OCTDL Eye Disease Classification: Deep Learning Model Comparison (ResNet50 vs InceptionV3)

Project Documentation

Team Members: Ahmad F. Obaid, Ibrahim Khdeir

Supervisor: Dr. Adnan Salman

Date: June 2025

Abstract

This document details the development and evaluation of a deep learning pipeline for classifying retinal diseases from Optical Coherence Tomography (OCT) images. Leveraging transfer learning with pre-trained convolutional neural networks (ResNet50 and InceptionV3), the project aims to build a robust classifier capable of distinguishing among multiple retinal disease categories, including Age-related Macular Degeneration, Diabetic Macular Edema, Epiretinal Membrane, Retinal Artery Occlusion, Retinal Vein Occlusion, Vitreomacular Interface Disease, and Normal cases. The methodology encompasses comprehensive data preparation, model architecture design, a two-phase training approach (frozen backbone and fine-tuning), and rigorous evaluation using various metrics such as accuracy, precision, recall, F1-score, and confusion matrices. The project emphasizes reproducibility, aiming for high classification accuracy and clinical relevance to potentially assist in early diagnosis and ophthalmologist review.

1. Project Overview

1.1 Objective

The primary objective of this project is to develop a highly accurate and robust deep learning model for the classification of various retinal diseases from Optical Coherence Tomography (OCT) images. The model is designed to differentiate between six specific disease categories—Age-related Macular Degeneration (AMD), Diabetic Macular Edema (DME), Epiretinal Membrane (ERM), Retinal Artery Occlusion (RAO), Retinal Vein Occlusion (RVO), Vitreomacular Interface Disease (VID)—and normal retinal conditions. This classification capability is crucial for aiding ophthalmologists in early diagnosis and treatment planning, thereby improving patient outcomes.

1.2 Methodology

The chosen methodology for achieving this objective is transfer learning, a powerful technique in deep learning that leverages knowledge gained from solving one problem and applies it to a different but related problem. Specifically, pre-trained Convolutional Neural Networks (CNNs), namely ResNet50 and InceptionV3, are utilized as feature extractors. These models, pre-trained on the vast ImageNet dataset, possess a strong ability to recognize complex visual patterns. By using them as a starting point, the project aims to achieve high accuracy even with a relatively limited dataset of OCT images. The process involves two main phases: initially freezing the pre-trained layers to train a custom classification head, and then fine-tuning a subset of the pre-trained layers to adapt the model more specifically to the nuances of OCT images.

1.3 Scope

The scope of this project includes the entire pipeline from data acquisition and preprocessing to model training, evaluation, and reporting. It covers the programmatic handling of the OCTDL dataset, implementation of data augmentation strategies, construction of custom classification heads on top of pre-trained models, a two-stage training regimen, and a comprehensive evaluation framework. The evaluation focuses on quantitative metrics such as overall accuracy, per-class precision, recall, and F1-score, as well as qualitative assessments through confusion matrices and error analysis. While deployment is considered optional, the project aims to produce a well-documented and reproducible codebase that can serve as a foundation for future clinical integration.

2. Data and Resources

2.1 Dataset: OCTDL

The Optical Coherence Tomography Deep Learning (OCTDL) dataset is the cornerstone of this project, providing the necessary imaging data for training and evaluating the deep learning models. This publicly available dataset is accessible via Nature Scientific Data [1].

Source: Nature Scientific Data article: https://www.nature.com/articles/s41597-024-03182-7

Volume: The dataset comprises over 2,000 labeled OCT images. Each image is meticulously tagged with one of seven categories: six distinct retinal disease categories (Age-related Macular Degeneration (AMD), Diabetic Macular Edema (DME), Epiretinal Membrane (ERM), Retinal Artery Occlusion (RAO), Retinal Vein Occlusion (RVO), Vitreomacular Interface Disease (VID)) and a 'Normal' category.

Access Instructions: The dataset can be programmatically downloaded or requested through the DOI link provided in the Nature Scientific Data article. The `Deelp_main_clean.ipynb` notebook indicates an expected local directory structure after extraction:

```
data/
  train/
    AMD/
    DME/
    ERM/
    RAO/
    RVO/
    VID/
    Normal/
  val/
    AMD/
    DME/
    ERM/
    RAO/
    RVO/
    VID/
    Normal/
  test/
    AMD/
    DME/
    ERM/
    RAO/
    RVO/
    VID/
    Normal/
```

This structured organization facilitates efficient data loading and management for machine learning tasks.

2.2 Pre-trained Models

Transfer learning relies on the use of pre-trained Convolutional Neural Networks (CNNs) that have learned rich feature representations from large-scale image datasets. For this project, two prominent architectures pre-trained on ImageNet are considered:

- **ResNet50:** A deep residual network known for its ability to mitigate the vanishing gradient problem through skip connections, enabling the training of very deep networks.
- **InceptionV3:** An architecture that employs inception modules to capture multi-scale features by performing convolutions with different filter sizes in parallel, leading to efficient use of computational resources.

Frameworks: Both TensorFlow/Keras and PyTorch frameworks provide convenient APIs to load these pre-trained models. The `Deelp_main_clean.ipynb` notebook specifically uses TensorFlow/Keras for its implementation.

Download Instructions: The models are loaded without their original classification heads (`include_top=False`) to allow for the addition of a custom classification layer tailored to the OCT image classification task. The `weights="imagenet"` argument ensures that the models are initialized with weights learned from the ImageNet dataset.

```
 # TensorFlow/Keras ResNet50
from tensorflow.keras.applications import ResNet50
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=
(224,224,3))

# TensorFlow/Keras InceptionV3
from tensorflow.keras.applications import InceptionV3
base_model = InceptionV3(weights="imagenet", include_top=False, input_shape=
(299,299,3))
```

2.3 Data Preparation

Effective data preparation is critical for the success of any deep learning project. This involves several key steps to transform raw OCT images into a format suitable for model training.

2.3.1 Download & Unpack

The initial step involves programmatically fetching the OCTDL archive and ensuring its correct extraction and organization. The `create_data_splits` function in `Deelp_main_clean.ipynb` handles the removal of existing splits and the creation of new directories for training, validation, and testing data. It also verifies the directory structure and image counts per class, ensuring data integrity before proceeding.

2.3.2 Preprocessing Pipeline

To standardize the input and enhance model performance, a robust preprocessing pipeline is implemented:

- **Image Resizing:** All OCT images are resized to the specific input dimensions required by the chosen pre-trained model (e.g., 224x224 pixels for ResNet50 and 299x299 pixels for InceptionV3). This ensures uniformity in input size across the dataset.
- **Normalization:** Pixel values are scaled to a consistent range. The implementation uses the `preprocess_input` functions provided by Keras

applications (`resnet_preprocess` and `inception_preprocess`), which handle the appropriate normalization (e.g., scaling to [-1, 1] or [0, 1] based on the model's requirements).

- **Augmentation:** To improve the generalization capabilities of the models and mitigate overfitting, a series of data augmentation techniques are applied. These transformations introduce variability into the training data without increasing the actual number of samples. The `data_augmentation` layer in the `Deelp_main_clean.ipynb` notebook includes:
    - Random horizontal flips
    - Random rotations (0.15 radians, approximately ±8.5 degrees)
    - Random zoom (0.15, scaling between 0.85x and 1.15x)
    - Random translation (0.1, shifting by up to 10% of image dimensions)
    - Random contrast adjustments (0.2)
    - Random brightness adjustments (0.15)
    - Gaussian noise (0.01)

These augmentations are crucial for exposing the model to a wider range of image variations, making it more robust to real-world data.

2.3.3 Train/Validation/Test Split

The dataset is systematically divided into training, validation, and test sets to ensure unbiased model evaluation. The `create_data_splits` function implements an 80/10/10 split for training, validation, and testing, respectively. This split ensures that each disease class is proportionally represented across all subsets, maintaining the integrity of the class distribution. The splitting process involves shuffling images within each class and then distributing them into the respective directories. The counts of images in each split are printed to confirm the successful partitioning of the dataset.

```
 # Example from Deelp_main_clean.ipynb
def create_data_splits():
    # ... (code for removing existing splits and creating directories)
    def split_class_data(src_dir, dest_dirs, ratios=[0.8, 0.1, 0.1]):
        # ... (code for splitting and copying files)
    split_class_data(BASE_DATA_DIR, [TRAIN_DIR, VAL_DIR, TEST_DIR])
    # ... (code for counting images and printing results)
```

This structured approach to data preparation ensures that the models are trained on diverse and representative data, and evaluated on unseen samples, providing a reliable assessment of their performance.

3. Model Architecture & Transfer Learning

This section details the design and implementation of the deep learning models, focusing on how transfer learning is applied to classify retinal diseases from OCT images.

3.1 Load Pre-trained Backbone

The foundation of our models lies in leveraging pre-trained Convolutional Neural Networks (CNNs) as feature extractors. The `create_model` function in `Deelp_main_clean.ipynb` is responsible for instantiating the chosen backbone (ResNet50 or InceptionV3) with weights pre-trained on the ImageNet dataset. Crucially, the `include_top=False` argument is used to remove the original classification head of these models. This allows us to attach a custom classification head tailored to our specific task of classifying retinal diseases.

Initially, all convolutional layers of the base model are set to `trainable = False`. This freezing mechanism ensures that the pre-trained weights, which have learned general image features, are preserved during the initial training phase. This approach is particularly beneficial when working with limited datasets, as it prevents overfitting and allows the new classification head to learn to interpret the extracted features without altering the robust feature extraction capabilities of the pre-trained network.

```
 # Example from Deelp_main_clean.ipynb
def create_model(architecture=\'resnet50\', num_classes=None,
input_shape=None):
    # ... (loading base_model based on architecture)
    base_model.trainable = False # Freeze base model
    # ...
```

3.2 Add Custom Classification Head

Following the frozen pre-trained backbone, a custom classification head is appended to the model. This head is designed to interpret the high-level features extracted by the CNN and map them to the seven distinct disease categories (including Normal). The architecture of the custom head is as follows:

- **Global Average Pooling 2D Layer:** This layer reduces the spatial dimensions of the feature maps, converting them into a single feature vector. It helps in reducing the number of parameters and makes the model more robust to spatial translations of the input image.
- **Dense Layer (256 units, ReLU activation):** A fully connected layer with 256 neurons and a Rectified Linear Unit (ReLU) activation function. This layer introduces non-linearity and allows the model to learn complex relationships between the extracted features and the target classes.
- **Dropout Layer (rate = 0.5):** A dropout layer is included with a dropout rate of 0.5. This regularization technique randomly sets a fraction of input units to zero at each update during training, which helps prevent overfitting by reducing co-adaptation among neurons.
- **Final Dense Output Layer (Softmax activation, `num_classes = 7`):** The final layer is a fully connected layer with `num_classes` (which is 7 in our case) neurons, corresponding to the seven retinal disease categories. A softmax activation function is applied to this layer, which outputs a probability distribution over the classes, indicating the likelihood of an input image belonging to each category.

This custom head is crucial for adapting the general features learned by the pre-trained model to the specific patterns relevant for OCT image classification.

3.3 Compile Model

Before training, the model needs to be compiled with appropriate loss functions, optimizers, and metrics. The `create_model` function configures the compilation as follows:

- **Loss Function:** `Categorical Cross-Entropy` is chosen as the loss function. This is suitable for multi-class classification problems where labels are one-hot encoded, as is the case with the `image_dataset_from_directory` with `label_mode='categorical'`.
- **Optimizer:** The `Adam` optimizer is used with an initial learning rate of `1e-4`. Adam is an adaptive learning rate optimization algorithm that is widely used for

its efficiency and effectiveness in various deep learning tasks.

- **Metrics:** To comprehensively evaluate the model's performance during training and evaluation, the following metrics are monitored:
    - `accuracy`: The proportion of correctly classified samples.
    - `Precision`: A measure of the accuracy of positive predictions (true positives divided by true positives + false positives).
    - `Recall`: A measure of the model's ability to find all positive samples (true positives divided by true positives + false negatives).
    - `F1-Score`: The harmonic mean of precision and recall, providing a single metric that balances both.

These metrics provide a holistic view of the model's performance, especially important for medical imaging where false negatives or false positives can have significant implications.

3.4 Initial Training (Frozen Backbone)

The training process is divided into two phases. The first phase involves training the model with the pre-trained backbone layers frozen. This initial training allows the newly added custom classification head to learn to interpret the features extracted by the fixed backbone. The model is trained for `10-15 epochs` (specifically `INITIAL_EPOCHS = 15` in `Deelp_main_clean.ipynb`).

During this phase, a `batch size` of `32` (`BATCH_SIZE = 32`) is used. To prevent overfitting and optimize training efficiency, several callbacks are employed:

- **Early Stopping:** Monitors the `validation loss` with a `patience` of `5 epochs`. If the validation loss does not improve for five consecutive epochs, training is stopped, and the model weights from the best epoch are restored. This prevents the model from continuing to train on the training data when it is no longer generalizing well to unseen data.
- **Model Checkpoint:** Saves the model weights (`best_model_{model_name.lower()}.keras`) whenever an improvement in `validation accuracy` is observed. This ensures that the best performing model during training is preserved.

- **ReduceLROnPlateau:** Reduces the learning rate by a factor of `0.2` if the `validation loss` plateaus for `3 epochs`, with a minimum learning rate of `1e-6`. This helps the optimizer to converge more effectively when the loss function landscape becomes flatter.
- **TensorBoard:** Logs training metrics and model graphs for visualization and analysis, aiding in understanding the training process and identifying potential issues.

Class weights are also calculated and applied during training to address potential class imbalance in the dataset. This ensures that the model does not become biased towards majority classes and gives appropriate attention to minority classes, which is crucial in medical diagnosis where rare diseases might be clinically significant.

3.5 Fine-Tuning (Unfreeze Some Layers)

After the initial training phase, the model undergoes a fine-tuning phase to further enhance its performance and adapt it more specifically to the OCT image domain. In this phase, a subset of the top convolutional layers of the pre-trained backbone are unfrozen. For ResNet50, the last 10 layers are unfrozen, while for InceptionV3, the last 20 layers are unfrozen. This allows these layers to be updated during training, enabling the model to learn more domain-specific features from the OCT images.

To facilitate stable fine-tuning, the learning rate is reduced by a factor of 10 (from `1e-4` to `1e-5`). The model is then recompiled with this lower learning rate. Training continues for an additional `10-15 epochs` (`FINE_TUNE_EPOCHS = 15`), building upon the knowledge gained in the initial phase. The same set of callbacks (Early Stopping, Model Checkpoint, ReduceLROnPlateau, TensorBoard) continue to monitor and guide the training process, ensuring optimal performance and preventing overfitting during fine-tuning.

This two-stage training strategy—initial training with a frozen backbone followed by fine-tuning—is a common and effective approach in transfer learning, allowing for both rapid initial convergence and subsequent specialized adaptation to the target dataset.

4. Evaluation & Reporting

Comprehensive evaluation and clear reporting are essential to assess the performance of the developed models and communicate the findings effectively. This section outlines the metrics used, visualization techniques, and error analysis procedures.

4.1 Final Evaluation on Test Set

Upon completion of the training and fine-tuning phases, the models are rigorously evaluated on the unseen test set to provide an unbiased assessment of their generalization capabilities. The evaluation focuses on a suite of metrics that offer a holistic view of performance across all classes:

- **Overall Accuracy:** The proportion of correctly classified instances out of the total number of instances. This provides a general measure of the model's correctness.
- **Per-Class Precision:** For each disease category, precision measures the proportion of true positive predictions among all positive predictions (True Positives / (True Positives + False Positives)). It indicates the model's ability to avoid false alarms.
- **Per-Class Recall:** For each disease category, recall measures the proportion of true positive predictions among all actual positive instances (True Positives / (True Positives + False Negatives)). It indicates the model's ability to find all relevant instances.
- **F1-Score (Macro):** The F1-score is the harmonic mean of precision and recall. The macro-averaged F1-score calculates the F1-score for each class independently and then takes the average, treating all classes equally. This is particularly useful in multi-class classification with imbalanced datasets.
- **Confusion Matrix:** A confusion matrix is a table that visualizes the performance of a classification algorithm. Each row of the matrix represents the instances in an actual class, while each column represents the instances in a predicted class. It allows for detailed analysis of misclassifications, showing where the model is performing well and where it is struggling. The confusion matrix is plotted and saved as a heatmap for clear visual interpretation.
- **ROC Curves:** Receiver Operating Characteristic (ROC) curves are generated for each class using a one-vs-rest approach. An ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The

Area Under the Curve (AUC) provides an aggregate measure of performance across all possible classification thresholds. A higher AUC indicates better discrimination ability of the model.

4.2 Logging & Visualization

To monitor the training process and facilitate performance analysis, various logging and visualization techniques are employed:

- **Training History:** The training history, including loss and accuracy curves for both training and validation sets, is saved. These plots are crucial for identifying overfitting or underfitting and understanding the learning dynamics of the model over epochs.
- **TensorBoard Integration:** For TensorFlow/Keras models, TensorBoard is utilized to visualize training vs. validation metrics, model graphs, and other relevant data. This interactive visualization tool provides deep insights into the model's behavior during training.
- **Model Checkpoints:** Model checkpoints are recorded at epochs where the validation accuracy shows improvement. This ensures that the best performing model weights are saved, allowing for restoration and further use without needing to retrain from scratch.

4.3 Error Analysis

Beyond quantitative metrics, a qualitative error analysis is performed to gain deeper insights into the model's limitations and identify common failure patterns. This involves:

- **Identifying Misclassified Samples:** Specific instances where the model made incorrect predictions are identified, with a particular focus on false negatives in critical disease categories (e.g., missing AMD or DME). Understanding why these crucial cases are misclassified is paramount for clinical applicability.
- **Visual Inspection:** For a representative sample of misclassified images, the predicted label is overlaid with the true label on the original OCT image. This visual inspection helps in understanding the characteristics of images that are challenging for the model. Common failure patterns might include low image quality, atypical disease presentations, or subtle visual cues that the model failed to capture.

This detailed error analysis provides actionable insights for future model improvements, such as targeted data augmentation strategies or architectural modifications to address specific types of misclassifications.

5. Deployment (Optional)

While the primary focus of this project is on model development and evaluation, the potential for real-world application necessitates consideration of deployment strategies. A lightweight inference pipeline can be developed to demonstrate the model's utility in a practical setting.

5.1 Lightweight Inference Pipeline

To enable on-demand predictions, the final trained model can be exported into a production-ready format. For TensorFlow/Keras models, this typically involves saving the model in the TensorFlow SavedModel format or converting it to ONNX (Open Neural Network Exchange) for broader compatibility across different inference engines. A simple inference script would then be developed to:

- Load an OCT image.
- Preprocess the image according to the same pipeline used during training (resizing, normalization).
- Run the preprocessed image through the loaded model to obtain predictions.
- Output the top-1 predicted label along with its associated probability.

For demonstration purposes, this model and script can be packaged into a Docker container, ensuring portability and reproducibility across different environments. Alternatively, a Flask API endpoint can be created, allowing the model to serve predictions via HTTP requests, which is a common approach for integrating machine learning models into web applications.

5.2 Documentation

Comprehensive documentation is crucial for ensuring the reproducibility and usability of the project. A `README.md` file would be provided with detailed instructions on how to set up the environment, download the data, train the model, and run inference. This documentation would include:

- **Environment Setup:** Instructions for installing necessary libraries and dependencies.
- **Data Download:** Steps to programmatically fetch and organize the OCTDL dataset.
- **Training the Model:** Commands and configurations required to initiate the training process, including sample commands like `python train.py --epochs 30 --backbone resnet50`.
- **Running Inference:** Guidance on how to use the `inference.py` script for making predictions on new OCT images.
- **Expected Directory Structure:** A clear outline of the project's file and folder organization.
- **Hardware Recommendations:** Suggestions for computational resources, such as GPU specifications and RAM requirements, to ensure efficient execution of the training and inference processes.

This documentation ensures that anyone can replicate the project's results and integrate the developed models into their own systems.

6. Expected Deliverables

To ensure a complete and reproducible project, the following deliverables are anticipated:

6.1 Code Repository

The codebase will be organized into modular scripts, each responsible for a specific part of the pipeline:

- `data_preprocessing.py`: Contains scripts for downloading, preprocessing, and augmenting the OCTDL dataset.
- `model_definition.py`: Defines the backbone architecture, custom classification head, and model compilation settings.
- `train.py`: Implements the training loop, including callbacks for early stopping and checkpointing, and logging logic.

- `evaluate.py` : Script for computing evaluation metrics and generating visualizations such as confusion matrices and ROC curves.
- `inference.py` : A lightweight script for performing on-demand predictions on new OCT images.

6.2 Trained Model Files

Key model artifacts will be provided to facilitate immediate use and further development:

- **Checkpoint File(s):** The best performing model weights based on validation accuracy (e.g., `best_model.h5` or `best_model.pt`).
- **Final Exported Model:** The model saved in a deployable format (e.g., `saved_model/` directory for TensorFlow SavedModel or `model.onnx` for ONNX).

6.3 Report & Visuals

A comprehensive report, either in PDF format or as a Jupyter Notebook, will summarize the project's findings and methodologies. This report will include:

- **Dataset Description:** Details about the OCTDL dataset and its class distribution.
- **Training/Validation Plots:** Visualizations of accuracy and loss curves over epochs.
- **Confusion Matrix Heatmap:** A graphical representation of the model's classification performance.
- **ROC Curves per Class:** Plots illustrating the true positive rate against the false positive rate for each disease category.
- **Error Analysis:** Examples of misclassified images with accompanying comments, highlighting common failure patterns.
- **Brief Written Summary:** A concise (1–2 pages) overview of the methodology, key results, and any identified limitations.

6.4 README & Instructions

As mentioned in Section 5.2, a detailed `README.md` will provide all necessary instructions for reproducing the project, from environment setup to data download, training, evaluation, and inference. It will also include hardware recommendations to ensure optimal performance.

7. Evaluation Criteria for Success

The success of this project will be evaluated based on a combination of quantitative metrics, robustness, and reproducibility:

7.1 Quantitative Metrics

- **Overall Accuracy:** Aim for $\geqslant$ 90% overall accuracy on the test set, or the highest achievable given the complexity of the dataset.
- **Precision and Recall:** Strive for precision and recall values of $\geqslant$ 85% for each individual disease category.
- **Confusion Matrix:** Expect a clear confusion matrix demonstrating minimal overlap between classes, indicating effective discrimination.

7.2 Robustness

The model's robustness will be assessed by its ability to maintain stable performance even when subjected to minor image perturbations, such as slight changes in brightness or contrast. This ensures practical applicability in varied clinical imaging conditions.

7.3 Reproducibility

A key criterion for success is the reproducibility of the entire project. The documented codebase should allow any user to replicate the results, from data download to final evaluation, with minimal setup and intervention.

7.4 Clinical Relevance (Optional Extra Credit)

For extra credit, a discussion on the potential integration of this classifier into a clinical workflow will be provided. This could include scenarios such as flagging high-risk cases for immediate ophthalmologist review, thereby streamlining diagnostic processes and improving patient care.

8. Additional Notes

8.1 Ethical Considerations

Ethical considerations are paramount in medical AI applications:

- **Dataset Permissions:** Ensure that the OCTDL dataset's licensing permits academic and/or commercial use. Any restrictions will be clearly noted in the `README.md`.
- **Bias Mitigation:** Discuss potential biases within the dataset (e.g., underrepresented subpopulations) and outline strategies for their mitigation. This includes acknowledging limitations and promoting fairness in model predictions.

8.2 Hyperparameter Tuning

Optimal model performance often requires careful hyperparameter tuning. While not exhaustively covered in the provided notebook, strategies such as grid search or random search can be employed to find the best configuration for learning rates (e.g., {1e-3, 1e-4, 1e-5}), batch sizes (e.g., {16, 32}), and dropout rates (e.g., {0.3, 0.5}).

8.3 Compute Resources

Training deep learning models, especially with large datasets and complex architectures, is computationally intensive. It is recommended to use a machine equipped with at least one NVIDIA GPU with $\geqslant$ 8 GB VRAM. If a GPU is unavailable, alternative strategies such as reducing batch size or employing mixed precision training can be considered to manage computational demands.

This comprehensive documentation serves as a guide for understanding, replicating, and extending the deep learning solution for retinal disease classification, emphasizing the use of transfer learning for maximizing performance with limited data. The project aims to contribute to the advancement of AI in medical imaging, ultimately benefiting patient care.