Configuration Manual

Advancing Malware Detection: A Deep Learning Approach with Transfer learning Techniques

MSC Cyber Security

Littletresa George

Student ID: X23233346

School of Computing

National College of Ireland

Supervisor: Khadijia Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Littletresa George
Student ID:	X23233346……………………………………………………………………
Programme:	MSC Cyber Security……………………… **Year:** 2024………………………..
Module:	MSC Research Project……………………………………………………………….…
Lecturer:	Khadijia Hafeez…………………………………………………….…
Submission Due Date:	12-12-2024………………………………………………………………………….……
Project Title:	Advancing Malware Detection:A Deep Learning Approach With Transfer Learning Techniques……………………………………………………
Word Count:	…………………………………… **Page Count:** ……………………………

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: [illegible] ……………………………………………………………………………

Date: 12-12-2024………………………………………………………………………………………………

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

Littletresa George
X23233346

1 Introduction

This configuration guide outlines the steps to build and execute a malware classification project from the ground up. The project employs convolutional neural network (CNN) architectures such as Xception, EfficientNetB0, and ResNet50 to categorize malware images into 25 different classes utilizing the MaleVis_split dataset. This manual addresses dataset preparation, environment configuration, model training, and performance evaluation.

2 System Specifications

2.1. Hardware

- **Google Colab Cloud Environment**
    - **RAM:** 12.7 GB
    - **GPU:** NVIDIA Tesla T4 with 15.0 GB memory
    - **Disk Space:** 112.6 GB

2.2. Software

- **Python Version:** Python 3 (Google Compute Engine backend)
- **Libraries:**
    - TensorFlow 2.x
    - NumPy
    - Matplotlib
    - Seaborn
    - Transformers
    - Scikit-learn

Additional Tools: Google Drive (for dataset storage and model persistence).

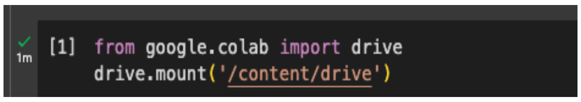
3. Step-by-Step Instructions to Build the Artifact

3.1. Setting Up the Environment

1. Access Google Colab:
   Open Google Colab.
   Create a new notebook.
2. Enable GPU Acceleration:
   Navigate to Runtime > Change Runtime Type.
   Set Hardware Accelerator to GPU.
3. Install Required Libraries:
   Run the following command to ensure all dependencies are installed

```
!pip install tensorflow numpy matplotlib seaborn
transformers scikit-learn
```

Mount Google Drive to Colab:

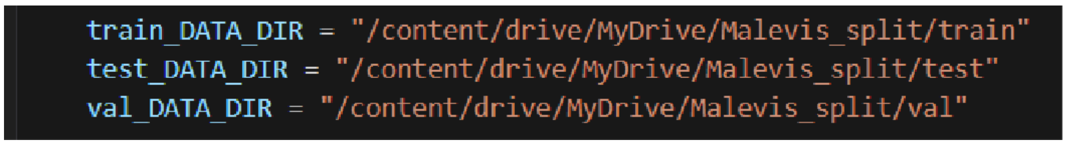


```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

4. Machine Learning Pipeline

4.1 Dataset Loading

Download the MaleVis_split Dataset:
Obtain the dataset from the official source or project repository.
Upload the dataset to your Google Drive
Ensure the dataset folders (train, test, and val) are organized in your Google Drive.
Place them in a dedicated project folder,
e.g: /content/drive/MyDrive/Malware_Classification_Project/.
Verify Dataset Structure:
Ensure the dataset folders (train, test, and val) are stored in Google Drive and update paths in the script accordingly:



```
train_DATA_DIR = "/content/drive/MyDrive/Malevis_split/train"
test_DATA_DIR = "/content/drive/MyDrive/Malevis_split/test"
val_DATA_DIR = "/content/drive/MyDrive/Malevis_split/val"
```

4.2Dataset Preprocessing steps

The dataset used is the MaleVis_split dataset, containing malware images split into train, test, and validation folders.

Organize dataset into respective directories:
train_DATA_DIR: Contains training images.
test_DATA_DIR: Contains testing images.
val_DATA_DIR: Contains validation images.
Balance Dataset (Remove excess images from classes like Other to balance the dataset. It is optional according to the dataset).
Resize images to a target size of 200x200 pixels for consistent input size.

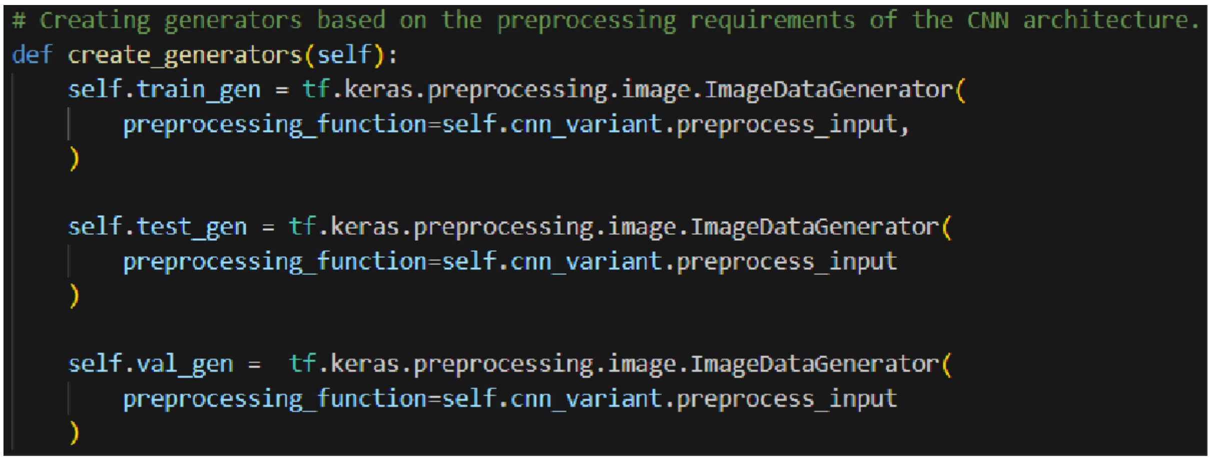
4.3Data Loading and Augmentation

- Use the ImageDataGenerator class from TensorFlow for data augmentation and preprocessing:

  To improve the dataset's diversity and reduce the risk of overfitting, we employ TensorFlow's ImageDataGenerator for data augmentation. The augmentations applied to the training dataset consist of rescaling, rotations, shifts in width and height, zooming, and horizontal flips. For evaluation, only rescaling is applied to the test and validation datasets.

Steps:

1. **Apply Augmentation to Training Data**:
    - Augmentations such as rotation, zoom, and flipping are applied to improve model robustness.
2. **Preprocess Testing and Validation Data**:
    - Images are rescaled to normalize pixel values (range 0-1).



```
# Creating generators based on the preprocessing requirements of the CNN architecture.
def create_generators(self):
    self.train_gen = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=self.cnn_variant.preprocess_input,
    )

    self.test_gen = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=self.cnn_variant.preprocess_input
    )

    self.val_gen =  tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=self.cnn_variant.preprocess_input
    )
```

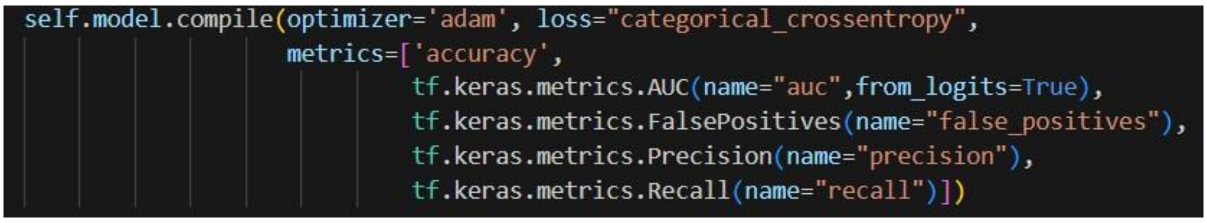
4.4 Model Training and Execution steps

To successfully classify malware images, we utilize three advanced CNN architectures: Xception, EfficientNetB0, and ResNet50. These models are refined for the MaleVis dataset by employing pre-trained weights while retraining solely the top layers.
`

Steps:

1. **Create a Generic Model Builder Function**:
    - This function takes the base architecture (Xception, EfficientNetB0, or ResNet50) and adds a global average pooling layer, a dense hidden layer, and a final softmax output layer for classification.
2. **Compile the Model**:
    - The model is compiled with the Adam optimizer, categorical crossentropy loss, and accuracy as the evaluation metric.
3. **Train the Model**:
    - The model is trained using the augmented training data and validated on the validation data for a specified number of epochs.

Fine-tune each model by freezing the earlier layers and training only the top layers:



```
self.model.compile(optimizer='adam', loss="categorical_crossentropy",
                   metrics=['accuracy',
                            tf.keras.metrics.AUC(name="auc",from_logits=True),
                            tf.keras.metrics.FalsePositives(name="false_positives"),
                            tf.keras.metrics.Precision(name="precision"),
                            tf.keras.metrics.Recall(name="recall")])
```

4.5 Model testing:

To ensure the performance of the trained models is accurately measured, we evaluate them on the test dataset. This involves calculating the test accuracy, generating a classification report, and visualizing the confusion matrix for detailed insights into the predictions.

Steps:

Evaluate the Model on Test Data:

- Use the model.evaluate() function to compute the loss and accuracy on the test dataset.

Generate Predictions:

- Obtain predictions for the test dataset using model.predict().

Generate Classification Report and Confusion Matrix:

- Use Scikit-learn's tools to generate a classification report and visualize the confusion matrix.

Explanation:

Test Accuracy:

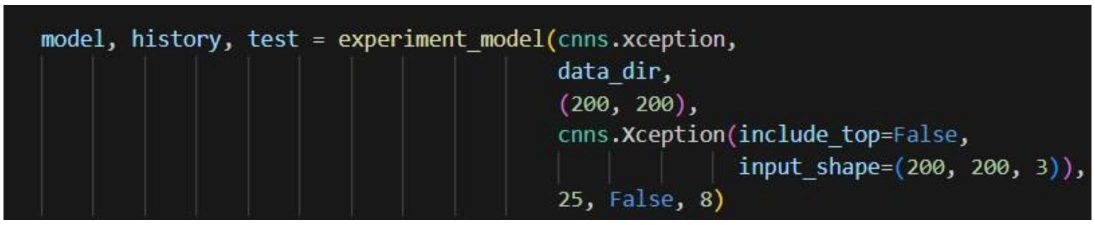
- The model.evaluate() function provides the overall accuracy and loss on unseen test data, ensuring no data leakage from training or validation.

Classification Report:

- The classification report includes precision, recall, and F1-score for each class, offering detailed performance metrics.

Confusion Matrix:

- The confusion matrix provides a grid visualization of true versus predicted classes, helping identify where the model misclassifies data.



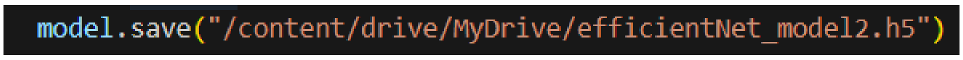
```
model, history, test = experiment_model(cnns.xception,
                                         data_dir,
                                         (200, 200),
                                         cnns.Xception(include_top=False,
                                                       input_shape=(200, 200, 3)),
                                         25, False, 8)
```

4.5 Save and Reload Models

We save the trained models to preserve them for future use or deployment. These saved models can be reloaded anytime to make predictions or resume training without retraining from scratch.

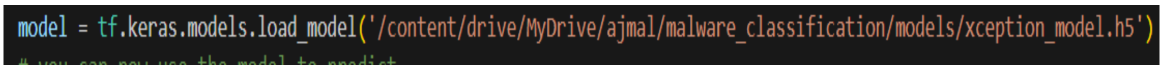
Steps:

1. **Save the Trained Model**:
   - Use TensorFlow's model.save() function to save the model in Google Drive or a specified location.
2. **Reload the Saved Model**:
   - Use TensorFlow's load_model() function to load the previously saved model for inference or further training.



```
model.save("/content/drive/MyDrive/efficientNet_model2.h5")
```

Model loading:



```
model = tf.keras.models.load_model('/content/drive/MyDrive/ajmal/malware_classification/models/xception_model.h5')
```

Model predicting:

Once the model is reloaded, it can be used to make predictions on new input data.

```
me.predict()
```

5 Results

Visualizing the results of the training process and evaluation metrics provides insights into the model's performance. It helps identify trends such as overfitting, underfitting, or areas where the model can be improved.

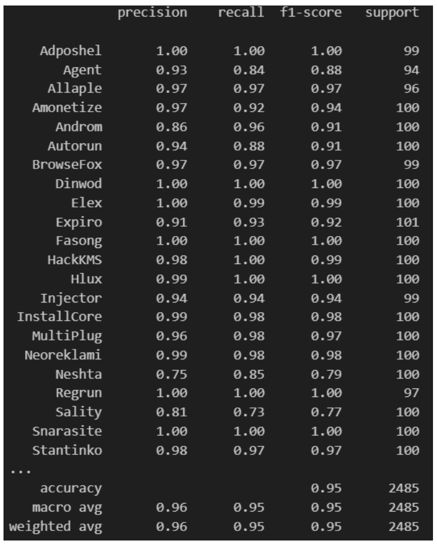
Steps:

1. Plot Training and Validation Accuracy:
   Compare training and validation accuracy across epochs to monitor learning progress.
2. Plot Training and Validation Loss:
   Analyze how the model's loss evolves during training to identify convergence or overfitting.
3. Visualize Performance Metrics:
   Use bar plots or tables to compare the accuracy, precision, recall, and F1-scores of different models.

We evaluated the performance of three deep learning models: EfficientNet B0, ResNet50, and Xception. For each model, the following evaluation metrics were analysed.

1. Classification Report:
   - Includes key performance metrics such as precision, recall, F1-score, and support for each class.
   - These metrics provide insights into the model's ability to correctly classify instances and handle imbalanced data.
2. Confusion Matrix:
   - Visual representation of the classification performance, showing true positives, false positives, true negatives, and false negatives for each class.
   - It highlights the specific areas where the models performed well and where misclassifications occurred.

5.2Xception Model



```
              precision    recall  f1-score   support

    Adposhel       1.00      1.00      1.00        99
       Agent       0.93      0.84      0.88        94
     Allaple       0.97      0.97      0.97        96
   Amonetize       0.97      0.92      0.94       100
      Androm       0.86      0.96      0.91       100
     Autorun       0.94      0.88      0.91       100
   BrowseFox       0.97      0.97      0.97        99
      Dinwod       1.00      1.00      1.00       100
        Elex       1.00      0.99      0.99       100
      Expiro       0.91      0.93      0.92       101
      Fasong       1.00      1.00      1.00       100
     HackKMS       0.98      1.00      0.99       100
        Hlux       0.99      1.00      1.00       100
    Injector       0.94      0.94      0.94        99
 InstallCore       0.99      0.98      0.98       100
   MultiPlug       0.96      0.98      0.97       100
  Neoreklami       0.99      0.98      0.98       100
      Neshta       0.75      0.85      0.79       100
      Regrun       1.00      1.00      1.00        97
      Sality       0.81      0.73      0.77       100
   Snarasite       1.00      1.00      1.00       100
   Stantinko       0.98      0.97      0.97       100
...
    accuracy                           0.95      2485
   macro avg       0.96      0.95      0.95      2485
weighted avg       0.96      0.95      0.95      2485
```

Figure 2: Classification report of Xception model

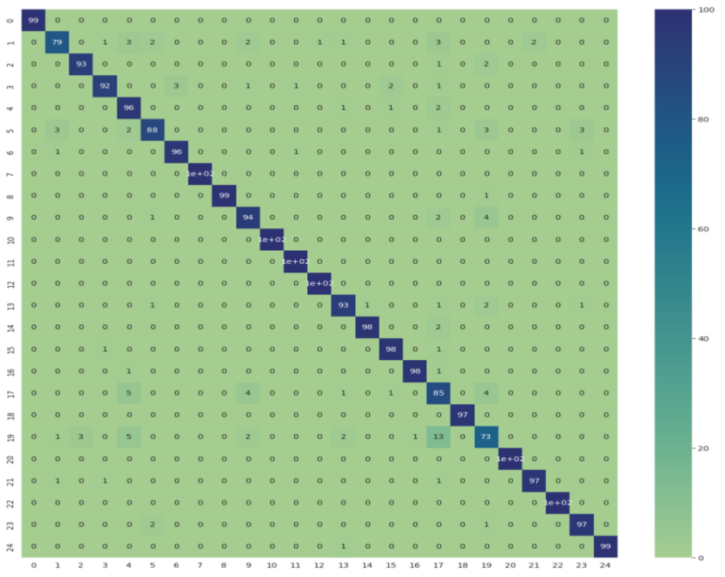
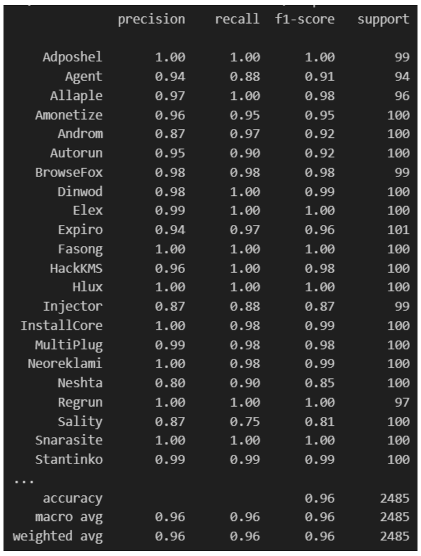


Figure 1: Confusion matrix of Xception Model

5.3EfficientNetB0 Model



```
              precision    recall  f1-score   support

    Adposhel       1.00      1.00      1.00        99
       Agent       0.94      0.88      0.91        94
     Allaple       0.97      1.00      0.98        96
   Amonetize       0.96      0.95      0.95       100
      Androm       0.87      0.97      0.92       100
     Autorun       0.95      0.90      0.92       100
  BrowseFox       0.98      0.98      0.98        99
      Dinwod       0.98      1.00      0.99       100
        Elex       0.99      1.00      1.00       100
      Expiro       0.94      0.97      0.96       101
      Fasong       1.00      1.00      1.00       100
    HackKMS       0.96      1.00      0.98       100
        Hlux       1.00      1.00      1.00       100
    Injector       0.87      0.88      0.87        99
 InstallCore       1.00      0.98      0.99       100
   MultiPlug       0.99      0.98      0.98       100
  Neoreklami       1.00      0.98      0.99       100
      Neshta       0.80      0.90      0.85       100
      Regrun       1.00      1.00      1.00        97
      Sality       0.87      0.75      0.81       100
   Snarasite       1.00      1.00      1.00       100
   Stantinko       0.99      0.99      0.99       100
...
    accuracy                           0.96      2485
   macro avg       0.96      0.96      0.96      2485
weighted avg       0.96      0.96      0.96      2485
```

Figure 4: Classification report of EfficientNetB0 Model

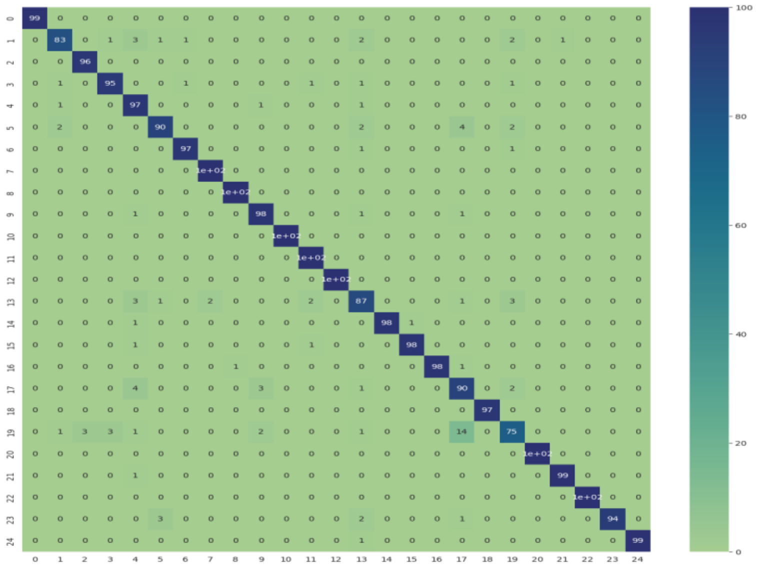
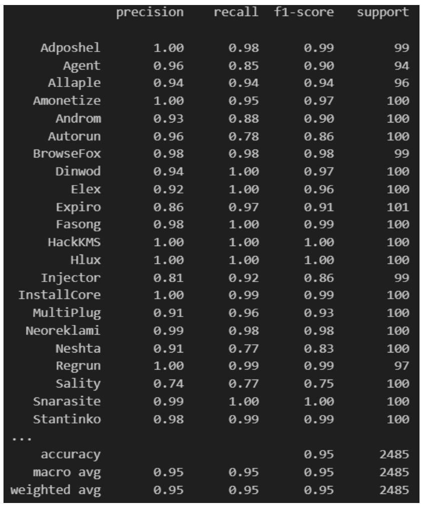


Figure 3: Confusion Matrix of EfficientNetB0

5.4ResNet50 Model



```
              precision    recall  f1-score   support

    Adposhel       1.00      0.98      0.99        99
       Agent       0.96      0.85      0.90        94
     Allaple       0.94      0.94      0.94        96
   Amonetize       1.00      0.95      0.97       100
      Androm       0.93      0.88      0.90       100
     Autorun       0.96      0.78      0.86       100
   BrowseFox       0.98      0.98      0.98        99
      Dinwod       0.94      1.00      0.97       100
        Elex       0.92      1.00      0.96       100
      Expiro       0.86      0.97      0.91       101
      Fasong       0.98      1.00      0.99       100
     HackKMS       1.00      1.00      1.00       100
        Hlux       1.00      1.00      1.00       100
    Injector       0.81      0.92      0.86        99
 InstallCore       1.00      0.99      0.99       100
   MultiPlug       0.91      0.96      0.93       100
  Neoreklami       0.99      0.98      0.98       100
      Neshta       0.91      0.77      0.83       100
      Regrun       1.00      0.99      0.99        97
      Sality       0.74      0.77      0.75       100
   Snarasite       0.99      1.00      1.00       100
   Stantinko       0.98      0.99      0.99       100
...
    accuracy                           0.95      2485
   macro avg       0.95      0.95      0.95      2485
weighted avg       0.95      0.95      0.95      2485
```

Figure 5: Classification report of ResNet50 model

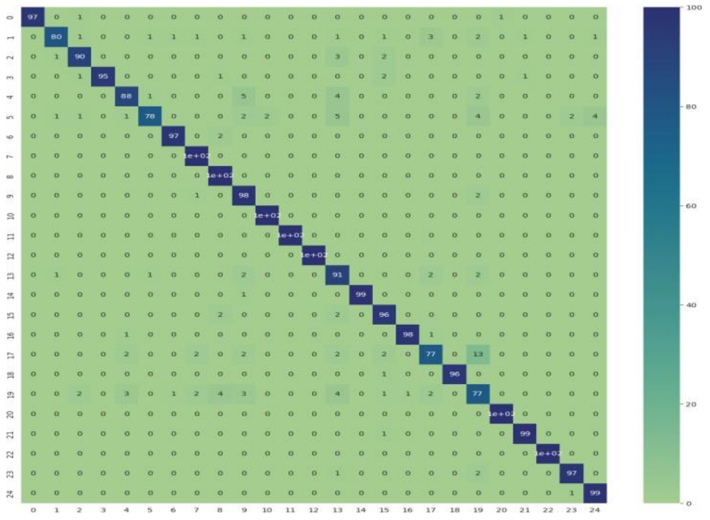


Figure 6: Confusion matrix of ResNet50 model

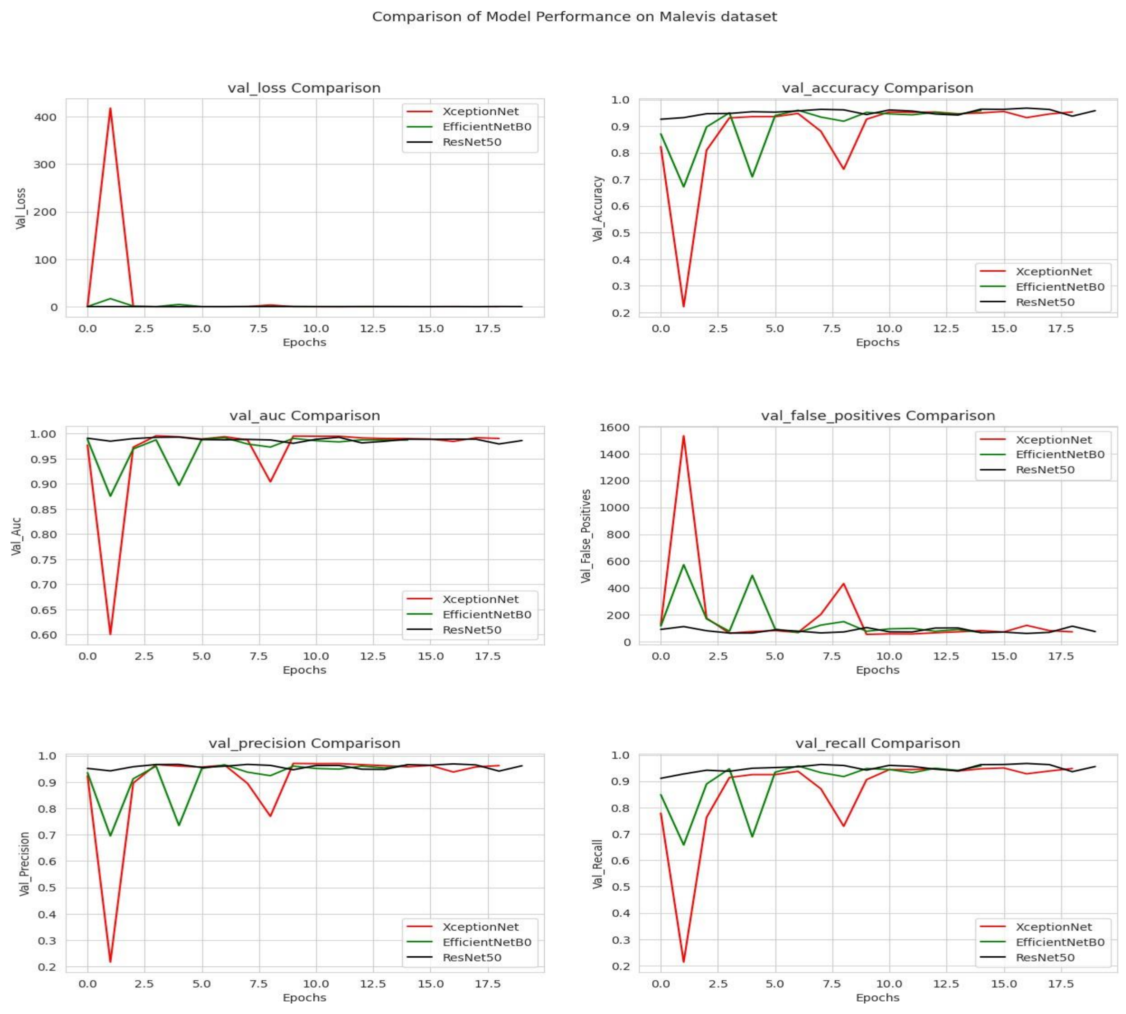


Figure 7: Compare training performance of 3 models

These above graphs show how well three different models Xception, EfficientNetB0, and ResNet50 performed on the task of classifying malware images. The key points to understand are:

1. Validation Loss: How much the model's predictions are wrong. Lower and steady loss means better performance.
2. Validation Accuracy: The percentage of correct predictions. Higher accuracy means better results.
3. Validation AUC: How well the model can distinguish between different categories. Higher AUC means better classification.
4. False Positives: How often the model incorrectly labels something as malware. Fewer false positives are better.

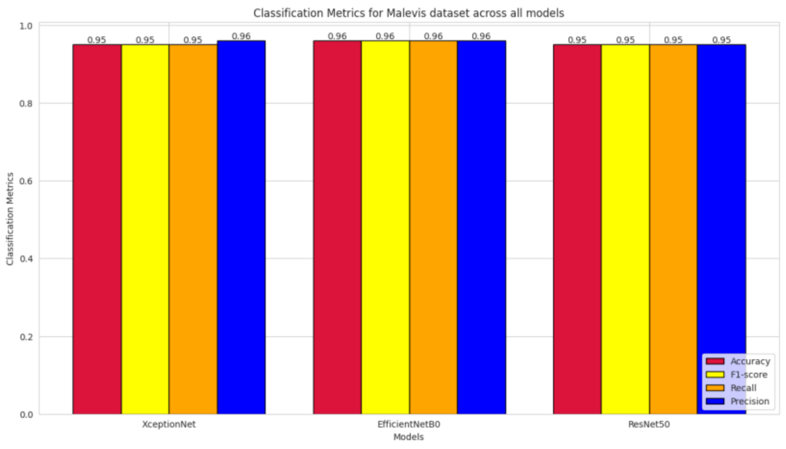


Figure 8: Evaluation metrics comparison of models

The bar chart above presents the evaluation metrics Accuracy, F1-score, Recall, and Precision for different deep learning models (XceptionNet, EfficientNetB0, ResNet50) tested on the Malevis dataset.

5. Precision & Recall:
- Precision: How often the model's malware predictions are correct.
- Recall: How good the model is at finding all the malware.

6. F1-Score:
- Balances Precision and Recall, giving a single measure of the model’s overall performance.
- A higher F1-score indicates both good accuracy in predicting malware and good coverage in identifying all malware cases.

7. Accuracy:
- Measures how often the model’s predictions are correct overall.
- A higher accuracy means the model correctly classified a large proportion of all images, both malware and non-malware.

References

- TensorFlow Documentation: https://www.tensorflow.org/
- Google Colab: https://colab.research.google.com/