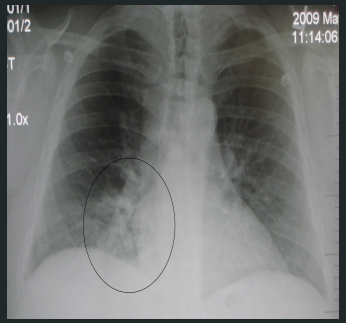


Multilabel Image Classification

Chest X-Ray images

Matt Harper
Pete Sharma
Rajeev Daithankarr

Project Overview

- In this project, we utilize a comprehensive dataset comprising 5600 X-ray images along with accompanying metadata, including patient demographics, image details, and disease labels.
- Our primary objective is to develop a Convolutional Neural Network (CNN) model capable of automatically detecting and classifying diseases from X-ray images. The disease labels, essential for training the model, are included in the dataset.
- Furthermore, we aim to create an intuitive user interface that allows users to upload new X-ray images and receive predictions of potential diseases, leveraging our trained CNN model. This interface will make our model accessible and practical for real-world medical diagnostics.

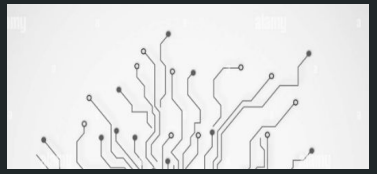
Background

Image classification using deep learning models can assist radiologists by providing consistent and rapid analysis of X-ray images.

ResNet50, a well-established deep learning model known for its deep residual learning framework, has shown exceptional performance in image classification tasks.

TorchVision, a library within the PyTorch ecosystem, offers pre-trained models and tools that facilitate efficient development and training of such models.

Technical Approach

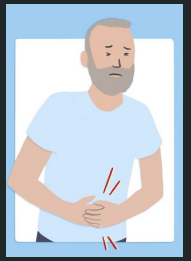


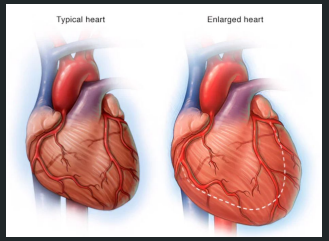
The project involves building a multi-label classification system that can simultaneously detect pneumonia and identify other related conditions from X-ray images.

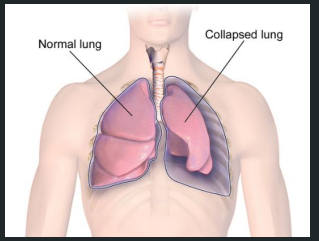
This approach will utilize deep learning techniques to analyze image features and predict multiple labels for each image.

The key steps include data preparation, model selection and modification, training, and evaluation.

Data







Dataset Collection:

- **Source**: Dataset of X-ray images with corresponding labels. https://www.kaggle.com/datasets/nih-chest-xrays/sample?resource=download-directory
  - sample.zip: Contains 5,606 images with size 1024 x 1024,convert to 224x224 for training the model.
  - sample_labels.csv: Class labels and patient data for the entire dataset
    i. Image Index: File name
    ii. Finding Labels: Disease type (Class label)
    iii. Follow-up #
    iv. Patient ID
    v. Patient Age
    vi. Patient Gender
    vii. View Position: X-ray orientation
    viii. OriginalImageWidth
    ix. OriginalImageHeight
    x. OriginalImagePixelSpacing_x
    xi. OriginalImagePixelSpacing_y
- Image Format: PNG and label CSV

- **Class :**
  - Hernia
  - Pneumonia
  - Fibrosis
  - Edema
  - Emphysema
  - Cardiomegaly
  - Pleural_Thickening
  - Consolidation
  - Pneumothorax
  - Mass
  - Nodule
  - Atelectasis
  - Effusion
  - Infiltration

Data Preparation

Preprocessing:

- **Handling Multi-Label Classification:**
  - **Label Encoding:** For multi-label classification, CSV file `encoded_labels.csv`
  - `index,labels`
  - `00000013_005.png,[0 0 0 0 0 1 0 0 1 0 0 1 0 1]`
  - `00000013_026.png,[0 1 0 0 0 1 0 0 0 0 0 0 0 0]`
  - `00000017_001.png,[0 0 0 0 0 0 0 0 0 0 0 0 0 0]`

- **Split the dataset into training and validation sets**
  - `train_image_paths`,
  - `val_image_paths`,
  - `train_labels`,
  - `val_labels`
- **Custom Dataset Class**

Data Preparation - cond

Preprocessing:

- **Transformation**
  - **Resizing**: ResNet50 typically expects input images of size 224x224 pixels.
    - `transforms.Resize((224, 224))`
  - **Compatibility with neural network -** Neural network (ResNet50) operates on numerical data. Images as stored in raw format (PNG) and these files need to be converted to numerical format to process.
    - Convert image to tensor - `transforms.ToTensor()`
  - **Normalization:** Normalize the images using the mean and standard deviation of the ImageNet dataset, as ResNet50 was pre-trained on ImageNet. This ensures that the input distribution is similar to what the model was originally trained on.

Data Preparation - contd

Dataset and DataLoader:

- **Dataset Creation:** Creation of customized Dataset class for multi-label classification.

```
train_dataset = CustomDataset(train_image_paths, train_labels, transform=transform)
val_dataset = CustomDataset(val_image_paths, val_labels, transform=transform)
```

- **DataLoader:** It is used to efficiently manage and load data in batches, which is essential for training models.
  - Batching: Splitting data into smaller batches, which helps better utilization of computational resources
  - Shuffling: Randomly shuffling of data to ensure that each batch is representative of the overall dataset, which can improve model generation.

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

- **Define the modified ResNet-50 model for grayscale images** - ResNet50 is designed for RGB images, hence need to modify grayscale images

Training the model

- **Loss Function :** Also known as cost function or objective function. It quantifies difference between model's prediction and actual target values. It provides how well model is performing . The objective is keep loss function minimum.

  Loss function for multi-label classification needs to handle multiple binay labels per instance.

  Binary Cross Entropy with logit Loss - `criterion = nn.BCEWithLogitsLoss()` It is combination of sigmoid layer and binary cross-entropy loss.

- **Optimizer :** It is algorithm to find optimal set of parameters (weights and bias) for model by minimizing loss function

  Adam (adaptive moment estimation) - `optimizer = optim.Adam(model.parameters(), lr=0.001)`

- **Training loop** : A training loop is a sequence of operation that repeatedly process batches of data, complutes loss and updates model's parameters to minimize the loss. Each iteration in call epoch

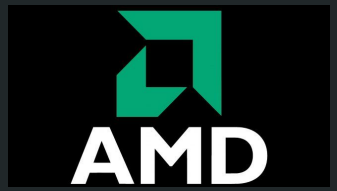
- **Validation Loop** : It is used to evaluate the model's performance on a separate validation that the model has not seen during training.

  Validation Accuracy per class: 0.9491341710090637

  Average Validation Accuracy: 94.91341710090637%

- **Saving**: Save the model for future use or deployment (resnet50_scratch.pth)

GPU vs CPU



Training Efficiency: GPUs are highly recommended for training deep learning models like ResNet50 due to their parallel processing capabilities, which significantly reduce training times and handle larger batch sizes more effectively.

Cost and Access: GPUs, while faster, can be more expensive and require specialized access. CPUs are more accessible and cost-effective but are not suited for handling large-scale deep learning tasks efficiently.

Development and Debugging: CPUs can be simpler for development and debugging but will not match the speed and efficiency of GPUs during actual model training.

Recommendation: For most deep learning tasks involving complex models like ResNet50, using a GPU is preferable due to the substantial performance gains in training time and efficiency. CPUs are better suited for initial development, small-scale experiments, or environments where GPU resources are not available.

```
# Define the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


#load the model for CPU
model.load_state_dict(torch.load(('resnet50_scratch.pth'),
map_location=torch.device('cuda')))
```

User Interface - gradio

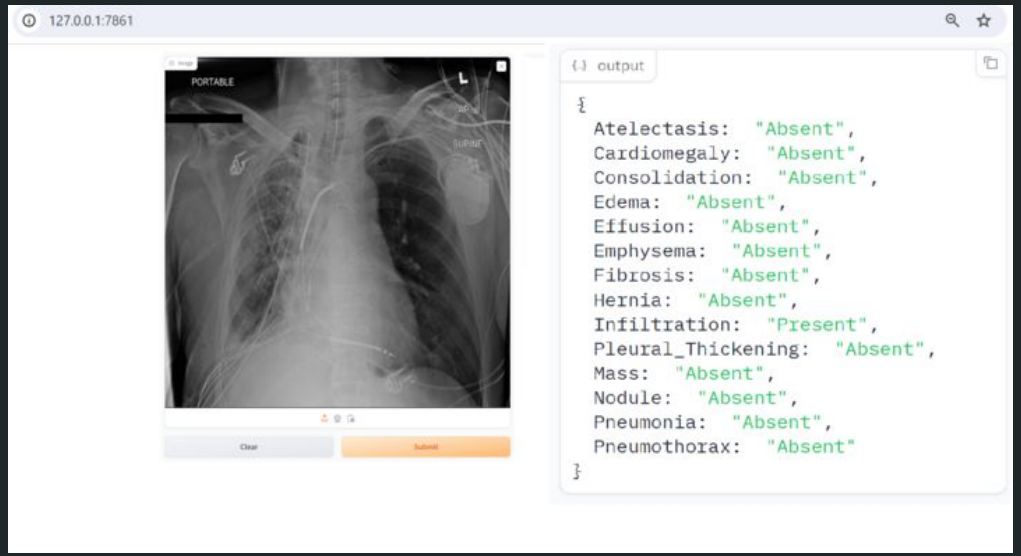
Install Gradio: Use `pip install gradio`.

Load Model: Load the saved model weights and set the model to evaluation mode.

Define Prediction Function: Process input images, run the model, and format the output as JSON.

Create Gradio Interface: Set up Gradio to handle image inputs and output JSON predictions.

Run and Test: Launch the Gradio interface to interact with your model through a web interface



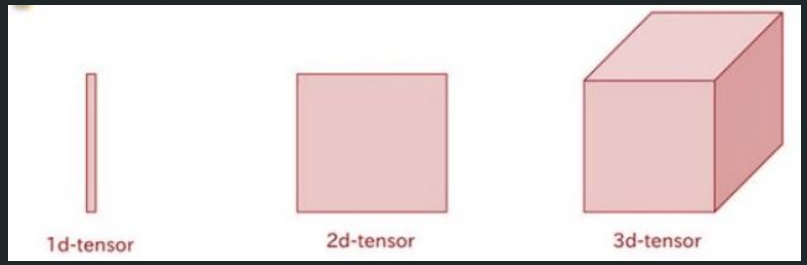
Expected Output

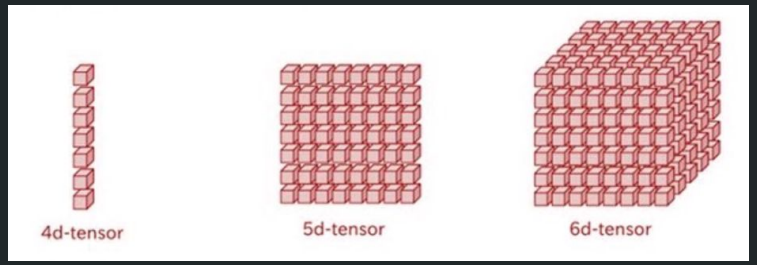
Framework: PyTorch will be used as the primary deep learning framework due to its flexibility and efficiency.

A fully trained multi-label classification model capable of detecting various pathologies in X-ray images.

Evaluation metrics demonstrating the model’s performance, including accuracy,

Deployment-ready model with gradio user interface.



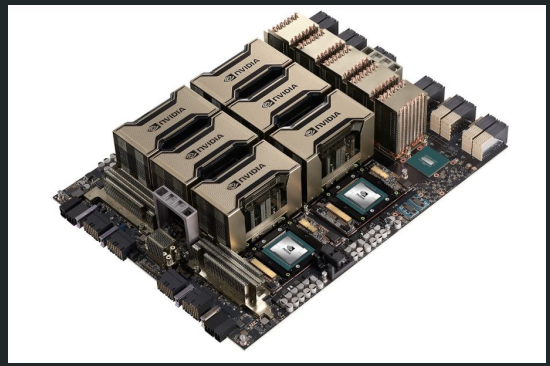


Challenges / Risks

Data Quality: Variability in image quality and inconsistency in labeling can impact model performance. Ensuring high-quality and well-labeled data is crucial.

Model Generalization: The model may face challenges in generalizing across different populations and imaging conditions. Continuous evaluation and updates will be necessary.

Computational Resources: Training deep learning models, especially with large datasets, can be computationally expensive and time-consuming. Adequate computational resources are required.



Future Work

Model Refinement: Further fine-tuning and improvements to the model may be needed based on additional data and performance feedback.

Dataset Expansion: Expanding the dataset to include a broader range of conditions and image variations will help improve model robustness.

Integration: Deploying the model in a real-world clinical setting involves integrating with existing systems and ensuring compliance with healthcare regulations.

Clinical Validation: Conducting real-world clinical trials to validate the model's effectiveness and safety in actual diagnostic scenarios.

User Interface Development: Creating a user-friendly interface for clinicians to interact with the model and integrate it into their diagnostic workflows.

Long-Term Monitoring: Implementing mechanisms for ongoing model evaluation and updates to ensure continued accuracy and relevance as new data and conditions emerge.

Thank You !