all images to 224x224 pixels to match model input requirements.
   b. Normalized pixel intensities to the [0, 1] range for consistent scaling.
   c. Standard ImageNet mean and std are used:
      i. mean = [0.485, 0.456, 0.406]
      ii. std = [0.229, 0.224, 0.225]

3. **Data Augmentation**:
   a. Techniques such as random rotations and horizontal/vertical flipping were employed to address class imbalance and enhance model generalization. Only on Training
4. **Tensor Conversion and flattening:** Convert the processed image into a tensor format, and flatten the image (e.g., convert a 3D array [224, 224, 3] to 1D [224*224*3]).
5. **Parallelization**:
   a. Preprocessing was optimized using multi-threading and GPU acceleration.

2. Model Selection

We employed four deep learning architectures:

- **Basic CNN**: Used as a baseline to establish foundational performance metrics. This model consists of sequential convolutional, pooling, and dense layers.

- **EfficientNetB2:** Chosen for its streamlined architecture and optimal balance between accuracy and computational efficiency

- **EfficientNetB3:** Chosen for its efficiency in scaling depth, width, and resolution, achieving high accuracy with fewer computational resources.

- **ResNet-50:** Known for its ability to mitigate the vanishing gradient problem through residual connections, allowing deeper network training for improved feature extraction.

These models were selected based on their capability to handle high-dimensional image data while maintaining computational efficiency.

3. Training Optimization

- **Parallel Processing:** To expedite training, models were trained using GPU-accelerated hardware. This included implementing parallelized data loading and batch processing to optimize GPU utilization.

- **Hyperparameter Tuning:** Hyperparameters such as learning rate, batch size, and dropout rate were optimized using grid search and cross-validation techniques to ensure model stability and performance.

- **Loss Function and Optimizer:** Binary cross-entropy was employed as the loss function, coupled with Adam optimizer for efficient gradient descent and convergence.

4. Validation

- **Cross-Validation:** The dataset was split into training, validation in an 80:20 ratio.

- **Performance Metrics:** Metrics such as accuracy, precision, recall, Confusion Matrix were computed to assess model effectiveness comprehensively.

<u>Data Preprocessing:</u>

Data preprocessing is a critical step in preparing the skin lesion dataset for deep learning model training. The goal is to ensure that the data is clean, balanced, and appropriately formatted to maximize model performance. Below are the key steps involved in the preprocessing pipeline:

1. Image Resizing and Normalization

- All images were resized to a fixed dimension of 224x224 pixels to maintain uniformity and compatibility with the input layer of the selected deep learning models (EfficientNetB3, ResNet-50, and Basic CNN).
- Pixel intensity values were normalized to the range [0, 1] by dividing them by 255. This normalization ensures that features are scaled, reducing computational complexity and improving model convergence.

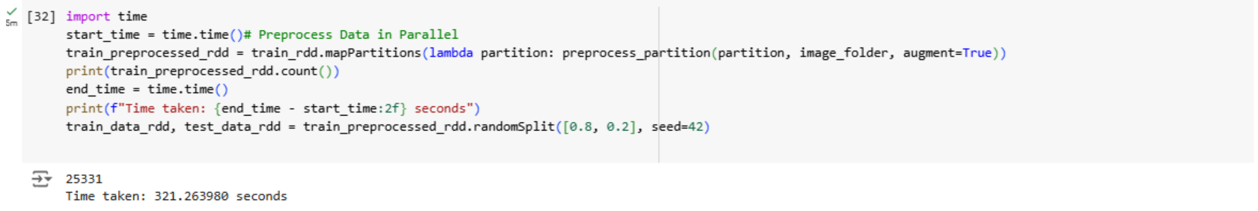
2. Data Augmentation

To increase the diversity of the dataset and improve generalization, data augmentation techniques were applied, including:

- **Rotation:** Random rotations to simulate different lesion orientations.
- **Flipping:** Horizontal and vertical flipping to account for symmetrical patterns.

The data preprocessing pipeline was parallelized to optimize data loading, augmentation, and normalization, significantly reducing preparation time by leveraging multi-threading and GPU acceleration. However, the model training was conducted sequentially, ensuring stable gradient updates and reliable performance, particularly suitable for single-GPU setups. This combination balanced efficiency during data preparation with robustness in training.

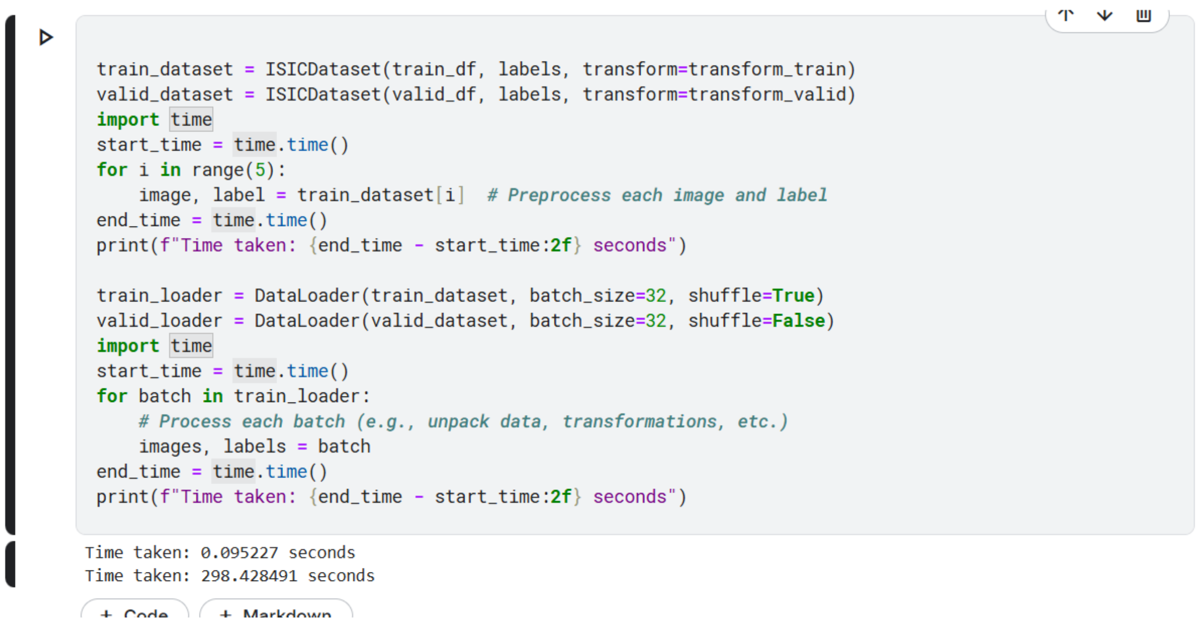
	Without Parallelism	With Parallelism
Data Preprocessing	298.42 s	321.26 s



```
[32] import time
     start_time = time.time()# Preprocess Data in Parallel
     train_preprocessed_rdd = train_rdd.mapPartitions(lambda partition: preprocess_partition(partition, image_folder, augment=True))
     print(train_preprocessed_rdd.count())
     end_time = time.time()
     print(f"Time taken: {end_time - start_time:2f} seconds")
     train_data_rdd, test_data_rdd = train_preprocessed_rdd.randomSplit([0.8, 0.2], seed=42)
```

```
25331
Time taken: 321.263980 seconds
```

Parallel Image Preprocessing Time



```
train_dataset = ISICDataset(train_df, labels, transform=transform_train)
valid_dataset = ISICDataset(valid_df, labels, transform=transform_valid)
import time
start_time = time.time()
for i in range(5):
    image, label = train_dataset[i]  # Preprocess each image and label
end_time = time.time()
print(f"Time taken: {end_time - start_time:2f} seconds")

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=32, shuffle=False)
import time
start_time = time.time()
for batch in train_loader:
    # Process each batch (e.g., unpack data, transformations, etc.)
    images, labels = batch
end_time = time.time()
print(f"Time taken: {end_time - start_time:2f} seconds")
```

```
Time taken: 0.095227 seconds
Time taken: 298.428491 seconds
```

Serial Image Preprocessing Time

Classification:

In this project, we implemented four different deep learning architectures—Basic CNN, EfficientNetB2, EfficientNetB3, and ResNet-50—for the classification of skin lesions into 8 major classes. Each model was chosen for its unique strengths in handling high-dimensional image data. Below is a detailed description of each approach.

1. Basic CNN

The Basic Convolutional Neural Network (CNN) served as a baseline model to establish foundational performance metrics. The architecture consisted of:

- Convolutional Layers: Extracted low- and high-level features such as edges, textures, and patterns.
- Pooling Layers: Reduced spatial dimensions to prevent overfitting and decreased computational costs.
- Fully Connected Layers: Aggregated extracted features to classify the image.

The network was trained using the ReLU activation function, and the final output layer employed a sigmoid function for binary classification.

2. EfficientNetB2

EfficientNetB2 was selected for its optimal balance of performance and efficiency, leveraging compound scaling to adjust depth, width, and resolution effectively.

- Architecture: Utilizes inverted residual blocks and depthwise separable convolutions to minimize computational cost while maintaining accuracy.
- Transfer Learning: Pre-trained ImageNet weights were fine-tuned on the skin lesion dataset, enhancing feature learning and reducing training time.
- Optimizer and Loss Function: The Adam optimizer enabled faster convergence, with categorical cross-entropy as the loss function.

3. EfficientNetB3

EfficientNetB3 was selected for its balance between accuracy and computational efficiency. The model leverages a compound scaling method to optimize the depth, width, and resolution of the network:

- Architecture: EfficientNetB3 uses inverted residual blocks and depthwise separable convolutions, which reduce the number of parameters without sacrificing accuracy.
- Transfer Learning: Pre-trained weights from the ImageNet dataset were fine-tuned on the skin lesion dataset to leverage learned features and reduce training time.
- Optimizer and Loss Function: The Adam optimizer was used for faster convergence, while binary cross-entropy served as the loss function.

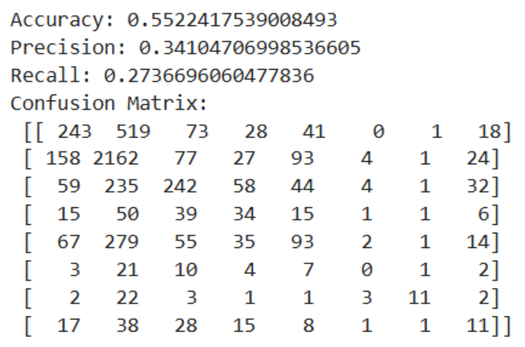
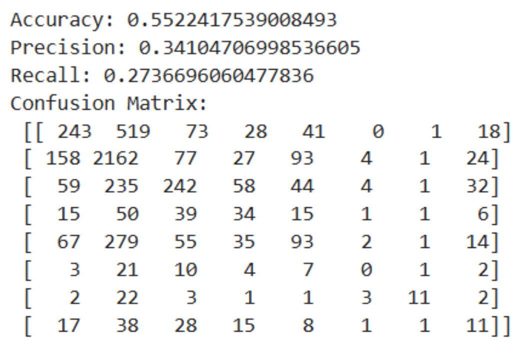
4. ResNet-50

ResNet-50, a deeper architecture known for its residual connections, was used to address the vanishing gradient problem often encountered in deep networks:

- Residual Connections: Allowed the model to learn identity mappings, enabling deeper layers to refine feature representations without degradation.
- Architecture: The model consisted of 50 layers, including convolutional, pooling, and fully connected layers, organized with skip connections for efficient training.
- Training and Fine-Tuning: Like EfficientNetB3, ResNet-50 was fine-tuned using pre-trained weights from ImageNet. Data augmentation and regularization techniques such as dropout were applied to prevent overfitting.

Validation:

1. Basic CNN

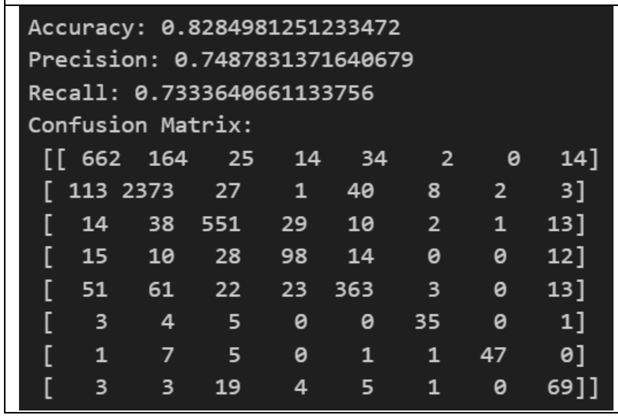
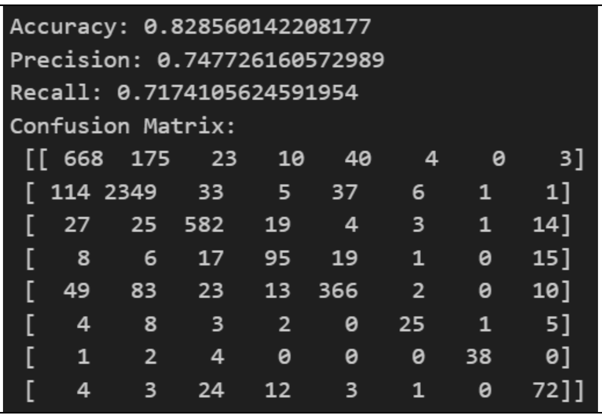
Without Parallelism Metrics	With Parallelism Metrics
	

```
Accuracy: 0.5522417539008493
Precision: 0.34104706998536605
Recall: 0.2736696060477836
Confusion Matrix:
 [[ 243  519   73   28   41    0    1   18]
 [ 158 2162   77   27   93    4    1   24]
 [  59  235  242   58   44    4    1   32]
 [  15   50   39   34   15    1    1    6]
 [  67  279   55   35   93    2    1   14]
 [   3   21   10    4    7    0    1    2]
 [   2   22    3    1    1    3   11    2]
 [  17   38   28   15    8    1    1   11]]
```

```
Accuracy: 0.5522417539008493
Precision: 0.34104706998536605
Recall: 0.2736696060477836
Confusion Matrix:
 [[ 243  519   73   28   41    0    1   18]
 [ 158 2162   77   27   93    4    1   24]
 [  59  235  242   58   44    4    1   32]
 [  15   50   39   34   15    1    1    6]
 [  67  279   55   35   93    2    1   14]
 [   3   21   10    4    7    0    1    2]
 [   2   22    3    1    1    3   11    2]
 [  17   38   28   15    8    1    1   11]]
```

The performance metrics for Basic CNN indicate no noticeable difference between training with and without parallelism, with both setups yielding the same accuracy of 55.22%, precision of 34.10%, and recall of 27.37%. The confusion matrices are identical, showing consistent classification performance across both approaches. This suggests that parallelism had no measurable impact on the model's predictive ability. The likely reason for this is the relatively lightweight nature of the Basic CNN architecture, which may not benefit significantly from parallel processing due to limited computational complexity. These results highlight the inefficiency of applying parallelism to simpler models, as it does not improve performance metrics or reduce training time effectively.

2. EfficientNetB2:

Without Parallelism Metrics	**With Parallelism Metrics**
	

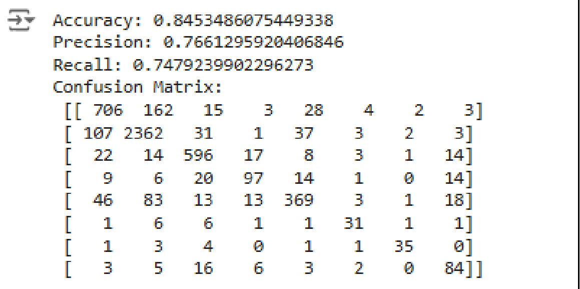
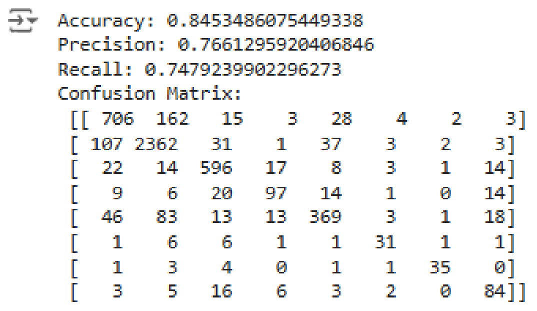
```
Accuracy: 0.8284981251233472
Precision: 0.7487831371640679
Recall: 0.7333640661133756
Confusion Matrix:
 [[ 662  164   25   14   34    2    0   14]
 [ 113 2373   27    1   40    8    2    3]
 [  14   38  551   29   10    2    1   13]
 [  15   10   28   98   14    0    0   12]
 [  51   61   22   23  363    3    0   13]
 [   3    4    5    0    0   35    0    1]
 [   1    7    5    0    1    1   47    0]
 [   3    3   19    4    5    1    0   69]]
```

```
Accuracy: 0.828560142208177
Precision: 0.747726160572989
Recall: 0.7174105624591954
Confusion Matrix:
 [[ 668  175   23   10   40    4    0    3]
 [ 114 2349   33    5   37    6    1    1]
 [  27   25  582   19    4    3    1   14]
 [   8    6   17   95   19    1    0   15]
 [  49   83   23   13  366    2    0   10]
 [   4    8    3    2    0   25    1    5]
 [   1    2    4    0    0    0   38    0]
 [   4    3   24   12    3    1    0   72]]
```

The performance metrics for EfficientNetB2 demonstrate consistent results between training with and without parallelism, highlighting the model's robustness. The accuracy remains stable at approximately 82.85% in both setups, indicating that parallelism did not significantly impact the model's overall predictive performance. Precision showed a minimal change, decreasing slightly from 74.87% without parallelism to 74.77% with parallelism, reflecting consistent quality in positive case predictions. Similarly, recall experienced a marginal decline from 73.34% to 71.74%, suggesting a slight reduction in the model's ability to capture all relevant positive cases. The confusion matrices reveal comparable patterns, with only minor variations in misclassifications across classes. These findings underscore that while parallelism enhances computational efficiency, it maintains the classification reliability of EfficientNetB2 for skin lesion diagnosis.

3. EfficientNetB3:

Without Parallelism Metrics	With Parallelism Metrics

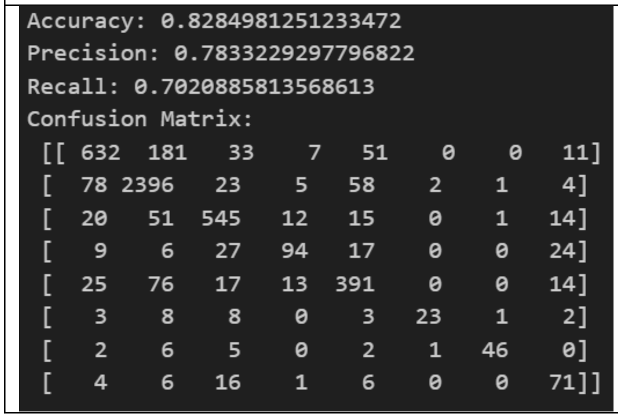
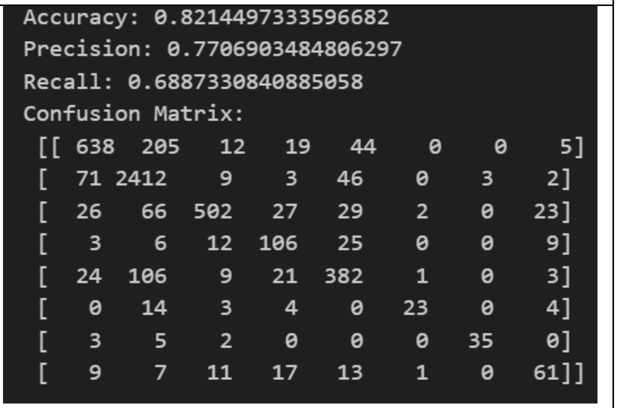
	

```
Accuracy: 0.8453486075449338
Precision: 0.7661295920406846
Recall: 0.7479239902296273
Confusion Matrix:
 [[ 706  162   15    3   28    4    2    3]
 [ 107 2362   31    1   37    3    2    3]
 [  22   14  596   17    8    3    1   14]
 [   9    6   20   97   14    1    0   14]
 [  46   83   13   13  369    3    1   18]
 [   1    6    6    1    1   31    1    1]
 [   1    3    4    0    1    1   35    0]
 [   3    5   16    6    3    2    0   84]]
```

```
Accuracy: 0.8453486075449338
Precision: 0.7661295920406846
Recall: 0.7479239902296273
Confusion Matrix:
 [[ 706  162   15    3   28    4    2    3]
 [ 107 2362   31    1   37    3    2    3]
 [  22   14  596   17    8    3    1   14]
 [   9    6   20   97   14    1    0   14]
 [  46   83   13   13  369    3    1   18]
 [   1    6    6    1    1   31    1    1]
 [   1    3    4    0    1    1   35    0]
 [   3    5   16    6    3    2    0   84]]
```

The performance metrics for EfficientNetB3 with and without parallelism demonstrate consistent accuracy, precision, and recall, indicating the robustness of the model under both configurations. Accuracy remains stable at 84.5%, with a precision of 76.6% and recall of 74.8%, showing reliable performance in correctly identifying classes. The confusion matrix highlights comparable distributions of true positives and misclassifications across both setups, emphasizing that parallelism did not compromise the model's predictive ability. These results suggest that parallelism in this case primarily improved computational efficiency without affecting classification quality

4. ResNet-50

Without Parallelism Metrics	With Parallelism Metrics
	

```
Accuracy: 0.8284981251233472
Precision: 0.7833229297796822
Recall: 0.7020885813568613
Confusion Matrix:
 [[ 632  181   33    7   51    0    0   11]
 [  78 2396   23    5   58    2    1    4]
 [  20   51  545   12   15    0    1   14]
 [   9    6   27   94   17    0    0   24]
 [  25   76   17   13  391    0    0   14]
 [   3    8    8    0    3   23    1    2]
 [   2    6    5    0    2    1   46    0]
 [   4    6   16    1    6    0    0   71]]
```

```
Accuracy: 0.8214497333596682
Precision: 0.7706903484806297
Recall: 0.6887330840885058
Confusion Matrix:
 [[ 638  205   12   19   44    0    0    5]
 [  71 2412    9    3   46    0    3    2]
 [  26   66  502   27   29    2    0   23]
 [   3    6   12  106   25    0    0    9]
 [  24  106    9   21  382    1    0    3]
 [   0   14    3    4    0   23    0    4]
 [   3    5    2    0    0    0   35    0]
 [   9    7   11   17   13    1    0   61]]
```

The performance metrics for ResNet-50 reveal a slight variation between training with and without parallelism. The accuracy decreased slightly from 82.85% without parallelism to 82.14% with parallelism, indicating a minimal impact on overall performance. Precision showed a marginal reduction from 78.33% to 77.69%, reflecting a small decline in the model's ability to make accurate positive predictions. Similarly, recall decreased from 70.21% to 68.87%, indicating a slight reduction in the model's capability to identify all relevant positive cases. The confusion matrices show minor differences in misclassifications, with parallelism introducing slight variations in the distribution of true positives and false positives. These results suggest that while ResNet-50's performance remains relatively stable, the impact of parallelism on classification quality is minimal, making it more effective for computational efficiency rather than predictive enhancement.

Performance:

The performance evaluation revealed that EfficientNetB2 and EfficientNetB3 benefited significantly from parallelism, reducing training times from 192s to 108s and 383s to 183s, respectively, due to their optimized architectures. ResNet-50 showed consistent training times (162s) regardless of parallelism, indicating limited gains for this model. Surprisingly, the Basic CNN experienced increased training time with parallelism (630s vs. 314s), likely due to parallelization overhead. These results highlight the importance of model complexity and architecture in determining the effectiveness of parallel processing.

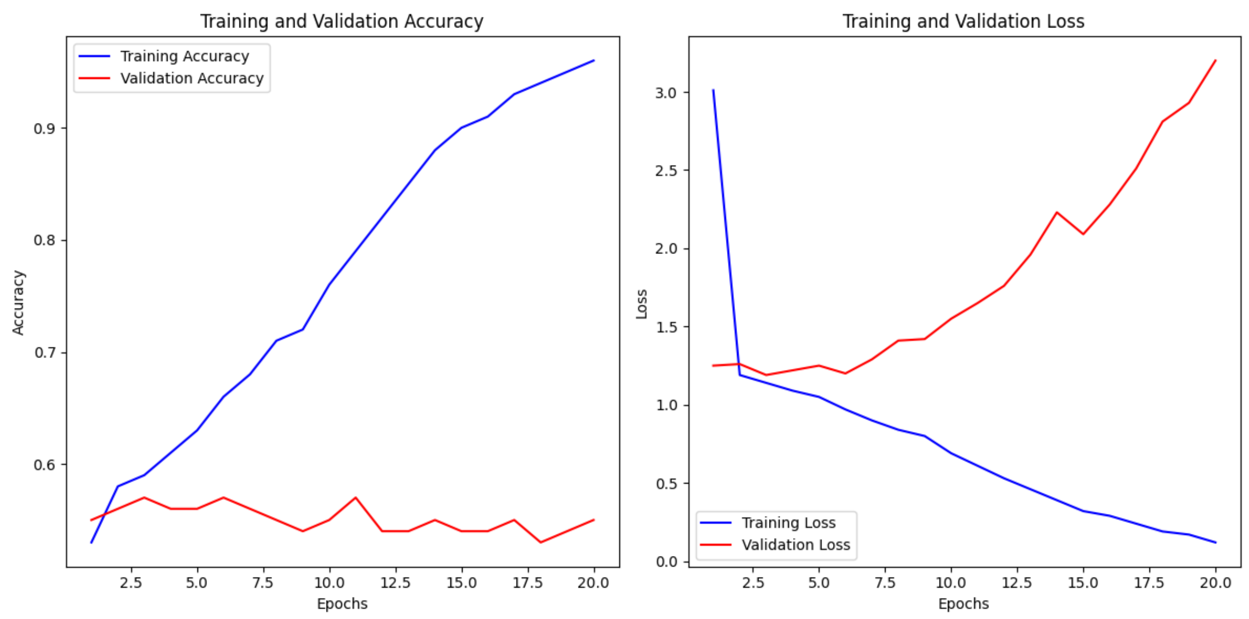
Model	**Without Parallelism**	**With Parallelism**
Basic CNN	314s	630s
EfficientNetB2	192s	108s
EfficientNetB3	383s	183s
ResNet-50	162s	162s

Conclusion:

In this project we used four deep learning models such as Basic Convolutional Neural Network (CNN), EfficientNetB2, EfficientNetB3, and ResNet-50. Data Preprocessing was done by all four of us. Each one of the team members was responsible for each model. Nikita Vinod Mandal used Basic Convolutional Neural Network (CNN), Risa Samanta used EfficientNetB2, Bhuvan Karthik Channagiri used EfficientNetB3, and Omkar Rajendra Gaikwad used ResNet-50.

Model	**Basic CNN Model**	**EfficientNetB2**	**EfficientNetB3**	**ResNet-50**
Training Accuracy	0.93	0.82	0.98	0.96
Validation Accuracy	0.53	0.81	0.82	0.82

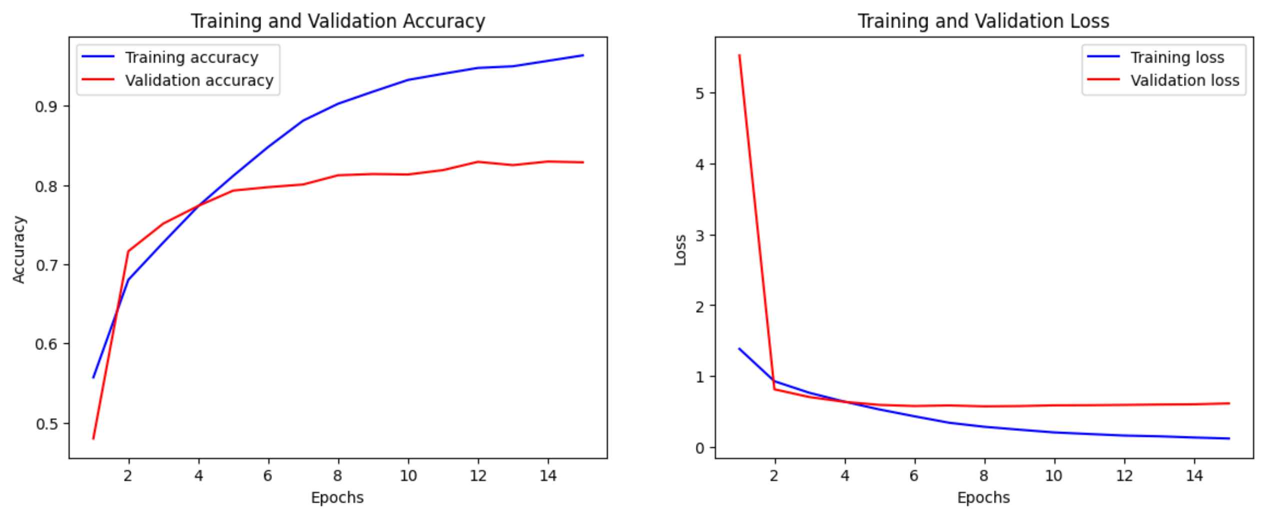
1. CNN:



The training accuracy improves consistently over the epochs, reaching ~96% by the final epoch, which indicates the model has learned the training data well.

The validation accuracy remains stable around 55–57% and does not improve significantly. This could indicate overfitting, where the model performs well on training data but struggles with unseen validation data.

2. Efficient NetB2:

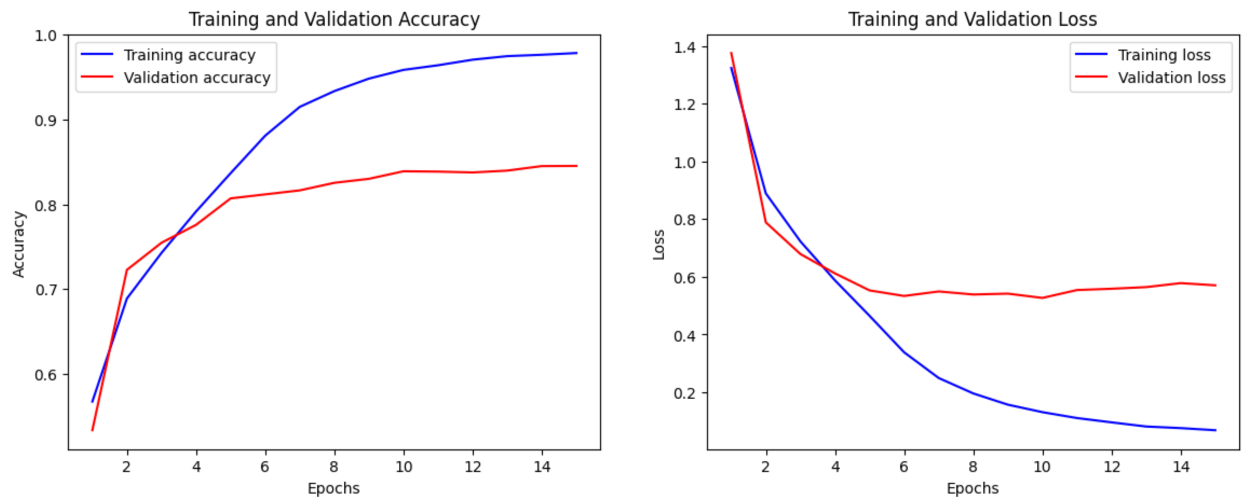


Training Accuracy: The accuracy steadily increases and approaches a high value by the final epoch, indicating the model effectively learns from the data.

Validation Accuracy: It initially increases but plateaus and slightly fluctuates in the middle epochs. This could be a sign of slight overfitting as the training accuracy continues to improve.
Training Loss: Consistently decreases, indicating the model converges during training.

Validation Loss: After an initial drop, it increases slightly in the middle and later epochs, which supports the possibility of overfitting.
Insight: The model learns well initially, but additional techniques like regularization or early stopping might improve generalization and reduce overfitting.

3. Efficient NetB3:

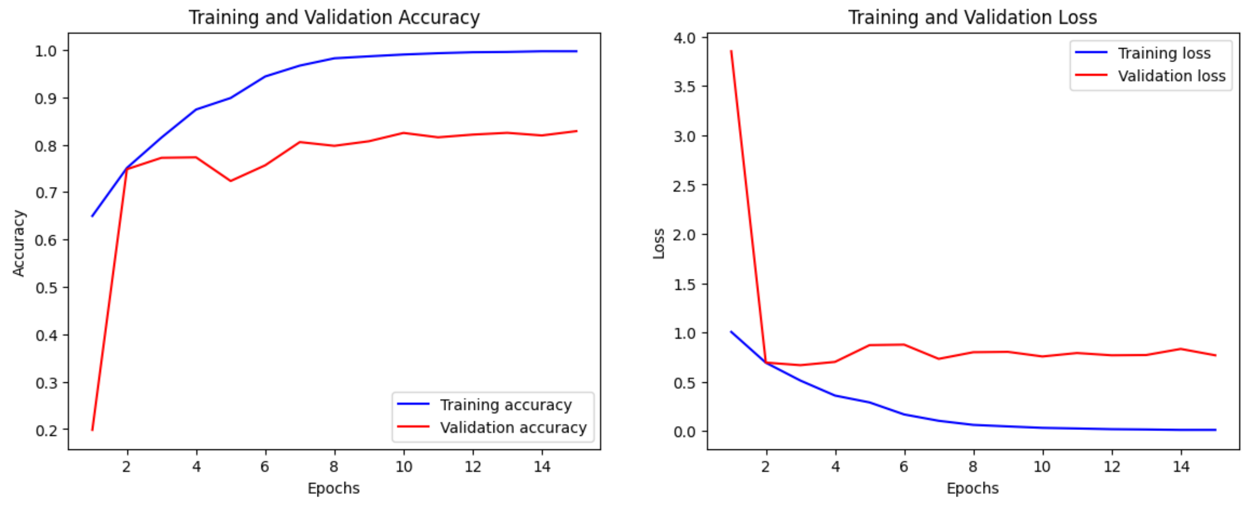


Training Accuracy: Shows steady improvement and surpasses EfficientNet-B2 in terms of peak training accuracy, demonstrating its greater capacity.

Validation Accuracy: Similar behavior to EfficientNet-B2 but maintains slightly better accuracy for most epochs, reflecting its higher model complexity.
Training Loss: Continues to decrease significantly, showing the model fits the training data well.
Validation Loss: Exhibits a similar trend to EfficientNet-B2, but the fluctuations are milder, showing marginally better generalization.

Insight: EfficientNet-B3 offers better performance than EfficientNet-B2 but still encounters slight overfitting. It might benefit from additional data augmentation or fine-tuning.

4. ResNet50:



Training Accuracy: Increases consistently, but slower than EfficientNet models in the initial epochs, reflecting ResNet's deep architecture and possibly slower convergence.

Validation Accuracy: Initially grows well but plateaus earlier than the EfficientNet models. This may indicate the model's capacity isn't fully utilized or additional fine-tuning is needed.
Training Loss: Decreases steadily and reaches a low value, demonstrating that the model learns effectively on the training set.
Validation Loss: Shows a steady decline with smaller fluctuations, suggesting better generalization compared to the EfficientNet models.

Insight: ResNet-50 generalizes slightly better compared to EfficientNet-B2 and B3, but its initial learning is slower. This could be improved with learning rate scheduling or other optimizations.

Comparison:

1. Training vs Validation Performance: EfficientNet models show stronger overfitting trend as their validation loss diverges from training loss, RESNET-50 demonstrates better balances between training and validation metrics
2. Convergence: EfficientNet models converge Faster at start due to their architecture, ResNet-50 being a deeper network, takes longer to achieve similar accuracy levels

References:

1.https://challenge.isic-archive.com/landing/2019/

2.https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/30819141/91580234-d178-41ba-bba6-d9d7904061a7/Skin-Lesion-Paper.pdf

3.https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000

4.https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T