Objective

To develop a machine learning model to classify images of items/industrial equipment into two categories:

- Defective
- Non-Defective

Bonus Objective

Optimize the model for hardware-accelerated inference (e.g., using GPU, TPU, NPU, FPGA, etc.).

The Solution

The solution involves loading the dataset, preprocessing the data, fine-tuning EfficientNet for the purpose of binary classification, saving the model, and developing an inference script to allow the model to perform inference on a test dataset of the user's choice.

The inference script loads the fine-tuned model using the saved weights and biases of the model and recreating its architecture, then exports the model to ONNX runtime and finally performs inference via 2 models-the original fine-tuned model, loaded in evaluation mode in Pytorch, and the exported ONNX model. The script also includes portions for profiling for time and memory usage during inference.

The inference script allows the user to enter the path to the saved model's weights and biases, input the intended write destination file path for storing the ONNX model, and input the file path to the test dataset on which inference should be run.

In the process of exporting the Pytorch model to ONNX runtime and saving it, some basic optimizations were enabled(built-in ONNX optimizations) to allow accelerated inference on GPUs.

A Dockerfile and requirements.txt file has also been included to help build a Docker image from the inference.py and allow the script to run on Docker containers.

(The fine-tuned model considered as the final solution, is the one that was fine-tuned in Iteration Two. And this same model is meant to be later used by the inference.py script as well. The files related to the initial training and evaluation of the fine-tuned model, can be found in the IterationTwo folder, whereas the inference folder contains all the files related to inference.)

Dataset

The dataset chosen for the task is available at [Kolektor Surface-Defect Dataset 2 (KolektorSDD2/KSDD2) | ViCoS Lab](). It has the following properties:

- 356 images with visible defects

- 2979 images without any defect
- image sizes of approximately 230 x 630 pixels
- train set with 246 positive and 2085 negative images
- test set with 110 positive and 894 negative images
- several different types of defects (scratches, minor spots, surface imperfections, etc.)

While developing the model, the same train-test split has been used as described above, although further validation splits were used during the process of fine-tuning itself.

Dataset Preparation

Loaded defect detection images and their corresponding ground truth (GT) label images. Also, filtered non-GT images for clarity. PIL was used for smooth image handling, and if no GT label image was found, defaulted to "no defect" label.

Resized images to 224×224 (standard for EfficientNet) and applied normalization for faster and better convergence.

Used weighted random sampling to deal with class imbalance effectively. It computes class weights based on inverse class frequencies to ensure that the minority class is not under-represented during the fine-tuning process.

Fine-tuning

Fine-tuning was performed with the original "train" split of the original dataset, containing 246 positive(containing a defect) and 2085 negative(without any defect) images, as well as their corresponding ground_truth label images.

Pre-trained Model: Used EfficientNet B0, and only the last layer was fine-tuned to adjust for binary classification.
Unified Focal Loss: Used a custom loss function that combines **focal loss** and **dice loss** helps with imbalanced datasets and improves the robustness of the model.
Dice score measures how much the predicted and actual objects overlap. It treats small and large objects the same: if you correctly predict half of an object, your score is 0.5, regardless of whether the object is small or large.
Meanwhile, focal loss handles class imbalance effectively by focusing on harder examples.
Cross-validation: Used 5-fold cross validation. Identified the best model based on highest F1 score, and computed training vs validation accuracy, loss, and F1-score after every epoch.
Optimization: Used the AdamW optimizer with a learning rate of 0.0001 for fine-tuning. In AdamW, weight decay is decoupled from the gradient update.

Evaluation on Test Set

Evaluation was first performed in a Jupyter Notebook on a Kaggle P100 GPU. The test dataset used was the original test split of the original dataset, containing 110 positive(defect) and 894 negative(non-defect) images. No data leakage occurred during the fine-tuning process, ie, the model was not exposed to any of the test data during the fine-tuning process.

Results

Accuracy: 0.9671
Precision: 0.9873
Recall: 0.7091
F1 Score: 0.8254

As we can see, the fine-tuned model has high accuracy(ratio of correctly classified instances in both classes, compared to all instances) which indicates strong overall performance.
The fine-tuned model also achieves high precision, which means that the model is highly reliable when it actually classifies an image as defective. False positives are very less.

With a recall of nearly 71%, the model has moderately good performance in reducing false negatives.
The F1-score of 82.54% indicates that a good balance between precision and recall has been reached.

Here is the confusion matrix: