**Binary Image Classification using ResNet-152 with Class Weights and Early Stopping**

**One Sentence Summary**

This repository contains a project to classify images into two classes ("yes" or "no") using a fine-tuned ResNet-152 model with class weighting to address imbalance and early stopping to prevent overfitting.

**Overview**

**Task Definition:**

The objective is to classify images into one of two categories using a deep convolutional neural network (CNN). The dataset is imbalanced, with more "yes" than "no" labels.

**Approach:**

The problem was formulated as a supervised image classification task. I used a pretrained ResNet-152 model and replaced the final layer to adapt it to binary classification. To address class imbalance, I applied class weights in the loss function. Additionally, early stopping and learning rate scheduling were implemented to optimize training.

**Performance Summary:**

The final model achieved **88.24% accuracy** on the validation set, with a **macro F1-score of 0.8712**. The model demonstrates good balance between precision and recall across both classes, outperforming the initial unweighted baseline.

**Summary of Work Done**

**Data**

**Type:**

Image dataset for binary classification.

**Size:**

* **Training:** 202 images (124 "yes", 78 "no")
* **Validation (Test):** 51 images (31 "yes", 20 "no")

**Preprocessing / Augmentation:**

* Images were resized and normalized according to ResNet preprocessing standards.
* (Optional: Mention specific transforms like RandomHorizontalFlip, etc. if used.)

**Data Visualization:**

* Label distribution was plotted to highlight imbalance.
* Sample image visualizations were reviewed for visual confirmation of classes.

**Problem Formulation**

* **Input:** RGB images
* **Output:** Binary label (0: "no", 1: "yes")

**Model Used:**

* Pretrained ResNet-152 (ImageNet weights)
* Final FC layer modified: nn.Linear(2048 → 2)

**Loss Function:**

* CrossEntropyLoss with class weights: [1.2949 (no), 0.8145 (yes)]

**Optimizer:**

* Adam optimizer (lr=1e-4, weight\_decay=1e-3)

**Training**

**Software & Hardware:**

* **Python 3.x**, **PyTorch**, **Torchvision**
* Training performed on GPU (if applicable, update with GPU type)

**Training Procedure:**

* Weighted loss to counter class imbalance
* StepLR learning rate scheduler (step\_size=7, gamma=0.1)
* Early stopping with patience of 5 epochs (monitored validation accuracy)
* Model checkpointing on best validation accuracy

**Training Time:**

* Training completed in 10 epochs due to early stopping
* Each epoch took ~[fill in time] on [your hardware]

**Performance Summary**

**Key Metrics (on Validation Set):**

| **Metric** | **Value** |
| --- | --- |
| Accuracy | 88.24% |
| Precision | 0.8973 (macro) |
| Recall | 0.8589 (macro) |
| F1-score | 0.8712 (macro) |

**Detailed Classification Report:**

| **Class** | **Precision** | **Recall** | **F1-score** | **Support** |
| --- | --- | --- | --- | --- |
| no | 0.9375 | 0.7500 | 0.8333 | 20 |
| yes | 0.8571 | 0.9677 | 0.9091 | 31 |

* **Weighted average F1-score:** 0.8794
* **Confusion matrix and ROC curves** were generated to visualize model performance (if applicable).

**Performance Comparison**

**Before using class weights:**

* Accuracy: 60.78%
* Recall on “no”: 0.00%
* Severe bias toward majority class (“yes”)

**After applying class weights:**

* Accuracy: 88.24%
* Balanced precision/recall across both classes
* Much better generalization and fairness

**Conclusions**

* Class weighting significantly improved the model’s ability to learn minority class ("no").
* ResNet-152, when fine-tuned properly, is highly effective for binary image classification tasks.
* Early stopping helped avoid overfitting after ~10 epochs.
* The final model achieves strong, balanced performance suitable for deployment or further experimentation.

**Future Work**

* Add **data augmentation** to improve generalization
* Test **Focal Loss** as an alternative to further handle imbalance
* Try other architectures (EfficientNet, ViT) for potential accuracy gains
* Deploy the model with a web or mobile interface
* Conduct experiments on a larger, more diverse dataset

**How to Reproduce Results**

1. Clone the repository
2. Install dependencies using pip install -r requirements.txt
3. Place images in the expected directory structure (train/yes, train/no, etc.)
4. Run the training script: train\_model.ipynb
5. Evaluate using evaluate\_model.ipynb

**Repository Structure**

| **File / Folder** | **Description** |
| --- | --- |
| train\_model.ipynb | Fine-tunes ResNet-152 and applies class weighting |
| evaluate\_model.ipynb | Performs evaluation and prints classification report |
| utils.py | Contains helper functions (e.g., class weight computation) |
| requirements.txt | Python dependencies (PyTorch, torchvision, sklearn, etc.) |
| README.md | Project overview and instructions |

**Software Setup**

* Python 3.8+
* PyTorch
* Torchvision
* scikit-learn
* numpy, matplotlib
* Install with: pip install -r requirements.txt