Comparative Research on Tiny Deep Learning Models for Radiography

Introduction

This project focuses on the comparative analysis and prototypical learning of three lightweight deep learning models MobileNetV2, SqueezeNet, and EfficientNet-Lite on a multi-class radiography dataset. The core objective is to evaluate the performance and interpretability of these models for medical image classification, specifically targeting COVID-19, Normal, Lung Opacity, and Viral Pneumonia classes using chest X-ray images.

The workflow encompasses data preprocessing, meta-learning episode creation, model architecture definition, training with prototypical networks, performance evaluation, and interpretability through explainable AI techniques. This documentation details each stage, providing a comprehensive and professional overview of the implemented methodology.

1. Data Preparation and Preprocessing

1.1 Dataset Overview

- **Source**: COVID-19 Radiography Database
- **Classes**: COVID-19, Normal, Lung Opacity, Viral Pneumonia
- **Total Images**: 21,165

1.2 Preprocessing Steps

- **Image Loading**: Images are loaded in grayscale for initial preprocessing.
- **Resizing**: All images are resized to 224x224 pixels to standardize input dimensions across models.
- **Normalization**: Pixel values are normalized to a 1 range.
- **Histogram Equalization**: Applied to enhance contrast.
- **Noise Reduction**: Gaussian and median filtering are used to reduce image noise.
- **Final Transformation**: Images are converted to 3-channel RGB format and normalized for use with pre-trained models.

1.3 Data Splitting

- **Stratified Split**: Ensures balanced class distribution.
    - **Training Set**: 96%
    - **Validation Set**: 2%
    - **Test Set**: 2%

2. Meta-Learning Setup

2.1 Few-Shot Episode Creation

- **Meta-Tasks**: Constructed using an N-way, K-shot approach.
    - **N (Classes per Episode)**: 3
    - **K (Support Samples per Class)**: 5
    - **Q (Query Samples per Class)**: 5
    - **Number of Episodes**: 20 (for each split: train, validation, test)
- **Purpose**: Simulates few-shot learning scenarios, enabling the models to generalize from limited samples.

2.2 MetaDataset Class

- **Function**: Wraps the episodes into a PyTorch Dataset, facilitating batch-wise loading for meta-training.

3. Model Architectures

3.1 MobileNetV2 Encoder

- **Backbone**: Pre-trained MobileNetV2
- **Output**: 128-dimensional embedding
- **Adaptation**: Final layer replaced to produce compact feature vectors.

3.2 SqueezeNet Encoder

- **Backbone**: Pre-trained SqueezeNet 1.1
- **Output**: 128-dimensional embedding
- **Adaptation**: Final convolutional output mapped to embedding space.

3.3 EfficientNet-Lite Encoder

- **Backbone**: EfficientNet-Lite0 (via TIMM library)
- **Output**: 128-dimensional embedding
- **Adaptation**: Classifier removed; output mapped to embedding vector.

4. Prototypical Networks and Training

4.1 Prototypical Learning

- **Concept**: Each class is represented by a prototype (mean embedding of support set).
- **Classification**: Query samples are classified based on Euclidean distance to prototypes.

4.2 Training Loop

- **Optimizer**: Adam (learning rate = 0.001)
- **Scheduler**: StepLR (decays LR by 0.5 every 10 epochs)
- **Epochs**: 100
- **Loss Function**: Cross-entropy on negative distances
- **Metrics Logged**: Loss and accuracy per epoch for each model

4.3 Training Results (Sample)

Epoch	MobileNetV2 Loss	MobileNetV2 Accuracy	SqueezeNet Loss	SqueezeNet Accuracy	EfficientNet-Lite Loss	EfficientNet-Lite Accuracy
1	1.81	72.33%	1.44	55.33%	0.57	74.33%
10	0.25	92.67%	0.60	69.33%	0.04	99.33%
20	0.00	100.00%	0.27	87.33%	0.00	100.00%
100	0.00	100.00%	0.00	100.00%	0.00	100.00%

5. Model Evaluation

5.1 Evaluation Metrics

- **Accuracy**: Overall correct predictions
- **Precision, Recall, F1-score**: Computed per class
- **AUC-ROC**: One-vs-Rest multiclass

5.2 Test Set Results

Model	Accuracy	Precision (per class)	Recall (per class)	F1-score (per class)	AUC-ROC
MobileNetV2 Encoder	85.00%	[0.79, 0.88, 0.88]	[0.84, 0.88, 0.83]	[0.82, 0.88, 0.86]	0.96
SqueezeNet Encoder	68.33%	[0.63, 0.71, 0.71]	[0.61, 0.72, 0.72]	[0.62, 0.71, 0.72]	0.88
EfficientNet-Lite	83.00%	[0.78, 0.85, 0.87]	[0.84, 0.82, 0.83]	[0.81, 0.83, 0.85]	0.96

- **Interpretation**: MobileNetV2 and EfficientNet-Lite both achieved high accuracy and AUC, with EfficientNet-Lite showing slightly faster convergence. SqueezeNet, while lightweight, lagged in performance but remains valuable for resource-constrained environments.

6. Model Export and Deployment

- **Model Saving**: All trained models were saved in both PyTorch (`.pth`) and ONNX (`.onnx`) formats for interoperability and deployment.
- **ONNX Export**: Ensured compatibility for integration into production pipelines and inference on various platforms.

7. Model Interpretability

7.1 Explainable AI Techniques

- **LIME (Local Interpretable Model-Agnostic Explanations)**: Used to generate heatmaps highlighting influential regions in X-ray images for each model.
- **Workflow**:
    - Select a test image.
    - Generate perturbed samples and obtain model predictions.
    - Visualize the most influential superpixels for each class.

7.2 Insights

- **Interpretability**: LIME visualizations provided transparency into the decision process, helping to validate that models focus on clinically relevant regions of the radiographs.
- **Clinical Relevance**: Such interpretability is crucial for trust and adoption in medical AI applications.

8. Conclusion

This project demonstrates a robust workflow for benchmarking and interpreting lightweight deep learning models in medical imaging. Through careful preprocessing, meta-learning, and rigorous evaluation, the study provides actionable insights into the trade-offs between model size, accuracy, and interpretability. The use of explainable AI further enhances the transparency and trustworthiness of the models, making them suitable candidates for deployment in real-world clinical settings.

Appendix: Key Implementation Highlights

- **Reproducibility**: All random seeds and splits are controlled for consistent results.
- **Code Modularity**: Each component (data, model, training, evaluation, interpretability) is encapsulated for clarity and reusability.
- **Scalability**: The workflow can be adapted to other datasets and model architectures with minimal changes.