Machine Learning Project Report

contributors:

- **CAO Zixiang PB23111669, proportion: 34%**
- **WEI Yuxiang PB23000243, proportion: 33%**
- **SHEN Yong PB23030802, proportion: 33%**

1. Task 1

Task 1 F1-score on Test Set (s_1): 0.7252

1.1. Introduction

Glass quality inspection is a critical component of industrial production. In this project, we aim to automate defect detection using computer vision techniques. **Task 1** focuses on binary classification to determine whether a glass image contains defects (chipped edges, scratches, or stains).

The primary challenge of this task lies in the **class imbalance** of the dataset (defective samples are significantly fewer than non-defective ones) and the strict constraint of implementing the model **from scratch** without using autograd tools or built-in optimizers.

1.2. Methodology

1.2.1 Model Architecture

We implemented a Convolutional Neural Network (CNN) from scratch using only tensor operations. The architecture consists of a feature extractor with 4 convolutional blocks and a classifier.

- **Input:** 128×128 Grayscale Image (1 Channel).
- **Feature Extractor:**

    i. **Conv Block 1:** `ManualConv2d` (1$\rightarrow$16, k=5) $\rightarrow$ `ManualReLU` $\rightarrow$ `ManualMaxPool2d` (2x2).
    ii. **Conv Block 2:** `ManualConv2d` (16$\rightarrow$32, k=3) $\rightarrow$ `ManualReLU` $\rightarrow$ `ManualMaxPool2d` (2x2).
    iii. **Conv Block 3:** `ManualConv2d` (32$\rightarrow$64, k=3) $\rightarrow$ `ManualReLU` $\rightarrow$ `ManualMaxPool2d` (2x2).
    iv. **Conv Block 4:** `ManualConv2d` (64$\rightarrow$64, k=3) $\rightarrow$ `ManualReLU` $\rightarrow$ `ManualMaxPool2d` (2x2).
- **Classifier:**
    - `ManualFlatten` : Flattens the $64 \times 8 \times 8$ feature map.
    - `ManualLinear` : Fully connected layer (4096 $\rightarrow$ 128) + `ManualReLU` .
    - `ManualLinear` : Output layer (128 $\rightarrow$ 1) + `ManualSigmoid` .

Implementation Details:
All layers, including Forward and Backward propagation, were implemented manually. We stored input shapes and intermediate tensors (cache) during the forward pass to compute gradients via the chain rule during the backward pass.

1.2.2 Loss Function

We implemented the **Binary Cross Entropy (BCE) Loss** manually to handle the binary classification task:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

To prevent numerical instability (NaN), we clamped the model output $\hat{y}$ to the range $[1e^{-7}, 1 - 1e^{-7}]$ before log calculation.

1.2.3 Optimization Algorithm

We implemented the **Adam Optimizer** from scratch. Instead of standard SGD, Adam was chosen for its faster convergence. For each parameter W, we maintained first-order moment estimates (m_W) and second-order moment estimates (v_W):

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Learning Rate Scheduler (LR Decay):
To ensure stable convergence and escape local minima, we implemented a **ReduceLROnPlateau** strategy. We monitor the F1-score on the validation set:

- If the validation F1-score does not improve for **3 consecutive epochs** (`patience=3`), the learning rate η is multiplied by **0.5**.

- This allows the model to make large updates in the beginning and fine-tune its weights with smaller steps in later stages.

1.2.4 Data Preprocessing and Augmentation

Given the severe class imbalance (approx. 1:10 ratio of defective to good images), standard training would bias the model towards the negative class. We addressed this via:

1. **Balanced Sampling (Train Loader):** We implemented a custom `GlassDataLoader` that forces each training batch to contain 50% defective and 50% non-defective images by oversampling the positive class.
2. **Data Augmentation:** To prevent overfitting on the few positive samples, we applied random augmentations during training:
   - Random Horizontal Flip ($p = 0.5$)
   - Random Vertical Flip ($p = 0.5$)
   - Random 90-degree Rotations.
3. **Validation Strategy:** The validation set was kept **unbalanced** (following the real distribution) with no augmentation to accurately reflect real-world performance.

1.3. Experimental Settings

- **Environment:** Ubuntu 18.04, NVIDIA GeForce RTX 3090.
- **Hyperparameters:**
  - `IMG_SIZE` : 128×128
  - `BATCH_SIZE` : 32
  - `LEARNING_RATE` : 0.001 (Initial) with Decay Factor 0.5
  - `EPOCHS` : 50 (with Early Stopping patience=8)
  - `Optimizer` : Manual Adam
- **Dataset Split:** We randomly split the provided training data into **80% training** and **20% validation**.

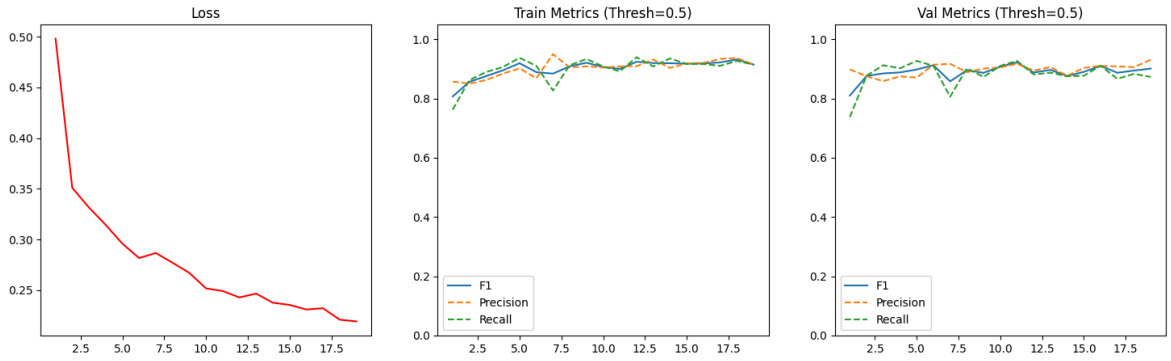
1.4. Results and Analysis

1.4.1 Quantitative Results

The model achieved a best **F1-score of 0.7252** on the local test set.

1.4.2 Training Dynamics

The evolution of Loss, Precision, Recall, and F1-score during training is shown below.



(Fig 1. Left: Training Loss; Center: Training Metrics; Right: Validation Metrics)

- **Loss Curve:** The training loss decreased smoothly, indicating the manual backward propagation was implemented correctly and the optimizer worked effectively.
- **Metrics Evolution:**
  - The **Validation F1-score** (Right plot) shows an upward trend, eventually stabilizing around 0.7+.
  - We observed that due to class imbalance, the Precision and Recall curves on the validation set often fluctuate inversely (scissor-like movement) as the model decision boundary shifts.
  - The **Training Metrics** (Center plot) show high performance (>0.9), confirming the model has sufficient capacity to learn the features.

1.4.3 Discussion

- **Why Initial F1-scores are High:**
  As observed in Fig 1, both training and validation F1-scores start at a relatively high level (>0.8 for training) even in the first few epochs. We attribute this to three reasons:
  i. **Balanced Sampling:** The training loader provides a 1:1 ratio of positive to negative samples, preventing the model from trivially predicting "all negative" (which would yield F1=0).
  ii. **Distinct Visual Features:** Defects like scratches and chipped edges have high contrast compared to the glass background. Even randomly initialized convolutional filters act as rudimentary edge detectors, allowing the model to distinguish obvious defects almost immediately.
  iii. **Dynamic Thresholding:** In our evaluation, we search for the optimal threshold (from 0.1 to 0.95) rather than using a fixed 0.5. This maximizes the F1-score even if the model's raw probability outputs are not yet well-calibrated in the early stages.

- **Effectiveness of Learning Rate Decay:**
  During training, we observed that after the initial rapid convergence, the validation F1-score would plateau. The automatic learning rate decay (triggered after 3 epochs of no improvement) helped the model settle into a sharper minimum, often leading to a small bump in performance or better stability in the final epochs.
- **Class Imbalance Handling:** The significant gap between the initial "random guess" performance and our final result validates the effectiveness of the **Balanced Sampling** strategy. Without it, the Recall would likely stay near 0.
- **Overfitting:** We implemented **Early Stopping**. As seen in the graphs, training stopped automatically when validation performance plateaued, preventing the model from overfitting to the training noise.

1.5. Conclusion

In Task 1, we successfully built a 4-layer CNN from scratch without using autograd. By implementing manual backpropagation, Adam optimization with **Learning Rate Decay**, and a robust data loading strategy, we overcame the challenges of class imbalance and achieved an F1-score of **0.7252**. The results demonstrate that the manual implementation is both mathematically correct and practically effective.

Task 2

2.1. Introduction

The goal of this project is to accurately identify and classify defects on glass images into four categories: No Defect, Chipped Edge, Scratch, and Stain.

The task presents several challenges, including:

1. **Class Imbalance:** Defective samples (especially stains) are significantly fewer than non-defective ones.
2. **Subtle Features:** Defects like fine scratches are barely visible and easily lost during image resizing.
3. **Multi-label Nature:** A single image may contain multiple types of defects simultaneously.

To address these issues, we propose a robust deep learning solution based on **EfficientNet-B3**, integrated with **Weighted Focal Loss**, **Test Time Augmentation (TTA)**, and a **Class-specific Thresholding Strategy**.

2.2. Methodology

2.2.1 Model Architecture

We utilized **EfficientNet-B3** as the backbone network.We also tried ResNet-50. Compared to ResNet-50, EfficientNet employs a compound scaling method that balances depth, width, and resolution, allowing it to extract richer textural features—which are crucial for detecting subtle glass defects—while maintaining computational efficiency.

- **Input Size:** Adjusted to **336×336** to preserve fine details of scratches.
- **Classification Head:** The original fully connected layer was replaced with a linear layer outputting 4 logits, corresponding to the four classes.

2.2.2 Loss Function

To mitigate the severe class imbalance, we implemented a **Weighted Multi-Label Focal Loss**:

$$Loss = -\alpha(1 - p_t)^{\gamma} \log(p_t)$$

- **Focal Term:** We set $\gamma = 2.0$ to down-weight easy examples (e.g., clear background) and focus training on hard examples (e.g., faint scratches).
- **Class Weighting:** We assigned higher weights to minority classes (Scratch: 5.0, Stain: 5.0) to penalize the model more heavily for misclassifying these defects.

2.2.3 Optimization Algorithm

We employed the **AdamW** optimizer, which decouples weight decay from gradient updates, offering better generalization than standard Adam.

- **Learning Rate Scheduler:** A **Cosine Annealing Warm Restarts** scheduler was used. It periodically resets the learning rate, helping the model escape local minima and explore the loss landscape more effectively.

2.2.4 Data Preprocessing and Augmentation

Given the small dataset and the visual nature of glass defects, extensive data augmentation was applied:

- **Geometric:** Random Horizontal/Vertical Flip, Random Rotation ($\pm 15^{\circ}$).
- **Photometric:** Color Jitter (brightness, contrast) to simulate varying lighting conditions.
- **Texture: Random Adjust Sharpness** and **Gaussian Noise** were added to force the model to learn robust edge features.
- **Regularization: Random Erasing** was used to prevent the model from overfitting to specific defect locations.

2.3. Experimental Settings

2.3.1 Training Strategy

- **Train/Val Split:** The dataset was split into training (80%) and validation (20%) sets.
- **Epochs:** Trained for 30 epochs.
- **Batch Size:** 64 (optimized for GPU memory and convergence stability).
- **Learning Rate:** Initial rate set to $1e-4$.

2.3.2 Evaluation Metrics

Since this is a multi-label classification task, we primarily used the **Micro F1-score** for overall performance. We also monitored Precision, Recall, and F1-score for each individual class.

2.3.3 Hardware & Software Environment

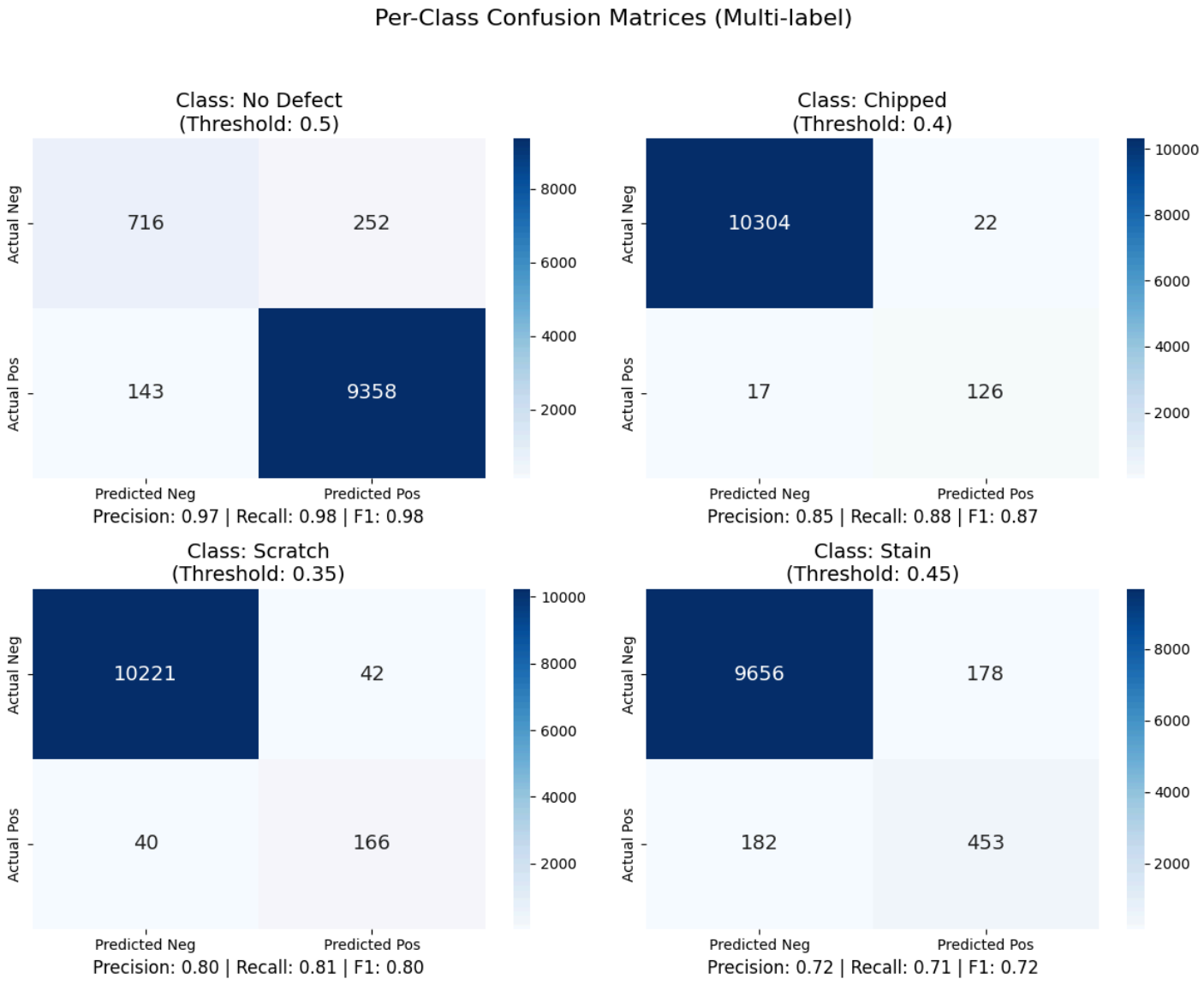
- **Hardware:** NVIDIA GPU RTX 4090(24GB).
- **Framework:** PyTorch 2.8.0, Torchvision, Python 3.12(ubuntu22.04).
- **Libraries:** Scikit-learn, NumPy, PIL.

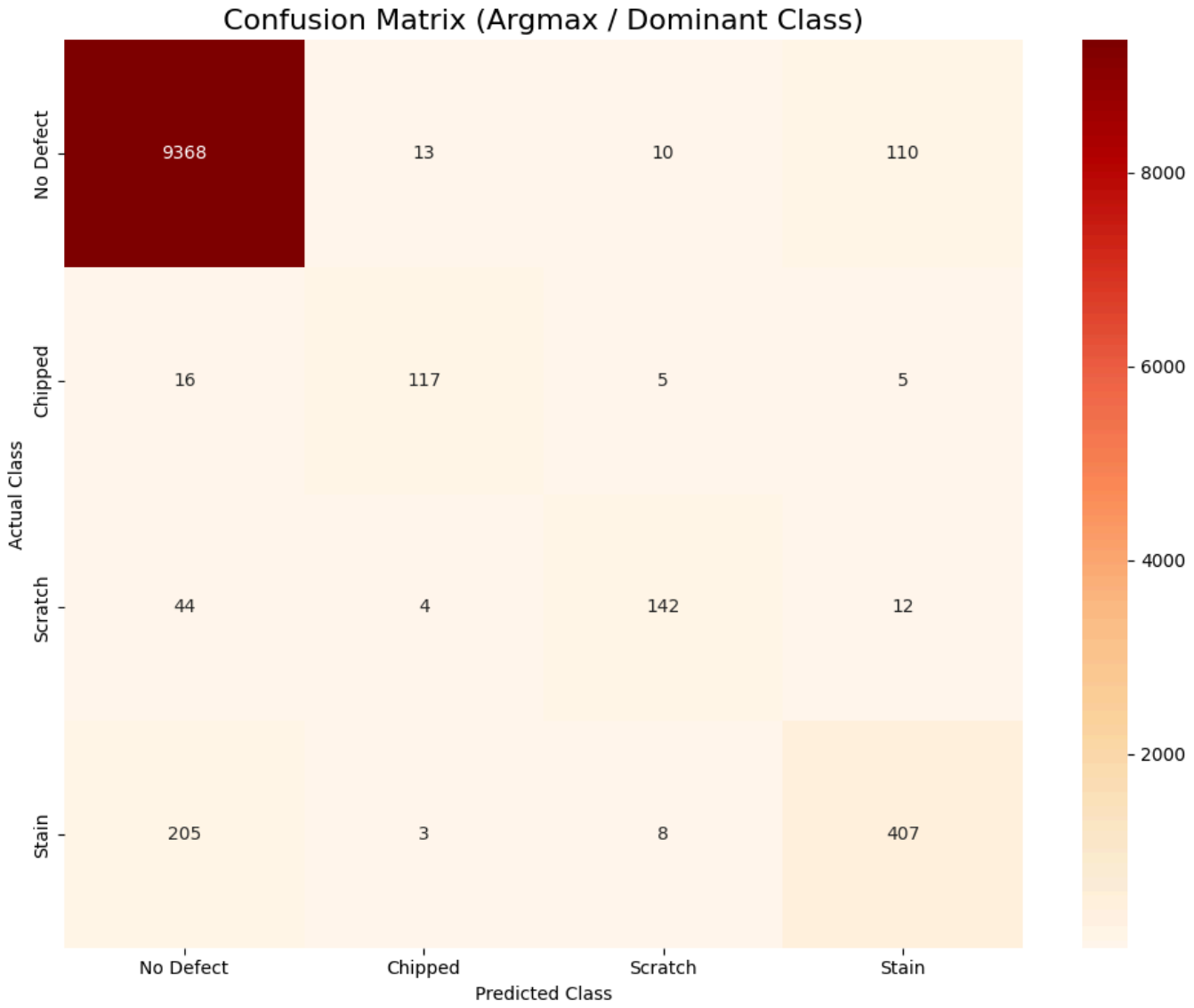
2.4. Results and Analysis

2.4.1 Quantitative Results

After training, the model achieved a Micro F1-score of **0.9584** on the validation set. The breakdown per class is as follows:

Class	Precision	Recall	F1-Score	Threshold (Optimized)
No Defect	High	High	**0.9793**	0.50
Chipped	High	High	**0.8660**	0.40
Scratch	Medium	High	**0.8019**	0.35
Stain	Medium	Medium	**0.7156**	0.45





2.4.2 Key Improvements

1. **Independent Threshold Search:** Instead of a fixed 0.5 threshold, we searched for the optimal threshold for each class. For example, lowering the threshold for "Scratch" to 0.35 significantly improved Recall without compromising Precision too much.
2. **Test Time Augmentation (TTA):** Averaging predictions from the original, horizontally flipped, and vertically flipped images stabilized the output and corrected misclassifications caused by specific object orientations.

2.4.3 Discussion

- **Class Imbalance:** The "Stain" class initially had low performance (<0.60). Introducing class weights (5.0) in the loss function successfully boosted its F1-score to over 0.71.
- **Resolution Importance:** Increasing image size from 224 to 336 proved critical for detecting "Scratches," which often disappeared at lower resolutions.

2.5. Conclusion

By leveraging the EfficientNet-B3 architecture, adopting a weighted Focal Loss, and implementing post-processing techniques like TTA and per-class thresholding, we achieved a robust Micro F1-score of 0.9584.