# Execution Steps for Oral Cancer Histopathology Classification

This document outlines the comprehensive steps to execute a deep learning project for classifying oral cancer histopathology images (Normal vs. OSCC) using multiple pre-trained models (MobileNet, ResNet, InceptionV3, VGG16) in a Jupyter Notebook. The steps are designed to achieve high accuracy (≥95%), generate metrics (confusion matrix, classification report, training plots), and predict specific images (e.g., `Normal\_100x\_12.jpg`, `OSCC\_100x\_305.jpg`). The process assumes a Windows environment and a dataset structure with train, validation, and test folders (e.g., `C:\Users\shash\Desktop\Execution\_oral`).

## 1. Set Up the Development Environment

- \*\*Install Anaconda\*\*:

- Download and install Anaconda (Python 3.9 recommended) from anaconda.com.

- Follow the installation wizard, optionally adding Anaconda to the system PATH for easier command-line access.

- \*\*Create a Conda Environment\*\*:

- Open Anaconda Prompt.

- Create a new environment to isolate dependencies: `conda create -n oral\_cancer python=3.9`.

- Activate the environment: `conda activate oral\_cancer`.

- \*\*Install Required Libraries\*\*:

- In Anaconda Prompt (with `oral\_cancer` environment activated), install packages: `pip install opencv-python numpy pandas seaborn matplotlib scikit-learn tensorflow==2.10.0`.

- These libraries support image processing, data handling, visualization, and deep learning.

- Verify TensorFlow: `python -c "import tensorflow as tf; print(tf.\_\_version\_\_)"` (should output `2.10.0`).

- \*\*Set Up Jupyter Notebook\*\*:

- Install Jupyter if not present: `pip install jupyter`.

- Launch Jupyter Notebook: `jupyter notebook` (opens a browser-based interface).

## 2. Prepare the Dataset

- \*\*Organize Dataset\*\*:

- Ensure the dataset is structured in a directory (e.g., `C:\Users\shash\Desktop\Execution\_oral`) with subfolders:

- `train\Normal` and `train\OSCC` for training images.

- `val\Normal` and `val\OSCC` for validation images.

- `test\Normal` and `test\OSCC` for test images, including `Normal\_100x\_12.jpg` and `OSCC\_100x\_305.jpg`.

- Verify sufficient images (e.g., \~4946 train, \~120 validation, \~126 test).

- \*\*Check Image Accessibility\*\*:

- Confirm all images are accessible and not corrupted.

- Ensure `Normal\_100x\_12.jpg` and `OSCC\_100x\_305.jpg` are in the correct test subfolders.

- \*\*Adjust Dataset Paths\*\*:

- Note the base directory path (e.g., `C:\Users\shash\Desktop\Execution\_oral`).

- If located elsewhere (e.g., `D:\Data\OralCancer`), update paths in the code.

## 3. Configure the Jupyter Notebook

- \*\*Open the Notebook\*\*:

- In Jupyter Notebook (launched via `jupyter notebook`), navigate to the directory containing the project notebook (e.g., `ResNet101\_(final\_project).ipynb` or a new notebook).

- Open or create a notebook for the project.

- \*\*Organize Notebook Structure\*\*:

- Plan cells for: library imports, data preparation, model building, training, evaluation, plotting, and predictions.

- Ensure cells are sequential to avoid dependency errors.

- \*\*Update Paths in Code\*\*:

- Replace dataset paths with the correct base directory (e.g., `C:\Users\shash\Desktop\Execution\_oral`).

- Update model save paths and image paths for specific predictions.

- \*\*Save the Notebook\*\*:

- Save periodically: File &gt; Save and Checkpoint.

## 4. Implement and Execute the Code

- \*\*Import Necessary Libraries\*\*:

- Add a cell to import modules (e.g., `cv2`, `numpy`, `pandas`, `seaborn`, `matplotlib`, `sklearn`, `tensorflow`).

- Include TensorFlow imports for pre-trained models (e.g., `MobileNet`, `ResNet50`, `InceptionV3`, `VGG16`) and utilities.

- \*\*Prepare Data with Augmentation\*\*:

- Create a cell to define data generators for training, validation, and test sets using `ImageDataGenerator`.

- Apply augmentation (e.g., rotation, flips, zoom, brightness) to training data.

- Load images, resizing to the model’s input size (e.g., 224x224).

- Verify image counts (e.g., 4946 train, 120 validation, 126 test).

- \*\*Build and Configure Models\*\*:

- Add a cell for each model or a loop for MobileNet, ResNet, InceptionV3, VGG16.

- Load pre-trained models with ImageNet weights, freezing early layers.

- Add custom top layers (e.g., Dense, Dropout) with regularization for two classes (Normal, OSCC).

- Compile with an optimizer (e.g., Adam), loss (e.g., categorical crossentropy), and metrics (e.g., accuracy, precision, recall).

- \*\*Train the Models\*\*:

- Create a cell to train each model using data generators and callbacks:

- `EarlyStopping` to halt when validation accuracy plateaus.

- `ReduceLROnPlateau` to adjust learning rate.

- `ModelCheckpoint` to save the best model.

- Train in phases (initial training, fine-tuning) to optimize performance.

- Save models (e.g., `best\_model.h5`, `phase1\_model.h5`, `final\_model.h5`).

- \*\*Evaluate Model Performance\*\*:

- Add a cell to load the best saved model for each architecture.

- Evaluate on the test set, reporting accuracy, precision, and recall.

- Generate confusion matrix and classification report for test and entire dataset.

- Visualize confusion matrices using heatmaps.

- \*\*Plot Training History\*\*:

- Create a cell to plot training/validation accuracy/loss over epochs using Matplotlib.

- \*\*Predict Specific Images\*\*:

- Add a cell to load the best model and predict on `Normal\_100x\_12.jpg` and `OSCC\_100x\_305.jpg`.

- Preprocess images and display predictions with confidence scores and visualizations.

- \*\*Run All Cells\*\*:

- Select Cell &gt; Run All or run cells individually (Shift + Enter) to execute sequentially.

## 5. Monitor and Verify Outputs

- \*\*Check Library Imports\*\*:

- Confirm no import errors; modules should load successfully.

- \*\*Verify Data Loading\*\*:

- Ensure data generators report correct image counts (e.g., 4946 train, 120 validation, 126 test).

- Check class labels (Normal, OSCC).

- \*\*Monitor Training\*\*:

- Observe logs for increasing accuracy and decreasing loss.

- Confirm models are saved (e.g., check for `.h5` files).

- \*\*Validate Evaluation Metrics\*\*:

- Expect test accuracy ≥95%, with high precision and recall.

- Verify confusion matrices show high true positives.

- Ensure classification reports reflect balanced performance.

- \*\*Inspect Training Plots\*\*:

- Confirm training/validation accuracy converge to ≥95% without overfitting.

- Check loss decreases steadily.

- \*\*Confirm Predictions\*\*:

- Verify `Normal\_100x\_12.jpg` is predicted as Normal and `OSCC\_100x\_305.jpg` as OSCC, with ≥95% confidence.

- Ensure images display with prediction labels.

## 6. Troubleshoot and Optimize

- \*\*Handle Import Errors\*\*:

- Reinstall missing libraries (e.g., `pip install tensorflow==2.10.0`).

- \*\*Fix Path Issues\*\*:

- Double-check dataset paths and folder structure if loading fails.

- Update paths if dataset or model save location differs.

- \*\*Address Low Accuracy\*\*:

- Extend epochs, adjust augmentation, or fine-tune more layers if accuracy is &lt;95%.

- Check dataset quality (e.g., mislabeled images).

- \*\*Resolve Memory Issues\*\*:

- Reduce batch size (e.g., from 32 to 16) for memory errors.

- \*\*Debug Prediction Errors\*\*:

- Verify file paths and model loading if images are misclassified.

- Retrain or use a different saved model if needed.

## 7. Document and Save Results

- \*\*Save the Notebook\*\*:

- Save after execution: File &gt; Save and Checkpoint.

- \*\*Export Outputs\*\*:

- Copy confusion matrices, classification reports, and plots into a Word document.

- Note metrics (e.g., test accuracy) and prediction results.

- \*\*Archive Models\*\*:

- Store saved model files (e.g., `best\_model.h5`) for future use.

- \*\*Back Up Dataset\*\*:

- Keep a dataset copy for reproducibility.

## Notes

- \*\*Purpose\*\*: This framework supports classification using MobileNet, ResNet, InceptionV3, and VGG16, adaptable to any pre-trained model.

- \*\*Expected Outcome\*\*: With \~5192 images, models should achieve ≥95% test accuracy and correctly classify specified images.

- \*\*Time Consideration\*\*: Training may take hours on a CPU; a GPU reduces time significantly.

- \*\*Customization\*\*: Adjust paths, batch sizes, or epochs based on system and dataset specifics.