Module 2 Assignment: Customize a Pre-trained Model for CV Classification

Jigar Thummar (002481709)

Master in Analytics, Northeastern University College of Professional Studies
EAI 6010: Applications of Artificial Intelligence
Richard J. Munthali
March 8, 2025

Introduction

This report discusses the application of transfer learning in computer vision for detecting defects in mono-crystalline and poly-crystalline solar cells. A pre-trained deep learning model in PyTorch was customized and fine-tuned using the Electroluminescence Photovoltaic (ELPV) dataset, which contains 2,624 electroluminescence images with varying defect probabilities. The study investigates the occurrence of overtraining and evaluates techniques such as regularization and early stopping to prevent it. Additionally, the model's classification performance is analyzed, highlighting the best and worst-performing defect categories. Potential improvements are suggested to enhance detection accuracy for the least accurately classified defects.

Know the Data

The ELPV dataset consists of 2,624 grayscale electroluminescence images with dimensions of 300×300 pixels, representing both **mono-crystalline (1,074 images)** and **poly-crystalline (1,550 images)** solar cell types. Each image is labeled with a damage probability score ranging from 0.0 (undamaged) to 1.0 (completely damaged), with intermediate values of 0.33 and 0.67 representing varying degrees of damage. The dataset shows an uneven distribution across these probability categories, with the majority being **undamaged cells (1,508)** and **completely damaged cells (715)**, while the intermediate damage categories have fewer samples (**295 slightly damaged and 106 moderately damaged** cells). Due to the limited number of samples in these intermediate damage categories, particularly for the 0.67 probability class, the classification task was simplified to a binary problem: normal (probability = 0.0) versus damaged (probability > 0.0). This binary approach provides **1,508 normal samples** and **1,116 damaged samples**, creating a more balanced classification problem while still addressing the practical need to detect any level of solar cell damage.

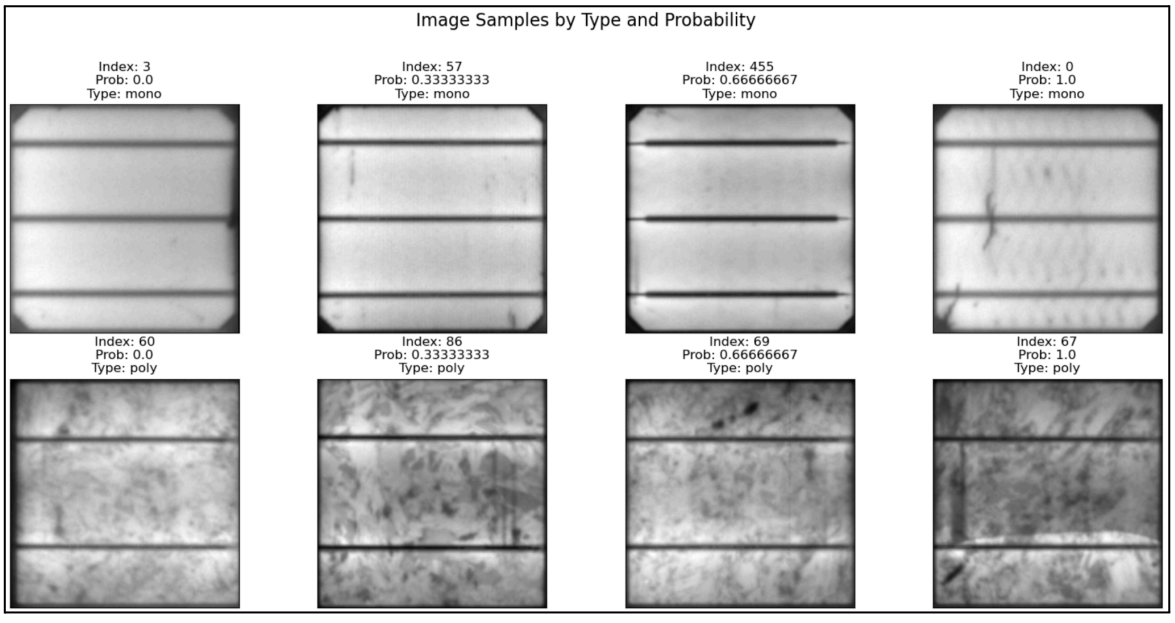

Figure 1: Image Samples by Type and Probability

Preprocessing

Dataset Organization

A custom PyTorch Dataset class SolarPVDataset was implemented to handle the ELPV dataset efficiently. This class encapsulated all preprocessing steps and provided standardized access to the image data. The dataset organization included several key components:

1. **Data Conversion:** The original electroluminescence images from the ELPV dataset were stored as grayscale numpy arrays. These were converted to PIL Image format to enable compatibility with PyTorch's transformation pipeline.

2. **Contrast Limited Adaptive Histogram Equalization (CLAHE):** To enhance feature visibility, CLAHE preprocessing was applied to the images. This technique was particularly valuable for solar panel defect detection as it improved the contrast in local regions while preventing over-amplification of noise. The CLAHE implementation used a clip limit of 2.0 and a tile grid size of 8×8, balancing enhancement with noise control.

3. **Image Transformations:** A series of transformations were applied to prepare the images for the EfficientNet-B1 model (I have also tried larger models like EfficientNet-B7, VGG, and ResNet, each of which supports different image sizes and configurations. However, due to our small dataset, the larger models tend to overfit easily. Therefore, I chose EfficientNet-B1 as a balanced option.):
    - Resizing to 255×255 pixels using bilinear interpolation
    - Center cropping to 240×240 pixels to match the EfficientNet-B1 input size
    - Converting grayscale images to 3-channel format (required for the pretrained model)
    - Normalizing pixel values using the ImageNet standard mean and standard deviation

4. **Labeling Strategy:** While the original dataset provided granular defect probabilities (0.0, 0.33, 0.67, and 1.0), a binary classification approach was adopted for this study. Labels were transformed into a binary format where 0 represented normal panels (original probability 0.0) and 1 represented panels with any level of defect (original probabilities > 0).

Data Splitting and Loading

The ELPV dataset was strategically divided using a stratified sampling approach to maintain consistent class distribution across subsets. A 70-15-15 split allocated 1,836 images to training, with 394 images each for validation and testing. This was accomplished through a two-stage process, first creating a training-temporary split (70-30), then dividing the temporary set equally between validation and test sets. Data loading was optimized using

PyTorch's DataLoader with a batch size of 32, shuffling enabled for training data only, and four parallel worker processes. These configurations balanced computational efficiency with model stability while ensuring the robust evaluation of the EfficientNet-B1 model's performance on solar panel defect detection.

Model Training

Model Architecture

The EfficientNet-B1 architecture was selected as the foundation for this solar panel defect detection system. EfficientNet is a lightweight yet high-performance model that outperforms most existing models in terms of top-1% accuracy. Originally designed for multi-class image classification, the model was customized for binary classification by modifying the final fully connected layer to output a single value. This approach leveraged transfer learning to benefit from the feature extraction capabilities of EfficientNet-B1 pre-trained on ImageNet, while adapting it to the specific task of solar panel defect detection.

Loss Function and Optimization

For this binary classification task, Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss) was chosen as the loss function. This combines a sigmoid activation with binary cross-entropy loss in a single, numerically stable operation. The Adam optimizer was employed with a learning rate of 1e-4 and weight decay of 1e-4 to provide regularization.

A learning rate scheduler (ReduceLROnPlateau) was implemented to automatically adjust the learning rate based on validation loss. The scheduler reduced the learning rate by a factor of 0.1 after two epochs without improvement, enabling fine-grained optimization as the model approached convergence.

Training Protocol

The model was trained for 20 epochs with comprehensive monitoring of both training and validation metrics. The training loop incorporated several key components:

1. **Batch Processing**: Each batch of images was processed and forward-passed through the model to obtain predictions.
2. **Loss Calculation**: Binary cross-entropy loss was computed between predictions and ground truth labels.
3. **Backpropagation**: Gradients were calculated and used to update model weights via the Adam optimizer.
4. **Metric Tracking**: Training and validation loss, accuracy, and learning rate were recorded for each epoch.
5. **Learning Rate Adaptation**: The scheduler adjusted the learning rate based on validation performance.

6. **Model Checkpointing**: Model weights were saved after each epoch to enable selection of the optimal model.

Initially, the model was trained using 3 fully connected layers, batch normalization, and a dropout rate of 0.3. The architecture consisted of EfficientNet-B1 pretrained output features followed by layers of 512 and 256 neurons. However, its performance did not meet expectations. Figure 2 shows that the accuracy is plateaued at around 80%.

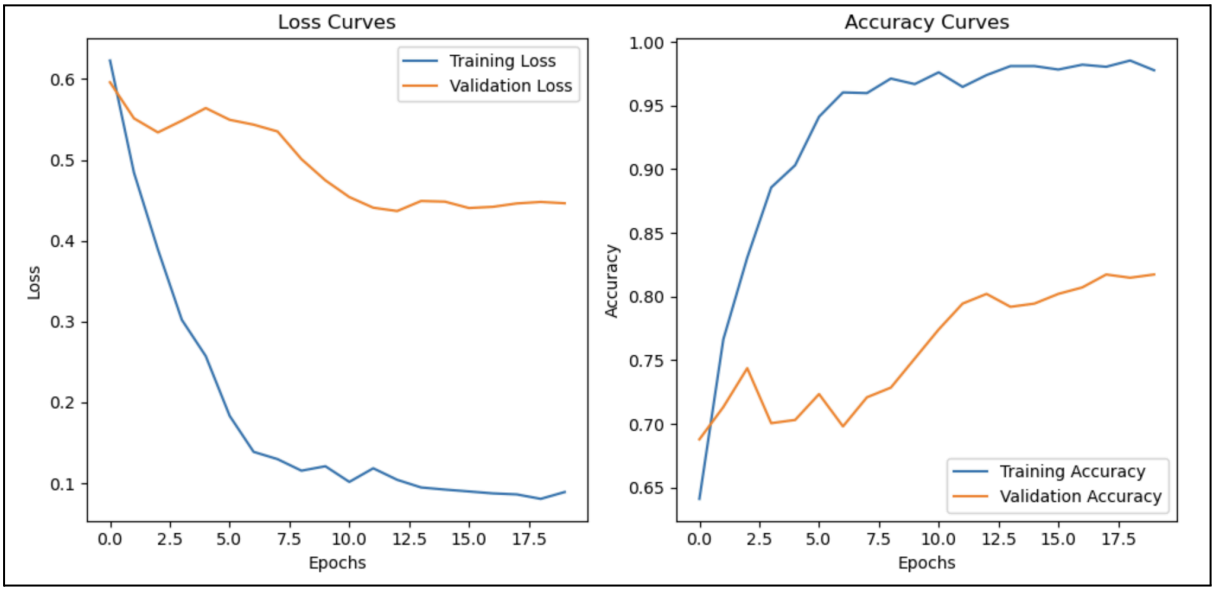

Figure 2: Loss and Accuracy curves of model with three fc layers and dropout

To improve performance, I modified the architecture by using a single fully connected layer with one output neuron.

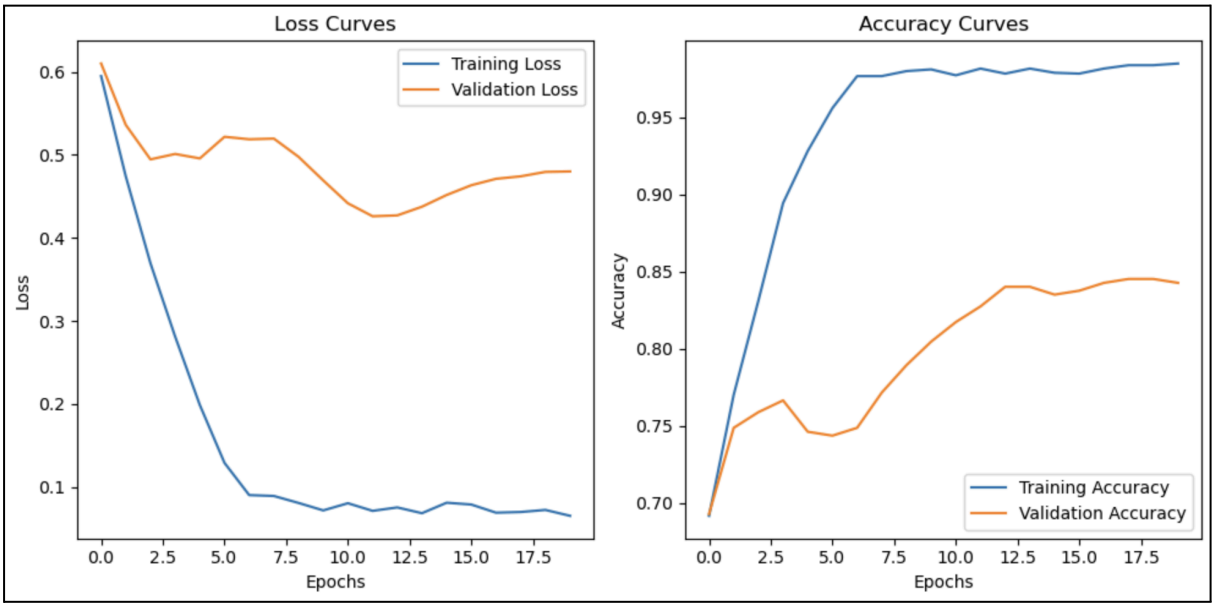

Figure 3: Loss and Accuracy curves of model with one fc layer without dropout

The training curves in Figure 3 demonstrate clear evidence of overtraining beginning after epoch 12. As the epochs progress, training loss continues to decrease steadily while validation loss reaches its minimum around epoch 12 before gradually increasing. Similarly, training accuracy rapidly climbs to approximately 98% by epoch 6, while validation accuracy improves more slowly, reaching about 84-85% by the end of training. The widening gap between training and validation metrics (approximately 14 percentage points in accuracy) indicates classic model overfitting, where the model becomes increasingly specialized to the training data at the expense of generalization capability. This pattern establishes epoch 12 as the optimal stopping point, after which continued training produces diminishing returns despite the implemented regularization techniques.

Effective Techniques for Preventing Overfitting

Weight Decay: The Adam optimizer was configured with a weight decay parameter of 1e-4, which applied L2 regularization to the model weights. This effectively penalized large weight values, encouraging the model to learn more generalizable features rather than memorizing the training data.

Learning Rate Scheduling: The ReduceLROnPlateau scheduler monitored validation loss and reduced the learning rate when improvements stalled. This approach was particularly effective during the later stages of training, helping the model find more optimal minima in the loss landscape without overfitting.

Model Checkpointing: Saving model weights after each epoch allowed for the selection of the most generalized model based on validation performance rather than training metrics. This technique enabled the identification of the optimal stopping point at epoch 19.

Appropriate Architecture Selection: After experimenting with complex architectures, a simpler model with fewer parameters (single fully connected layer) provided better generalization. This demonstrates that for datasets of limited size, simpler models may outperform more complex ones due to reduced overfitting risk.

Data Augmentation through CLAHE: While not a traditional data augmentation technique, the application of CLAHE preprocessing enhanced feature visibility and effectively expanded the model's understanding of the dataset's feature space without introducing artificial transformations that might not represent real-world scenarios.

Model Performance Analysis

The final model achieved strong performance with an overall accuracy of 86.0% on the test set. When evaluated using a decision threshold of 0.4, the model demonstrated balanced capabilities across both classes: normal cells (Class 0) with precision=0.88, recall=0.88, F1-score=0.88; and damaged cells (Class 1) with precision=0.84, recall=0.83, F1-score=0.83.

The confusion matrix revealed 200 correctly classified normal cells and 139 correctly identified damaged cells, with relatively few misclassifications (27 false positives and 28 false negatives). The model performed best on completely damaged cells (probability=1.0) and normal mono-crystalline cells due to their distinctive visual patterns and uniform appearance, respectively.

```
Classification Report on Test Set:
              precision    recall  f1-score   support

     Class 0       0.88      0.88      0.88       227
     Class 1       0.84      0.83      0.83       167

    accuracy                           0.86       394
   macro avg       0.86      0.86      0.86       394
weighted avg       0.86      0.86      0.86       394

[[200  27]
 [ 28 139]]
```

Figure 4: Classification report and confusion matrix

Ways to Improve Model Performance

Targeted Data Augmentation: Generate synthetic examples specifically for underrepresented damage classes (0.33 and 0.67 probability). This would address the class imbalance that contributes to poor performance on these categories.

Multi-Stage Classification: Implement a hierarchical approach where one model detects damage presence and a second specialized model determines severity. This separation of concerns would allow specialized training for the difficult intermediate damage cases.

Collect Additional Training Data: Acquire more real-world examples of slightly damaged (0.33) and moderately damaged (0.67) solar cells. Expanding the dataset with more of these underrepresented categories would directly address the class imbalance issue.

Conclution

In conclusion, this study successfully applied transfer learning to detect defects in solar cells using a customized EfficientNet-B1 model. The simplified binary classification approach achieved an overall accuracy of 86.0% on the test set, with balanced performance across normal and damaged cell categories. While the model excelled at identifying completely damaged cells and normal cells, it struggled with intermediate damage categories due to data limitations. Effective techniques for preventing overfitting included weight decay, learning rate scheduling, model checkpointing, appropriate architecture selection, and CLAHE preprocessing. Future improvements should focus on targeted data augmentation, implementing a multi-stage classification approach, and expanding the dataset with additional examples of underrepresented damage categories to enhance the model's ability to detect subtle defects in solar cells.

References

Buerhop-Lutz, C., Deitsch, S., Maier, A., Gallwitz, F., Berger, S., Doll, B., Hauch, J., Camus, C., & Brabec, C. (2018, September 24–28). A Benchmark for Visual Identification of Defective Solar Cells in Electroluminescence Imagery [Conference paper]. 1287–1289. https://doi.org/10.4229/35thEUPVSEC20182018-5CV.3.15

Huilgol, P. (2025, March 8). *Top 4 pre-trained models for image classification with Python code*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

Marimuthu, P. (2024, October 12). Image contrast enhancement using CLAHE. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/08/image-contrast-enhancement-using-clahe/

PyTorch. (n.d.). *torchvision.models.efficientnet_b1*. PyTorch.https://pytorch.org/vision/main/models/generated/torchvision.models.efficientnet_b1.html#torchvision.models.efficientnet_b1

Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning. https://doi.org/10.48550/arXiv.1905.11946

PyTorch. (n.d.). *torch.optim*. PyTorch. https://pytorch.org/docs/stable/optim.html