**Steel Surface Defect Detection: Baseline CNN and Segmentation Preprocessing**

**Authors:** Dhruv Bajaj (230365), Bipin Kumar Jaiswal (230300), Nonit Gupta (230712), Rutul Bhanushali (230884)

**1. Introduction**

Surface defects on steel, such as scratches, patches, pitting, and crazing, can significantly compromise product quality and safety. Traditional manual inspection methods are time-consuming and error-prone, particularly in high-speed production environments. By contrast, automated vision-based inspection systems offer a promising alternative, improving accuracy, reducing labor costs, and accelerating quality control processes.

In this project, we develop an image classifier to identify four types of steel surface defects: crazing, patch, pitted surface, and scratch. We begin by implementing a standard Convolutional Neural Network (CNN) as a baseline model. Subsequently, we apply traditional image-processing techniques (segmentation) to highlight probable defect regions and retrain the CNN on these preprocessed images. Our approach emphasizes clear, explainable methods—such as Canny edge detection and Otsu thresholding—over complex deep segmentation models, ensuring interpretability and simplicity.

**2. Dataset and Class Imbalance**

The dataset comprises RGB images of steel surfaces, each labeled with one of four defect types: crazing, patch, pitted surface, or scratch. An initial inspection revealed a significant class imbalance, with the "Scratch" class dominating the dataset, while the other classes are underrepresented. This imbalance, typical of real-world datasets, can bias classifiers toward the majority class, resulting in poor performance on minority classes like crazing or pitted surface.

To address this issue, we implemented the following strategies:

* **Class Weighting:** During CNN training, we utilized class\_weight='balanced' from scikit-learn, assigning higher penalties to errors on minority classes. This adjustment boosts their influence in the loss function, counteracting the bias toward the majority class.
* **Stratified Sampling (Shuffling):** We shuffled the dataset while preserving the original class proportions in each batch, ensuring that validation and test sets mirror the training distribution. This prevents bias from uneven data splits.
* **Normalization:** All images were resized to 128×128 pixels and their pixel values scaled to the range [0,1]. Normalization facilitates faster and more stable training by standardizing input values.

While we did not employ data augmentation or oversampling in this iteration, these techniques could further mitigate imbalance in future work. Given the skewed class distribution, accuracy alone is insufficient for evaluation. Thus, we also report precision, recall, and F1-score, which provide a more nuanced assessment of performance across all classes. Precision measures the fraction of correct predictions, recall indicates the fraction of true instances identified, and F1-score is their harmonic mean.

**3. Baseline CNN Classifier**

**3.1 Architecture and Theory**

We implemented a simple yet effective CNN using Keras (TensorFlow) for multi-class classification. The architecture is structured as follows:

1. **Input Layer:** Accepts 128×128 RGB images.
2. **Conv2D Layer (32 filters, 3×3 kernel, ReLU activation):** Applies 32 learnable 3×3 filters to detect low-level features like edges and textures. The ReLU activation introduces non-linearity, enabling the model to capture complex patterns.
3. **MaxPooling2D (2×2):** Reduces the spatial dimensions of feature maps by selecting the maximum value in each 2×2 region, retaining key features while lowering computational load.
4. **Conv2D Layer (64 filters, 3×3 kernel, ReLU activation):** Extracts higher-level features by combining patterns from the previous layer.
5. **MaxPooling2D (2×2):** Further downsamples the feature maps, preparing them for dense layers.
6. **Flatten Layer:** Converts the 2D feature maps into a 1D vector.
7. **Dense Layer (128 units, ReLU activation):** Integrates the extracted features into a fully connected layer.
8. **Dropout (0.5):** Randomly deactivates 50% of neurons during training to prevent overfitting.
9. **Output Layer (4 units, softmax activation):** Produces a probability distribution over the four defect classes.

This design leverages fundamental deep learning principles for vision tasks: convolutional layers detect local patterns through parameter sharing and spatial locality, pooling layers provide spatial invariance, and dense layers perform classification. We compiled the model using the Adam optimizer—a gradient-based method that adapts learning rates—and categorical cross-entropy loss, which is well-suited for multi-class problems.

**3.2 Training and Evaluation**

The CNN was trained for 10 epochs with a batch size of 32, incorporating the computed class weights to address imbalance. Performance was assessed on a held-out test set, yielding the following metrics:

| **Metric** | **Score** |
| --- | --- |
| Test accuracy | 72.00% |
| Precision | 76.27% |
| Recall | 72.50% |
| F1-Score | 73.38% |

* **Confusion Matrix Insights:** The majority class (Scratch) exhibited the highest true-positive rate, while minority classes (crazing, patch, pitted surface) showed lower recall, underscoring the persistent challenge of class imbalance despite weighting.

**4. Segmentation-Based Preprocessing**

To potentially enhance classification performance, we explored traditional computer vision techniques to preprocess images, aiming to isolate defect regions and reduce background noise before feeding them into the CNN.

**4.1 Pipeline Overview**

The segmentation pipeline consists of the following steps:

1. **Grayscale Conversion:** Converts the RGB image to a single-channel grayscale image, as defects often manifest as intensity variations rather than color differences.
2. **Canny Edge Detection (thresholds 50/150):** Identifies edges by detecting strong intensity gradients, producing a binary edge map of potential defect boundaries.
3. **Morphological Closing (3×3 kernel):** Applies dilation followed by erosion to close small gaps in the edge map, forming continuous defect outlines.
4. **Otsu Thresholding:** Automatically determines an optimal threshold to binarize the edge map, separating defect regions from the background.
5. **Dilation:** Slightly enlarges the binary mask to ensure full defect coverage.

The resulting mask is applied to the original normalized RGB image, retaining defect pixels and zeroing out the background, thus creating a defect-enhanced image.

**4.2 Theoretical Background**

* **Canny Edge Detection:** A multi-stage algorithm that detects edges by identifying strong gradients while suppressing noise. It is widely used for its reliability in outlining boundaries of defects like scratches or grooves.
* **Morphological Operations:** Closing fills gaps in edge maps, ensuring defect regions are cohesive, while dilation expands the mask to capture irregular defect shapes fully.
* **Otsu Thresholding:** An unsupervised method that selects an optimal threshold by maximizing the variance between foreground and background pixels, ideal for datasets without ground-truth masks.

**4.3 Implementation and Results**

The pipeline was implemented using OpenCV functions within a tf.py\_function for integration into the TensorFlow data pipeline. We retrained the same CNN architecture on the segmented images for 10 epochs, applying class weights. The results were:

| **Metric** | **Score** |
| --- | --- |
| Segmentation Test accuracy | 69.00% |
| Precision | 69.16% |
| Recall | 68.83% |
| F1-Score | 68.73% |

* **Observation:** All metrics dropped by approximately 3 percentage points compared to the baseline. While segmentation focuses the model on defect regions, it occasionally discards subtle texture cues critical for classification, leading to a net performance decrease.

**5. Comparative Analysis**

The performance of both models is summarized below:

| **Metric** | **Baseline CNN** | **Segmentation-Based** |
| --- | --- | --- |
| Accuracy | 72.00% | 69.00% |
| Precision | 76.27% | 69.16% |
| Recall | 72.50% | 68.83% |
| F1-Score | 73.38% | 68.73% |

* **Inference Time:** Segmentation preprocessing introduced computational overhead without improving performance.
* **Class Imbalance:** Both models struggled with minority classes, though the baseline outperformed the segmented approach slightly.

**6. Visualization of Segmentation and Classification**

Qualitative analysis of segmented images revealed:

* **Clear Scratches:** Masks formed continuous white bands over scratches, simplifying classification.
* **Pitted/Patch Defects:** Masks isolated blob-like regions, aiding detection but sometimes including background noise.
* **Crazing:** Fine crack patterns were partially captured, with gaps occasionally hindering accurate classification.

Correctly classified samples typically featured clean, focused defect regions, while misclassifications often stemmed from incomplete or overly inclusive masks.

**7. Topic-Wise Workflow**

* **Data Handling & Preparation:** Loaded annotations and images into tf.data.Dataset, analyzed class imbalance, and applied class weights.
* **Baseline CNN Development:** Designed a 2-block CNN with dense layers, tuning hyperparameters like learning rate and dropout.
* **Evaluation Metrics & Analysis:** Computed accuracy, precision, recall, and F1-score using sklearn.metrics, and visualized confusion matrices.
* **Segmentation Pipeline Design:** Developed an OpenCV-based mask generator (Canny, Otsu, morphology) and integrated it into TensorFlow.
* **Segmented Model Training & Evaluation:** Retrained the CNN on segmented images, noting a performance drop.
* **Reporting & Documentation:** Compiled results and insights into this report.

**8. Day-Wise Effort and Challenges**

The project unfolded over approximately one week:

* **Day 1 – Data Exploration & Planning:** Loaded data into tf.data.Dataset, analyzed class distribution, and planned the approach. Challenges included ensuring correct file paths in Colab.
* **Day 2 – Implement Baseline CNN:** Built the CNN and addressed class imbalance with weights. Tuning learning rate and dropout stabilized training.
* **Day 3 – Evaluation Metrics:** Computed metrics and plotted confusion matrices. Ensuring correct label conversion for weighted metrics was challenging.
* **Day 4 – Segmentation Pipeline:** Developed the segmentation functions and debugged tensor shapes for TensorFlow integration.
* **Day 5 – Segmented Model Training:** Retrained the CNN on segmented images, noting slower training due to preprocessing overhead.
* **Day 6 – Report and Wrap-Up:** Compiled results and documented findings, focusing on clarity and reflection.

Key challenges included maintaining label-image alignment in the data pipeline and debugging segmentation integration. Class imbalance required careful handling to avoid over-predicting the majority class.

**9. Conclusions and Future Work**

* **Baseline CNN:** Performed well on the majority class (Scratch) but struggled with minority classes, mitigated somewhat by class weights.
* **Segmentation Preprocessing:** While intuitive, it did not improve overall performance, likely due to loss of subtle features.
* **Recommendations:**
  + Refine masks with adaptive thresholding or varied morphological operations.
  + Implement contour-based cropping to focus on defect regions.
  + Explore deep segmentation models (e.g., U-Net) if annotated masks are available.
  + Apply data augmentation (e.g., rotations, brightness shifts) to balance the dataset.

This project highlights the potential and limitations of combining traditional image processing with deep learning, suggesting areas for refinement in future iterations.