**Pathology Detection Using DenseNet121 - Full Documentation**

**1. Project Overview**

This project involves building a deep learning model to classify pathology images, specifically targeting **hilar pathology** from medical images. The pipeline includes preprocessing raw medical data, organizing it for training, training a deep learning classifier using DenseNet121, and finally evaluating the model's performance using various classification metrics.

**2. Folder and File Structure**

Here's an outline of the key components and scripts involved:

/pathology\_project/

├── DICOM Images/

├── PNG Images/

├── cleaned\_csv.csv

├── JPEG Folder Split/

├── models\_hilar2/

│ └── densenet121\_12.pth

├── val\_hilar2.csv

├── val\_preds\_hilar12.csv

├── Excel update 4.ipynb # DICOM to PNG

├── Preprocess 3.ipynb # CSV Cleaning

├── Seperating JPEG 1.ipynb # JPEG Extraction

├── Clean folder seperating 2.ipynb # Folder Splitting

├── train\_hilar 5.ipynb # Model Training

└── Validation 6.ipynb # Model Validation

**3. Step-by-Step Pipeline**

**Step 1: DICOM to PNG Conversion (Excel update 4.ipynb)**

* This notebook handles the conversion of DICOM images to .png format.
* Steps:
  + Loop over each patient/study DICOM folder.
  + Read image using pydicom.dcmread()
  + Normalize pixel values to [0, 255]
  + Save as .png using PIL.Image

**Step 2: Preprocessing CSV (Preprocess 3.ipynb)**

* A master CSV file is created which maps each image path to its label.
* Tasks:
  + Load and clean the original metadata.
  + Filter based on relevant fields such as hilar labels.
  + Save cleaned dataframe with columns like png\_path, hilar, etc.

**Step 3: JPEG Separation (Seperating JPEG 1.ipynb)**

* Converts and collects .jpg or .jpeg files from multiple directories.
* Stores them in a flat structure or organized folder for easy access.
* Extracts JPEGs for patients with valid labels only.

**Step 4: Clean Folder Splitting (Clean folder seperating 2.ipynb)**

* Organizes the dataset into:
  + train/
  + val/
  + test/
* Moves files to respective folders based on split logic (e.g., 80-10-10 rule or custom split)

**4. Model Training (train\_hilar 5.ipynb)**

**Model Architecture**

* A custom PyTorch class that uses torchvision.models.densenet121
* The final classifier layer is modified:

self.model.classifier = nn.Linear(1024, 1)

**Dataset Class**

* Custom dataset class loads image and label from dataframe:

img = Image.open(img\_path).convert('RGB')

label = np.array([label\_hilar])

* Images are transformed using:

transforms.Compose([

transforms.Resize((512, 512)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

])

**Training Loop**

* Trains for multiple epochs.
* Calculates loss.backward() and optimizer.step().
* Tracks loss and accuracy per epoch.
* Saves best model in models\_hilar2/densenet121\_12.pth

**5. Model Validation (Validation 6.ipynb)**

**Loading Model**

* Loads the trained weights:

model.load\_state\_dict(torch.load('models\_hilar2/densenet121\_12.pth'))

**Running Inference**

* Loads validation CSV: val\_hilar2.csv
* Creates test dataset and dataloader
* Performs inference using:

with torch.no\_grad():

pred = model(x)

**Prediction Saving**

* Predictions are post-processed using sigmoid:

def sigmoid(x):

return 1 / (1 + np.exp(-x))

* Predictions saved to CSV val\_preds\_hilar12.csv

**6. Evaluation Metrics**

After model inference, the following metrics are calculated:

acc = ((y\_preds > 0).astype('uint8') == y\_true).mean(0)

auc = metrics.roc\_auc\_score(y\_true, y\_preds, average=None)

recall = metrics.recall\_score(y\_true, (y\_preds > 0).astype('uint8'), average=None)

precision = metrics.precision\_score(y\_true, (y\_preds > 0).astype('uint8'), average=None)

f1 = metrics.f1\_score(y\_true, (y\_preds > 0).astype('uint8'), average=None)

Printed output:

hilar: Acc: 0.XXXX | AUC: 0.XXXX | Recall: 0.XXXX | Precision: 0.XXXX | F1: 0.XXXX

**7. Notes and Suggestions for Improvement**

* **Data Augmentation:** Currently only basic transforms are applied. Consider augmentations like random crop, horizontal flip.
* **Hyperparameter Tuning:** Tune learning rate, batch size, and number of epochs for better performance.
* **Multi-label Support:** Easily extendable to handle multiple pathologies if label columns are available.
* **Model Export:** You can export the model using torch.jit.trace or torch.save for deployment.
* **UI/Serving:** Consider wrapping inference in a FastAPI or Flask app for serving.

**8. Summary**

This documentation outlines a complete ML pipeline to detect hilar pathology using DenseNet121, including preprocessing, training, and evaluating the model with high granularity. With minor improvements, this project can be production-ready or used for publishing results.