ABSTRACT

This project aims to address the pressing need for accurate and timely diagnosis of pneumonia through the development of a robust machine learning-based detection system. The researchers leverage the power of convolutional neural networks (CNNs) and Class Activation Maps (CAM) to enhance the diagnostic accuracy and interpretability of their model. Utilising the RSNA Pneumonia Detection Challenge dataset from Kaggle, the team meticulously trains and fine-tunes a modified ResNet18 architecture using the PyTorch Lightning framework. The experimental results demonstrate the efficacy of the proposed approach, where the classifier exhibits commendable performance in distinguishing between healthy and pneumonia-affected individuals. Crucially, the integration of CAM provides valuable insights into the decision-making process of the classifier, thereby improving transparency and trust in the diagnostic system. This project represents a significant stride towards revolutionising pneumonia diagnosis, offering both accurate detection and interpretable results, with the ultimate goal of contributing to improved healthcare outcomes in respiratory medicine

Keywords: Convolutional neural networks (CNNs), Class Activation Maps (CAM), RSNA Pneumonia Detection,ResNet18 architecture,PyTorch Lightning frame-work, Pneumonia diagnosis

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Medical imaging plays a pivotal role in the early detection and diagnosis of various diseases, enabling healthcare professionals to provide timely and effective treatment. Among the multitude of medical imaging modalities, chest X-ray imaging holds particular importance in the diagnosis of respiratory conditions, including pneumonia. Pneumonia, a prevalent respiratory disease characterized by inflammation of the lung tissue, poses significant health risks, especially in vulnerable populations such as children, the elderly, and immunocompromised individuals. Early detection and accurate diagnosis of pneumonia are crucial for initiating appropriate treatment and preventing complications.

Traditionally, the interpretation of chest X-ray images for the presence of pneumonia has relied on manual inspection by radiologists and medical experts. However, this process is inherently time-consuming, subjective, and resource-intensive, often leading to delays in diagnosis and potential errors due to human factors. Moreover, the increasing volume of medical imaging data poses challenges for healthcare systems, necessitating the development of automated solutions to assist in image analysis and diagnosis.

In response to these challenges, advancements in artificial intelligence (AI) and deep learning have paved the way for the development of automated

systems capable of analyzing medical images with high accuracy and efficiency. Leveraging deep learning models, particularly convolutional neural networks (CNNs), researchers have made significant strides in the development of automated diagnostic tools for various medical imaging tasks, including pneumonia detection from chest X-ray images.

In this context, this report aims to present a comprehensive overview of the development and evaluation of an automated pneumonia detection system using chest X-ray images. The project utilizes state-of-the-art deep learning techniques, specifically a modified version of the ResNet18 architecture, trained on a dataset comprising chest X-ray images labeled for the presence or absence of pneumonia. Through the integration of advanced image processing algorithms and interpretability techniques, such as Class Activation Maps (CAM), the system not only detects pneumonia but also provides insights into the regions of interest within the images contributing to the classification decision.

The following sections of the report will delve into the methodology employed for data preprocessing, model development, and training. Additionally, the evaluation metrics used to assess the performance of the developed system on a separate validation dataset will be discussed. Furthermore, the report will explore the interpretability of the model's predictions through visualization techniques, shedding light on the decision-making process underlying pneumonia detection.

Overall, this project represents a significant step towards the integration of AI-
driven solutions in medical imaging, with the potential to enhance diagnostic accuracy,
streamline workflow efficiency, and improve patient outcomes in the diagnosis

and management of pneumonia and other respiratory conditions. Through rigorous
experimentation and evaluation, the insights gained from this study aim to contribute
to the ongoing efforts in leveraging AI for healthcare innovation and advancing the
field of medical image analysis.

1.1 LITERATURE REVIEW

1.1.1 Automated Pneumonia Detection

- The detection of pneumonia from chest X-ray images using automated methods has garnered significant attention in recent years.
- Numerous studies have explored the effectiveness of deep learning techniques, particularly convolutional neural networks (CNNs), in accurately identifying pneumonia patterns in medical images.

1.1.2 Model Architectures

- Researchers have employed various CNN architectures for pneumonia detection, each with its strengths and limitations.
- Commonly used architectures include ResNet, DenseNet, and Inception networks.
- Studies have proposed modified architectures, such as a modified DenseNet model for pneumonia detection.

1.1.3 Data Augmentation and Preprocessing

- Effective data augmentation techniques play a crucial role in enhancing the generalization and robustness of pneumonia detection models.
- Researchers have explored various augmentation strategies, including random rotations, translations, and scaling.
- Preprocessing steps such as normalization and resizing are essential for standardizing input images and facilitating model training.

1.1.4 Interpretability and Explainability

- Researchers have explored interpretability techniques such as Class Activation Maps (CAM), Grad-CAM, and attention mechanisms to visualize the regions of interest within chest X-ray images contributing to the model's decision.
- These techniques provide insights into the features learned by the model and aid in understanding its decision-making process, fostering trust and acceptance in clinical settings.

1.1.5 Performance Evaluation Metrics

- The evaluation of automated pneumonia detection models relies on various performance metrics to assess their accuracy, sensitivity, specificity, and overall performance.
- Commonly used metrics include area under the receiver operating characteristic curve (AUC-ROC), precision-recall curves, accuracy, sensitivity, specificity, and F1 score.

1.1.6 Challenges and Future Directions

- Challenges include data scarcity, class imbalance, and model generalization to diverse populations and imaging conditions.

- Future research directions include the development of robust transfer learning techniques, domain adaptation methods, and multimodal fusion approaches.
- Integration of clinical metadata, patient demographics, and longitudinal data may further enhance the performance and clinical utility of pneumonia detection models.

1.1.7 Ethical and Regulatory Considerations

- The deployment of automated pneumonia detection systems in clinical practice raises ethical and regulatory concerns regarding patient privacy, data security, and algorithm bias.
- Researchers and policymakers must address these concerns through transparent reporting, rigorous validation, and adherence to ethical guidelines.

1.2 SCOPE

The scope of this project encompasses the development and implementation of an automated system for pneumonia detection from chest X-ray images using deep learning techniques. The project aims to leverage advancements in convolutional neural networks (CNNs) and image processing to create a robust and accurate solution capable of assisting healthcare professionals in early pneumonia diagnosis. Specifically, the project will involve the exploration of different CNN architectures, including ResNet, DenseNet, and Inception networks, to identify the most suitable model for pneumonia detection. Additionally, the scope extends to data preprocessing and

augmentation techniques to enhance the quality and diversity of the training dataset, thereby improving model generalization and performance. Interpretability methods, such as Class Activation Maps (CAM) and Grad-CAM, will be investigated to visualize the regions of interest within chest X-ray images and provide insights into the model's decision-making process. Furthermore, the project will focus on evaluating the performance of the developed system using standard metrics such as accuracy, sensitivity, specificity, and area under the receiver operating characteristic curve (AUC-ROC). The ultimate goal is to deploy a clinically viable solution that can assist radiologists and healthcare practitioners in pneumonia diagnosis, leading to timely interventions and improved patient outcomes. Ethical and regulatory considerations regarding patient privacy, data security, and algorithm bias will also be addressed to ensure the responsible deployment of the automated pneumonia detection system.

1.2.1 Problem Statement

Pneumonia stands as a significant burden on public health worldwide, accounting for a substantial portion of respiratory-related morbidity and mortality. Timely and accurate diagnosis of pneumonia is paramount for initiating appropriate treatment and preventing adverse outcomes.

Firstly, manual interpretation of chest X-rays demands considerable time and resources, particularly from skilled healthcare professionals. This process involves meticulous scrutiny of radiographic images to identify subtle signs indicative of pneumonia, such as infiltrates or consolidations. Such manual

analysis not only consumes valuable time but also places a strain on healthcare systems, especially in settings with limited access to specialized expertise.

Moreover, the accuracy of pneumonia diagnosis heavily relies on the proficiency and experience of the interpreting radiologist or physician. Interpreting chest X-rays requires a nuanced understanding of radiographic patterns and the ability to discern between normal anatomical variations and pathological changes associated with pneumonia. Consequently, misinterpretation or oversight of relevant findings can lead to delayed diagnosis or misdiagnosis, compromising patient care and outcomes.

In light of the challenges of Pneumonia, there arises a critical need for an automated system capable of accurately detecting pneumonia from chest X-ray images. Such a system would not only alleviate the burden on healthcare professionals but also enhance diagnostic efficiency and accuracy. By leveraging advancements in machine learning and computer vision, an automated pneumonia detection system holds the promise of expediting diagnosis, facilitating prompt treatment initiation, and ultimately improving patient outcomes.

Therefore, the primary objective of this project is to develop and validate a robust and reliable automated system for pneumonia detection from chest X-ray images. By harnessing the power of artificial intelligence and deep learning algorithms, we aim to bridge the gap between traditional diagnostic methods and the evolving landscape of medical imaging, thereby contributing to the advancement of respiratory healthcare practices.

1.2.2 System Overview

The system overview of the pneumonia prediction project involves the development and deployment of an automated system capable of accurately detecting pneumonia from chest X-ray images. At its core, the system consists

of several key components: Data preprocessing, model development, training, evaluation, and inference.

Initially, raw chest X-ray images are processed to extract relevant features and normalize pixel intensities. Subsequently, a deep learning model, such as ResNet18, is constructed and trained using the preprocessed data to learn discriminative patterns indicative of pneumonia presence. During training, the model undergoes iterative optimization using techniques like Adam optimization and binary cross-entropy loss to improve its predictive performance. Once trained, the model is evaluated using metrics like accuracy, precision, recall, and confusion matrices to assess its effectiveness in pneumonia detection. Finally, the trained model is deployed for inference, where it receives new chest X-ray images as input and produces predictions regarding the presence or absence of pneumonia. Overall, the system aims to provide a reliable and efficient tool for early detection of pneumonia, thereby facilitating timely medical intervention and improving patient outcomes.

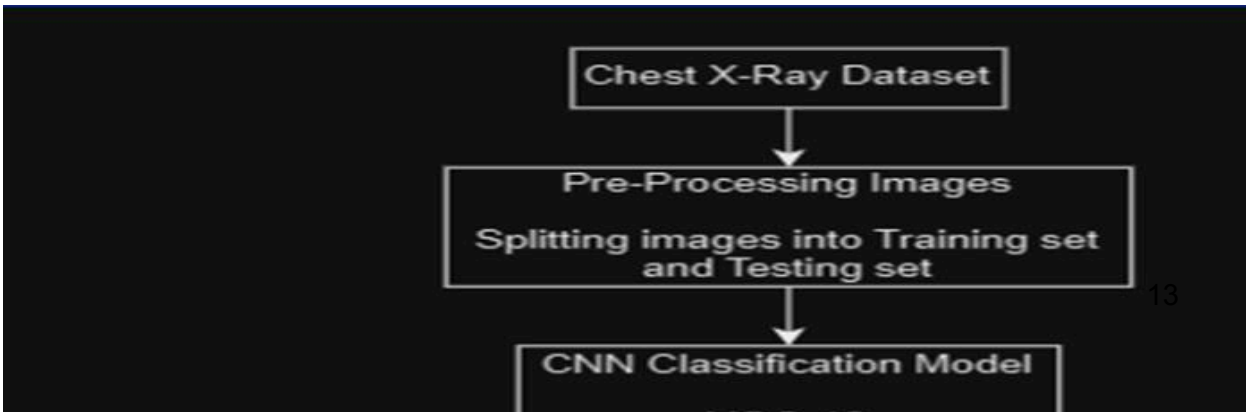


Figure 1.1: The overview of the model

1.2.3 Workflow

The workflow for the pneumonia detection project involves several sequential steps aimed at developing and deploying an automated system for pneumonia detection from chest X-ray images. Initially, the workflow begins

with data preprocessing, where raw chest X-ray images are collected and prepared for model training. This step may include tasks such as resizing images, normalizing pixel values, and splitting the dataset into training and validation sets.

Following data preprocessing, the next step is model development. In this phase, a deep learning architecture is selected, such as ResNet18, and adapted for the specific task of pneumonia detection. The model architecture is constructed, specifying the layers, activation functions, and other parameters necessary for effective learning from the input images. Once the model is defined, the workflow proceeds to the training phase. During training, the model is fed batches of chest X-ray images from the training dataset, and its parameters are optimized iteratively to minimize a predefined loss function. Optimization techniques such as Adam optimization may be employed to update the model's parameters efficiently.

As training progresses, the model's performance is evaluated using a separate validation dataset. Metrics such as accuracy, precision, recall, and F1 score are calculated to assess the model's ability to correctly classify pneumonia cases. This validation step helps monitor the model's generalization capabilities and identify potential overfitting or underfitting issues.

After the model has been trained and validated, the workflow moves to the

inference phase. Here, the trained model is deployed to make predictions on new, unseen chest X-ray images. The model takes these images as input and outputs predictions indicating whether pneumonia is present or not. These predictions can then be used to assist healthcare professionals in diagnosing and treating patients effectively.

Throughout the entire workflow, careful attention is paid to ensure reproducibility, interpretability, and scalability of the developed system.

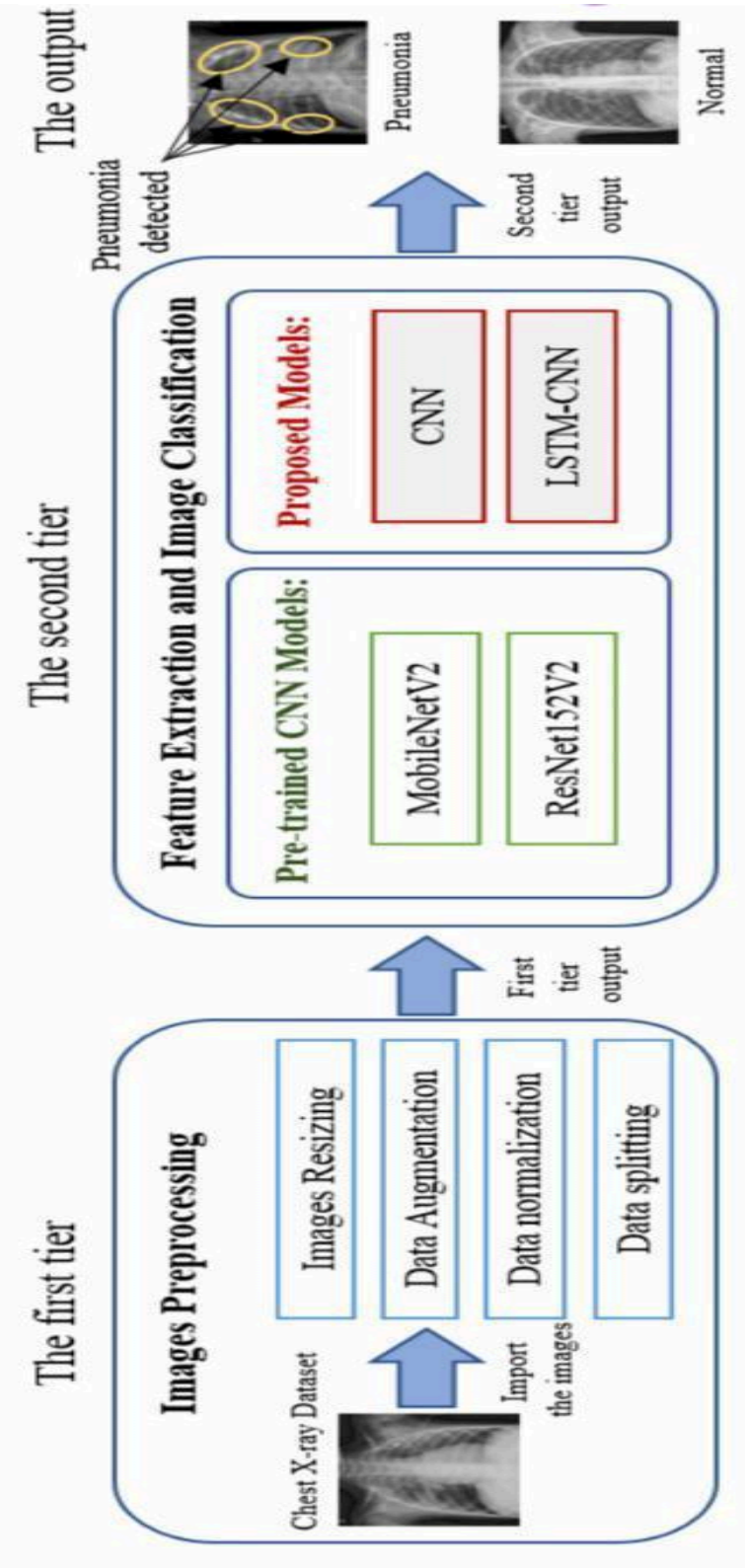


Figure 1.2: The workflow of the model

1.2.4 Input Layer

- The input to the model is a grayscale chest X-ray image. Since the ResNet18 architecture was originally designed for RGB images, adjustments are made to accommodate single-channel (grayscale) input images.

1.2.5 Convolutional Layers

- ResNet18 starts with a convolutional layer that extracts features from the input image. This layer applies a set of learnable filters to the input image, producing a set of feature maps. The convolutional layers are stacked to capture increasingly complex patterns in the input image.

1.2.6 Residual Blocks

- ResNet18 introduces residual learning to tackle the vanishing gradient prob-lem in deep networks. It employs residual blocks with two convolutional layers, batch normalization, ReLU activation, and a skip connection. This connection al-lows the network to learn residual mappings, aiding in gradient flow during training and enabling effective learning of deeper representations.

1.2.7 Pooling Layers

- After several residual blocks, the feature maps are downsampled using max-pooling layers. Max-pooling reduces the spatial dimensions of the feature maps while retaining the most important information, aiding in translation invariance and reducing computational complexity.

1.2.8 Fully Connected Layers

- Downscaled feature maps are flattened and fed into fully connected layers, also called dense layers. These layers classify by learning patterns in the feature maps and mapping them to output classes. In pneumonia detection, the final fully connected layer usually outputs a single value, representing the probability of pneumonia presence.

1.2.9 Output Layer

- The output layer of a pneumonia detection system typically uses a sigmoid activation function, yielding a probability score between 0 and 1 for pneumonia presence. A threshold, often 0.5, is then used to classify the image as either pneumonia-positive or pneumonia-negative.

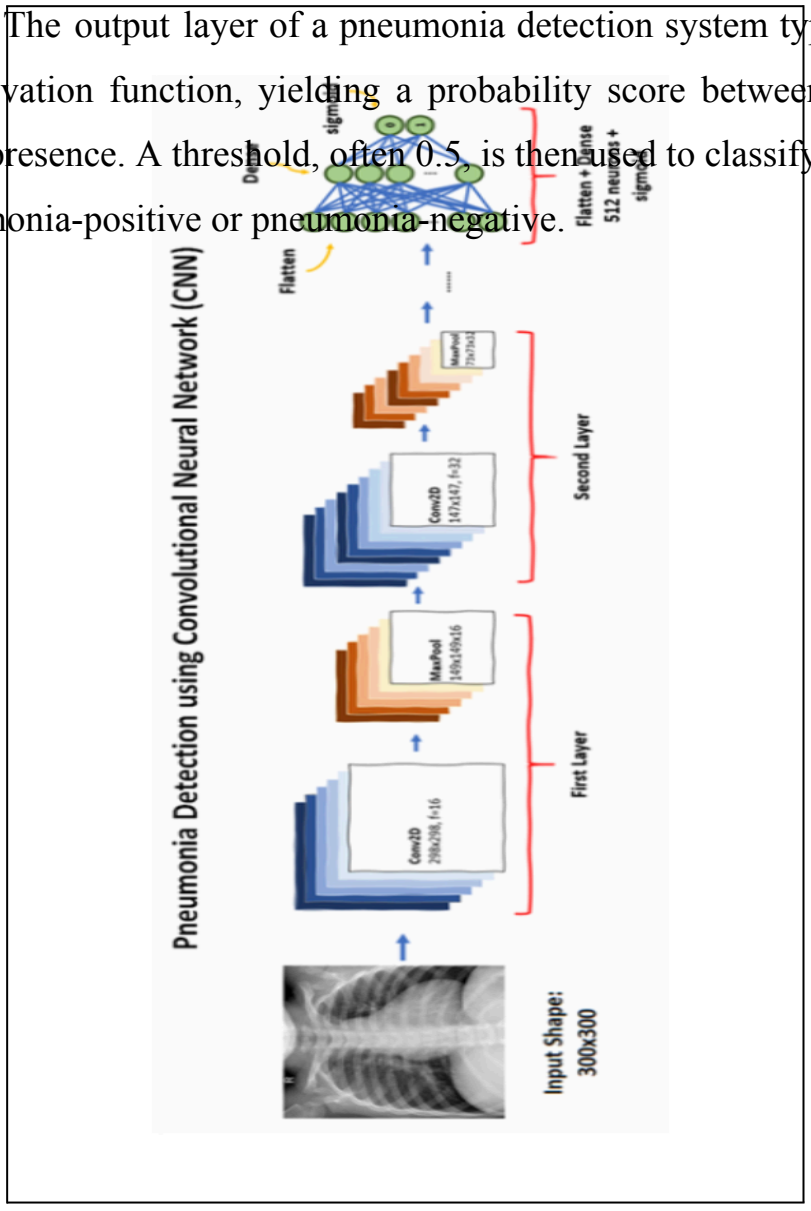


Figure 1.3: The model Architecture

DATASET

1.2.7 Dataset

- The dataset used for the RSNA Pneumonia Detection Challenge, hosted on Kaggle, comprises 26,684 chest X-ray images. These images serve as the primary data source for developing algorithms aimed at automatically detecting pneumonia. The dataset is divided into two categories based on the presence or absence of pneumonia:

Images without Pneumonia:

- This subset consists of 20,672 chest X-ray images where no signs of pneumonia are observed. These images serve as the negative class in the classification task and are crucial for training models to recognize normal chest X-rays.

Images with Pneumonia:

- This subset comprises 6,012 chest X-ray images showing manifestations of pneumonia. These images serve as the positive class in the classification task and are essential for training algorithms to identify patterns indicative of pneumonia in X-ray scans.

The dataset is valuable for machine learning research and development, as it provides a diverse range of chest X-ray images capturing various manifestations of pneumonia. The large size of the dataset allows for robust model training and evaluation, facilitating the development of accurate and reliable pneumonia detection algorithms. Additionally, the dataset's availability on Kaggle encourages collaboration

and innovation within the data science and medical imaging communities, fostering advancements in automated disease diagnosis and healthcare technology.

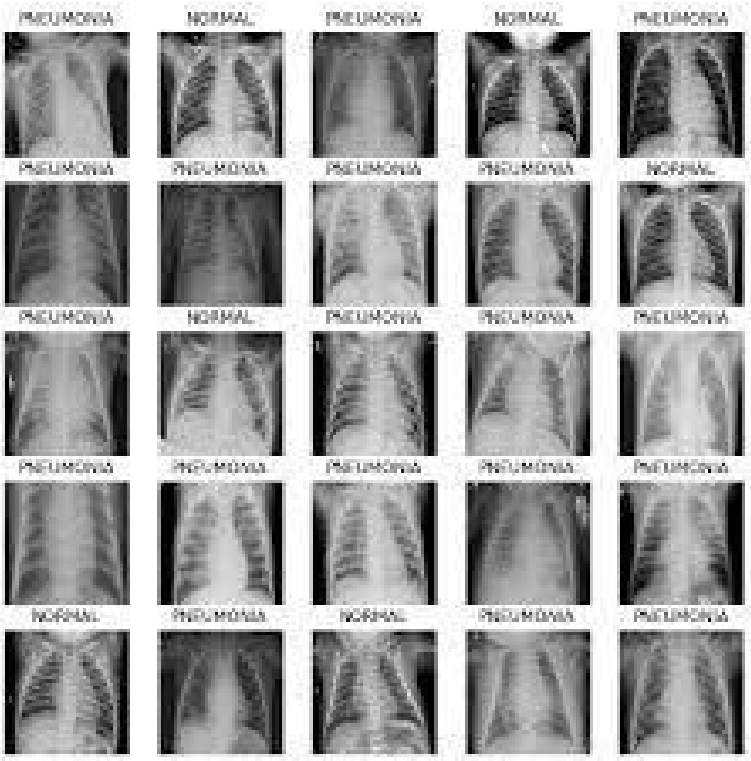


Figure 1.4: Dataset Overview

PREPROCESSING

1.3 Preprocessing

1.3.1 Importance of Preprocessing

Preprocessing is a crucial step in the development of machine learning mod-els, particularly for computer vision tasks like pneumonia detection from chest X-ray images. Effective preprocessing can have a significant impact on the performance and reliability of the final model. There are several key reasons why preprocessing is essential:

- Data Normalization: Standardizing the pixel values and resizing the images to a consistent size ensures that the data is presented to the model in a consistent format, facilitating effective learning and generalization.
- Feature Extraction: Preprocessing techniques, such as image enhancement and lung segmentation, can help extract more relevant features from the input data, improving the model's ability to identify patterns indicative of pneumonia.
- Robust Generalization: Data augmentation techniques introduce controlled variations to the training data, which can help the model generalize better and become more robust to real-world variations in chest X-ray images.
- Computational Efficiency: Resizing large images to a more manageable size can significantly reduce the computational resources required for model train-ing and inference, making the overall system more efficient.

1.3.2 Purpose of Preprocessing

The primary purpose of the preprocessing pipeline for the RSNA Pneumonia Detection Challenge dataset is to prepare the data in a format that is suitable for training and evaluating deep learning models for pneumonia detection. The specific goals of the

preprocessing steps are: Data Standardization: Ensuring all images have a consistent size, pixel value range, and normalization to facilitate effective model training and convergence.

- Feature Extraction Optimization: Enhancing the visibility of relevant features, such as lung regions, to help the model focus on the most informative parts of the chest X-ray images.
- Dataset Augmentation: Introducing controlled variations in the training data through techniques like rotation, translation, and scaling to improve the model's robustness and generalization capabilities.
- Efficient Data Handling: Organizing the preprocessed data in a format that can be easily loaded and utilized by the deep learning framework, such as PyTorch's DatasetFolder, to streamline the overall training and evaluation process.

By addressing these goals, the preprocessing pipeline prepares the RSNA Pneumonia Detection Challenge dataset to be effectively leveraged for training accurate and reliable pneumonia detection models, ultimately contributing to the advancement of automated medical imaging analysis and improved healthcare outcomes.

1.3.3 DICOM Reading & Effective Storage

- Conversion to Numpy Arrays: The DICOM images are read and converted into numpy arrays. This allows for efficient handling of the data within the DataLoader.
- Compute Mean and Standard Deviation: The overall mean and standard deviation of the pixel values across the entire dataset are computed for normalization purposes.
- Storage: The processed numpy images are stored in two separate folders based on their binary labels (0 for healthy, 1 for pneumonia). This facilitates easy retrieval and utilization of the data.

1.3.4 Standardization and Resizing

- Standardization: All images are standardized by dividing the pixel values by the maximum pixel value in the dataset, which is 255.
- Resizing: Due to the large size of the original images (1024x1024), they are resized to a more manageable size of 224x224. This resizing ensures compatibility with deep learning models while conserving computational resources.

1.3.5 Dataset Mean and Standard Deviation Computation

- Iterative Approach: The mean and standard deviation are computed iteratively using a trick to avoid loading the entire dataset into memory simultaneously.
- Summation of Pixel Values: The sums of pixel values and the sums of squared pixel values are computed for each image.
- Normalization: The computed sums are then normalized to obtain the mean and standard deviation of the dataset.

1.3.6 Data Augmentation

- Techniques Used: Various data augmentation techniques such as Random Rotation, Random Translations, Random Scales, and Random Resized Crops are applied to augment the dataset.
- Enhances Model Robustness: Data augmentation helps improve model generalization and robustness by introducing variations in the training data.

1.3.7 Storage and Organization

- Train-Validation Split: The dataset is split into training and validation sets, with approximately 4/5 of the data used for training and 1/5 for validation.
- Folder Structure: The preprocessed images are stored in folders correspond-ing to their class labels (0 for no pneumonia, 1 for pneumonia) within separate directories for training and validation data.
- Use of torchvision.DatasetFolder: The standardized and resized images are stored in a format compatible with torchvision.DatasetFolder, simplifying the data loading process during model training.

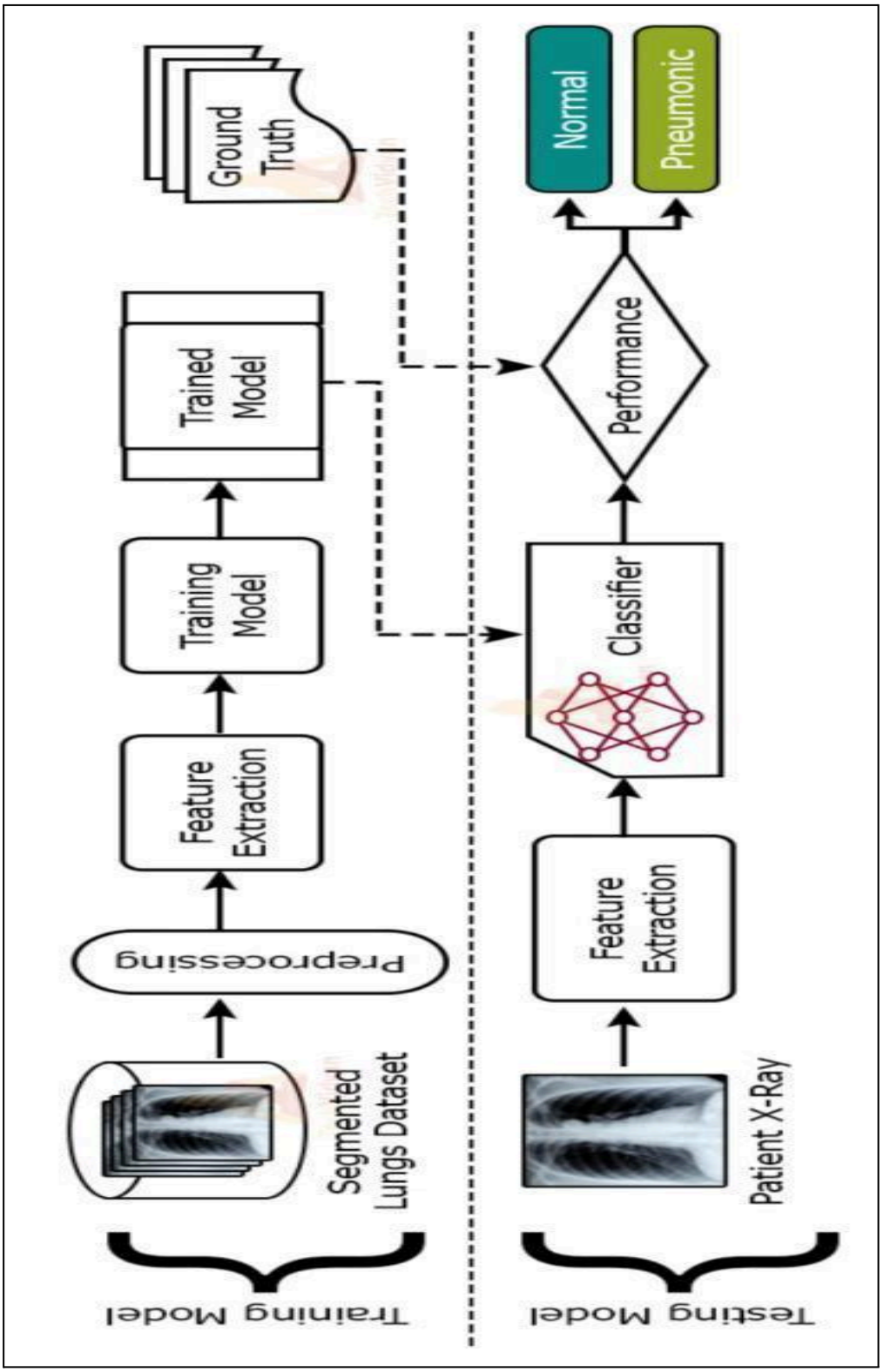


Figure 1.5: Preprocessing the data

1.3.8 Additional Preprocessing Techniques

In addition to the steps mentioned earlier, the preprocessing pipeline can be further enhanced by incorporating the following techniques:

- Image Quality Enhancement: Techniques such as histogram equalization, contrast adjustment, and image sharpening can be applied to improve the visual quality and contrast of the chest X-ray images, potentially leading to better model performance.
- Lung Segmentation: Extracting the lung regions from the chest X-ray images and focusing the model's attention on these regions can help improve the model's ability to identify pneumonia-related features.
- Cross-validation: Employing cross-validation strategies, such as k-fold cross-validation, can provide a more robust evaluation of the model's performance and help mitigate the impact of dataset biases.
- Anomaly Detection: Incorporating anomaly detection techniques can help identify and handle outliers or mislabeled instances in the dataset, further improving the model's generalization capabilities.

The comprehensive preprocessing pipeline ensures that the RSNA Pneumonia Detection Challenge dataset is appropriately prepared for training machine learning models. By standardizing, resizing, computing dataset statistics, and applying data augmentation techniques, the dataset is made suitable for training accurate and robust pneumonia detection algorithms. Moreover, the organized storage structure facilitates easy access and utilization of the preprocessed data during model development and evaluation. Incorporating additional preprocessing techniques, such as image quality enhancement, lung segmentation, cross-validation, and anomaly detection, can fur-ther enhance the effectiveness of the preprocessing pipeline and lead to more robust and reliable pneumonia detection models.

MODEL BUILDING

1.3.9 Initialization and Forward Function

Initialization

- In the init method, the model's architecture is defined using PyTorch modules such as nn.Linear, nn.Conv2d, nn.ReLU, nn.Dropout, etc. - The number of input and output features for each layer are specified, along with the size of convolutional filters, stride, and padding for convolutional layers. - Parameters such as the number of layers, hidden units, dropout probabilities, and other hyperparame-ters are set during initialization to define the model's capacity and complexity. - The model's layers are arranged in a specific sequence to define the flow of data through the network.

Forward Function

- The forward function describes how input data (e.g., images, text, or tabu-lar data) flows through the various layers of the model. - It specifies the sequence of operations (e.g., convolution, pooling, activation, normalization, fully-connected lay-ers) that are applied to the input data. - Activation functions such as ReLU, Sigmoid, or Softmax are used to introduce non-linearity and enable the model to learn complex representations. - The output of the final layer(s) represents the model's predictions, which can be used for tasks like classification, regression, or generation.

1.3.10 Training and Validation Steps

Training Step

- During each training step, a batch of input data is passed through the model to compute the model's predictions. - The loss function (e.g., Cross-Entropy Loss, Mean

Squared Error) is used to calculate the difference between the model's predictions and the ground truth labels. - The optimizer (e.g., Adam, SGD) then updates the model's parameters using backpropagation to minimize the calculated loss. - Additional operations such as gradient clipping (to prevent exploding gradients) or L1/L2 regularization (to prevent overfitting) may be applied during the training step.

Validation Step

- Similar to the training step, a batch of validation data is fed through the model to compute predictions. - Validation loss and accuracy metrics are calculated to assess the model's performance on unseen data. - No parameter updates occur during the validation step, ensuring that the model's performance is evaluated independently of the training process. - The validation step helps monitor the model's generalization capability and identify potential overfitting or underfitting issues.

1.3.11 Optimizer and Loss Function

Optimizer Selection

- The choice of optimizer (e.g., Adam, SGD, RMSProp) determines how the model's parameters are updated during the training process. - Optimizers differ in their update rules, learning rate schedules, and hyperparameters such as learning rate, momentum, and weight decay. - The selected optimizer should be well-suited for the problem at hand and the model architecture to achieve efficient and stable convergence.

Loss Function

- The loss function quantifies the difference between the model's predictions and the actual ground truth labels. - For binary classification tasks like pneumonia detection, Binary Cross-Entropy Loss (BCEWithLogitsLoss) is commonly used.

- This loss function accepts raw logits as input and applies the sigmoid activation internally before computing the loss, which simplifies the model's output layer. Other loss functions like Mean Squared Error (for regression tasks) or Categorical Cross-Entropy (for multi-class classification) may be used depending on the problem domain.

1.3.12 Metrics Computation

Accuracy Metric

- Accuracy measures the proportion of correctly classified instances out of the total number of instances. - It provides a simple and intuitive metric for evaluating the model's overall performance. - Accuracy is calculated by comparing the model's predictions with the ground truth labels and dividing the number of correct predictions by the total number of predictions. - Other metrics like precision, recall, F1-score, or area under the ROC curve may also be computed to provide a more comprehensive evaluation of the model's performance, especially for imbalanced datasets.

1.3.13 Epoch-end Callbacks

Epoch-end Operations

- At the end of each epoch, additional computations and analysis can be per-formed to monitor the model's training and validation performance. - This includes calculating overall dataset statistics (e.g., mean, standard deviation, min, max) for input features and output labels. - Metrics such as training and validation loss, accuracy, and other domain-specific metrics can be aggregated across batches and logged for monitoring and visualization. Callback functions like training epoch end and validation epoch end are used to execute these epoch-end operations, allowing for custom logic and analysis.

1.3.14 Device Management and Optimizer Configuration

Device Management

- PyTorch Lightning automatically handles device management, allowing seamless training on CPUs, GPUs, or TPUs. - Developers can specify the target device(s) for training, and PyTorch Lightning will manage data transfer and

parallelization transparently. - This simplifies the process of scaling the training process to utilize available hardware resources, improving efficiency and reducing the need for manual device-specific programming.

Optimizer Configuration

- The configure optimizers method is used to specify the optimizer(s) to be used during the training process. - This method allows for the configuration of multiple optimizers and learning rate schedulers, which can be useful for advanced techniques like differential learning rates or layer-wise adaptive rates. - Optimizer hyperparameters such as learning rate, momentum, weight decay, and gradient clipping thresholds can be fine-tuned using this method to improve the training stability and convergence of the model.

```
class PneumoniaModel(pl.LightningModule):
    def __init__(self, weight=1):
        super().__init__()

        self.model = torchvision.models.resnet18()
        # change conv1 from 3 to 1 input channels
        self.model.conv1 = torch.nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
        # change out_feature of the last fully connected layer (called fc in resnet18) from 1000 to 1
        self.model.fc = torch.nn.Linear(in_features=512, out_features=1)

        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=1e-4)
        self.loss_fn = torch.nn.BCEWithLogitsLoss(pos_weight=torch.tensor([weight]))

        # simple accuracy computation
        self.train_acc = torchmetrics.Accuracy()
        self.val_acc = torchmetrics.Accuracy()

    def forward(self, data):
        pred = self.model(data)
        return pred

    def training_step(self, batch, batch_idx):
        x_ray, label = batch
        label = label.float()  # Convert label to float (just needed for loss computation)
        pred = self(x_ray)[:,0]  # Prediction: Make sure prediction and label have same shape
        loss = self.loss_fn(pred, label)  # Compute the loss

        # Log loss and batch accuracy
        self.log("Train Loss", loss)
        self.log("Step Train Acc", self.train_acc(torch.sigmoid(pred), label.int()))
        return loss
```

Figure 1.6: Model Building

MODEL TRAINING

1.4 Model Creation in PyTorch Lightning

The model training process involves training a classifier to predict whether a chest X-ray image shows signs of pneumonia or not. This is accomplished using the PyTorch Lightning framework, which simplifies the implementation of deep learning models.

A PyTorch Lightning module named Pneumonia Model is defined to encapsulate the model architecture and training logic. The model is based on the ResNet18 architecture with modifications to adapt to single-channel X-ray images and binary classification. The Adam optimizer and Binary Cross-Entropy Loss function BCE-WithLogitsLoss are used for model optimization and loss computation, respectively.

1.5 Checkpoint Callback

A checkpoint callback is created to save the best models based on validation accuracy during training. This ensures that the best-performing models are saved for later evaluation or deployment.

1.6 Model Training

The model is trained using the Trainer class from PyTorch Lightning. The trainer is configured with parameters such as the number of GPUs, logging frequency, and maximum number of epochs. Training is performed using the fit method, which iterates over the training and validation datasets, updating model parameters based on computed gradients.

1.7 Evaluation

After training, the model is evaluated on the validation dataset to assess its performance. The latest checkpoint is loaded, and the model is sent to the appropriate device (GPU or CPU). Predictions are computed for each image in the validation set, and metrics such as accuracy, precision, recall, and confusion matrix are calculated to evaluate model performance.

The detailed model training process described above outlines the steps in-volved in training a deep learning classifier for pneumonia detection using PyTorch Lightning. By following this process, developers can efficiently train and evaluate models for various image classification tasks.

```
def training_step(self, batch, batch_idx):
    x_ray, label = batch
    label = label.float()  # Convert label to float (just needed for loss computation)
    pred = self(x_ray)[:,0]  # Prediction: Make sure prediction and label have same shape
    loss = self.loss_fn(pred, label)  # Compute the loss

    # Log loss and batch accuracy
    self.log("Train Loss", loss)
    self.log("Step Train Acc", self.train_acc(torch.sigmoid(pred), label.int()))
    return loss


def training_epoch_end(self, outs):
    # After one epoch compute the whole train_data accuracy
    self.log("Train Acc", self.train_acc.compute())
```

Figure 1.7: Model Training

RESULT

1.8 System Output

1.8.1 Classification Result

The classification result indicates whether an input chest X-ray image exhibits signs of pneumonia or not. This binary classification outcome is generated by the trained model based on the features extracted from the input image. The result is typically presented as a probability score or a binary label (0 for no pneumonia, 1 for pneumonia).

1.8.2 Class Activation Maps (CAM)

Class Activation Maps (CAM) are generated to provide visual interpretations of the model's decision-making process. CAM highlights the regions in the chest X-ray images that contribute most to the classification outcome. By overlaying these heatmaps onto the original images, users can visually identify the areas that influenced the model's prediction.

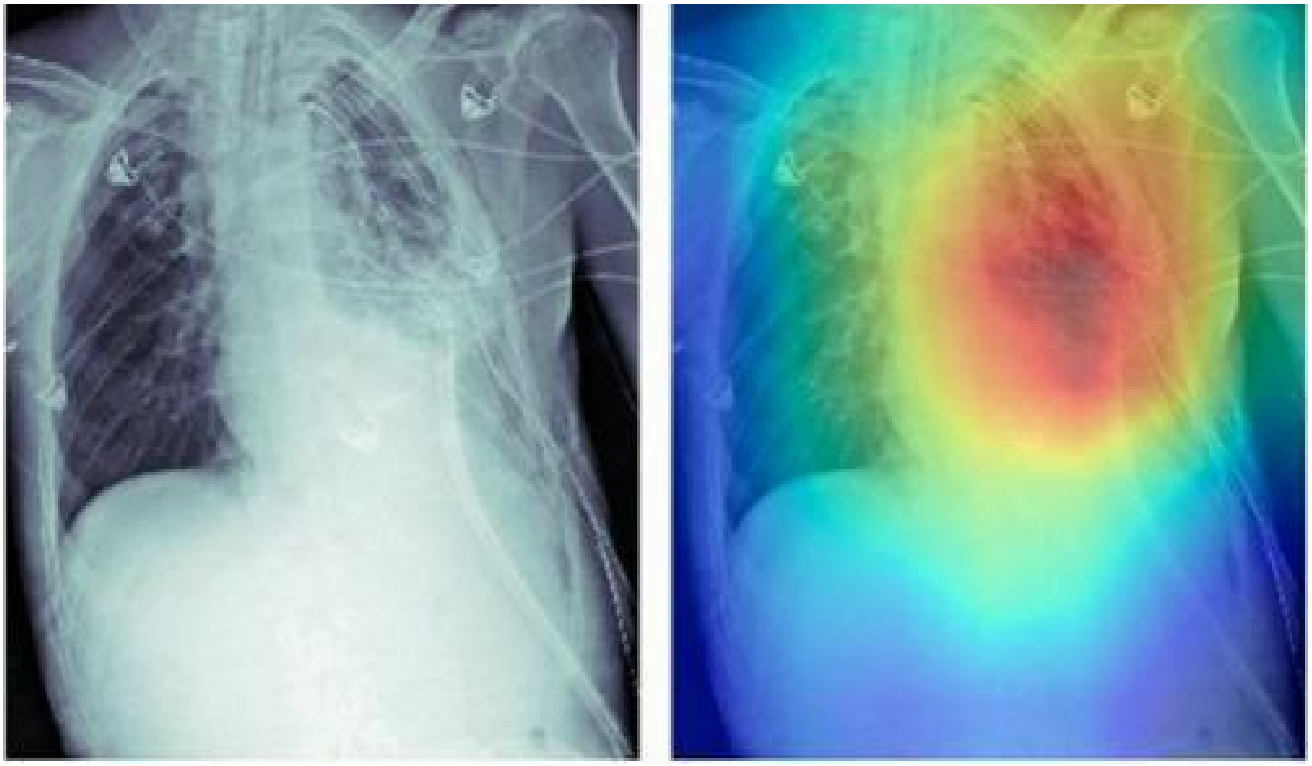


Figure 1.8: Result

1.9 Interpretability Result

The interpretability result enhances the understanding of the model's classifi-cation decisions by visually highlighting the regions of importance within the chest X-ray images. CAM allows clinicians and researchers to identify anatomical areas or pathological patterns that the model deems significant for pneumonia detection. This interpretability aspect not only helps validate the model's predictions but also provides insights into the underlying features learned by the neural network.

The output of the system includes both a classification result indicating pneu-monia presence and Class Activation Maps (CAM) for interpretability, enabling users to understand and interpret the model's decisions effectively.

INTERPRETABILITY

1.10 Interpretability Topic: Class Activation Maps (CAM)

Class Activation Maps (CAM) are a powerful technique used to visualize the decision-making process of convolutional neural networks (CNNs) by highlighting the regions of input images that contribute most significantly to the prediction of a specific class. CAM provides interpretability and insight into how CNNs make predictions, especially in image classification tasks. Here's a detailed explanation of how CAM works and its significance:

1.10.1 Conceptual Overview

- CAM operates on the principle that different regions of an image contribute differently to the final prediction made by a CNN.
- By examining which parts of an image are most relevant to the CNN's decision, CAM helps understand the model's reasoning process and identify key features used for classification.

1.10.2 Mathematical Basis

- CAM involves extracting the class-specific activation maps from the final con-volutional layer of a CNN.
- These activation maps capture the response of various features in the input im-age.
- CAM computes the class activation weights, representing the importance of each feature map for predicting a specific class.
- The CAM for a target class is obtained by weighting and combining the activa-tion maps based on the class activation weights.
- Mathematically, CAM is computed as the sum of the element-wise product of class activation weights and activation maps.

1.10.3 Implementation Steps

- Pass the input image through the CNN until the final convolutional layer, extracting the feature maps.
- Compute the class activation weights using the weights of the fully connected layer corresponding to the target class.
- Multiply each activation map by its corresponding class activation weight and sum the results to obtain the CAM.
- Normalize and standardize the CAM to highlight relevant regions of the input image.

1.10.4 Visualization

- CAM overlays the computed heatmap on the original input image, visually in-dicating the regions contributing most to the prediction.
- The heatmap intensity represents the importance of each pixel in influencing the model's decision.
- By visualizing CAM, one can identify which areas of the image are crucial for the CNN's classification, aiding in model interpretation and validation.

1.10.5 Interpretability Benefits

- CAM enhances the interpretability of CNNs by providing intuitive visual ex-planations of model predictions.
- It helps identify the discriminative features learned by the CNN, facilitating model debugging, validation, and improvement.
- Researchers and practitioners use CAM to gain insights into CNN behavior, verify model correctness, and build trust in AI systems, especially in critical applications like medical diagnosis.

1.10.6 Limitations and Considerations

- CAM requires specific CNN architectures with a global average pooling layer followed by a single fully connected layer.
- For other architectures, alternative methods like GradCAM or ScoreCAM may be more suitable.
- Interpretation of CAM results should be done cautiously, considering potential biases, limitations of the dataset, and the context of the application.

Class Activation Maps offer a valuable tool for understanding and interpreting the decisions made by CNNs in image classification tasks. By visualizing the regions of importance in input images, CAM promotes transparency, accountability, and trust in AI systems, making them more interpretable and explainable.

REQUIREMENTS

1.11 Requirements

1.11.1 Chest X-ray Dataset with Labeled Images

- A dataset containing a sufficient number of chest X-ray images with corre-sponding labels indicating whether each image exhibits signs of pneumonia or not.
- The dataset should be properly labeled and adequately preprocessed for use in training and validation.

1.11.2 PyTorch and PyTorch Lightning Frameworks

- Installation of PyTorch and PyTorch Lightning frameworks to develop and train deep learning models efficiently.
- Ensure that the frameworks are properly configured and updated to the latest stable versions for compatibility with other components.

1.11.3 torchvision Library for Model Creation and Data Handling

- Integration of the torchvision library, a part of the PyTorch ecosystem, to facil-itate model creation, data loading, and preprocessing.
- Utilization of torchvision's pre-built models, transforms, and datasets for stream-lined development and experimentation.

1.11.4 Necessary Hardware Resources for Model Training

- Availability of suitable hardware resources for model training, such as GPUs (Graphics Processing Units) or TPUs (Tensor Processing Units), to expedite the training process.

- Ensure compatibility and proper configuration of hardware resources with the PyTorch framework for optimal performance during training.

1.11.5 Class Activation Maps (CAM) Implementation for Interpretability

- Implementation of Class Activation Maps (CAM) technique to provide interpretability and insights into the model's decision-making process.
- Integration of CAM into the system architecture to generate visualizations high-lighting the regions of importance in chest X-ray images for pneumonia detec-tion.
- Ensure CAM implementation is compatible with the chosen deep learning framework and seamlessly integrates with the model training and evaluation pipeline.

1.12 PyTorch

PyTorch is an open-source machine learning framework developed by Face-book's AI Research lab (FAIR). It provides a flexible and dynamic approach to build-ing deep learning models, emphasizing ease of use, flexibility, and scalability. Here's a detailed definition of PyTorch:

1.12.1 Tensor Computation Library

- PyTorch serves as a powerful tensor computation library, enabling efficient ma-nipulation and computation of multi-dimensional arrays (tensors).
- It offers extensive support for tensor operations, including arithmetic opera-tions, linear algebra, indexing, slicing, and broadcasting.

1.12.2 Dynamic Computational Graphs

- One of the key features of PyTorch is its dynamic computational graph con-struction.

- Unlike static graph frameworks like TensorFlow, PyTorch constructs the com-putational graph dynamically during runtime, allowing for more flexible and intuitive model development.
- This dynamic nature enables easier debugging, experimentation, and model customization.

1.12.3 Automatic Differentiation

- PyTorch provides automatic differentiation capabilities through its autograd module.
- It automatically computes gradients of tensors with respect to a given loss func-tion, facilitating gradient-based optimization algorithms such as backpropagation.
- This feature simplifies the implementation of complex neural network architec-tures and training procedures.

1.12.4 Modular Design

- PyTorch adopts a modular design philosophy, providing a collection of modular and extensible components.
- It offers various modules and classes for building different components of deep learning models, including layers, activations, loss functions, optimizers, and more.
- This modular approach promotes code reusability, maintainability, and scalability.

1.12.5 Support for GPU Acceleration

- PyTorch seamlessly integrates with NVIDIA CUDA for GPU acceleration, al-lowing for high-speed computation on compatible GPUs.
- It provides GPU support for both tensor operations and model training, signifi-cantly reducing training times for large-scale deep learning models.

1.12.6 Rich Ecosystem and Community Support

- PyTorch boasts a vibrant ecosystem and active community of developers, re-searchers, and practitioners.
- It offers extensive documentation, tutorials, and resources to support users at all skill levels.
- Additionally, PyTorch benefits from contributions and advancements in the broader deep learning and machine learning community.

1.12.7 Deployment and Production

- PyTorch offers various tools and libraries for deploying and serving trained models in production environments.
- It supports model deployment on various platforms, including cloud services, mobile devices, and embedded systems, through frameworks like TorchServe and TorchScript.

Overall, PyTorch provides a flexible and user-friendly platform for building, training, and deploying deep learning models, making it a popular choice among researchers, practitioners, and enthusiasts in the machine learning community.

RESNET-SKIP CONNECTIONS

1.13 ResNet (Residual Neural Network)

ResNet, short for Residual Neural Network, is a deep learning architecture that revolutionized the field of computer vision when it was introduced by Kaiming He et al. in their seminal paper ”Deep Residual Learning for Image Recognition” in 2015. ResNet is renowned for its effectiveness in training very deep neural networks, mitigating the vanishing gradient problem, and achieving state-of-the-art performance on various image classification tasks.

1.13.1 Key Components of ResNet

Skip Connections (Residual Blocks)

- The hallmark of ResNet architecture is the use of skip connections or residual connections.
- Skip connections enable the network to learn residual functions, which repre-sent the difference between the input and output of a particular layer.
- Instead of directly mapping inputs to outputs, ResNet learns residual mappings, making it easier for the network to learn identity mappings (where the input and output have similar features).
- This alleviates the vanishing gradient problem and enables the training of very deep networks by allowing gradients to flow more easily through the network.
- Mathematically, the output of a residual block H(x) is computed as H(x) = F(x) + x, where F(x) is the residual mapping learned by the block.

Layer Stacking

- ResNet consists of multiple residual blocks stacked on top of each other, form-ing a deep neural network.

- Each residual block typically contains several convolutional layers followed by batch normalization and activation functions like ReLU.
- By stacking these residual blocks, ResNet can learn increasingly complex fea-tures and hierarchies of representations.

Global Average Pooling and Fully Connected Layer

- Unlike traditional architectures, ResNet replaces the fully connected layers at the end of the network with global average pooling (GAP) layers.
- GAP reduces the spatial dimensions of the feature maps to a single value per channel by computing the average of each channel's values.
- This reduces the number of parameters in the model and provides spatial infor-mation to the subsequent fully connected layer for classification.

1.13.2 ResNet Architecture and Mathematical Explanation

ResNet (Residual Neural Network) introduces the concept of skip connections or residual connections, which revolutionized the training of deep neural networks. The core idea behind ResNet is to learn residual mappings instead of directly mapping input to output. This is achieved by adding skip connections that enable the network to learn the residual functions, making it easier to train very deep networks.

Mathematical Architecture

Residual Block

- The basic building block of ResNet is the residual block, which consists of two main paths: the identity path and the residual path.
- Let x denote the input to the residual block, and H(x) denote the output of the block.
- The residual block computes the output H(x) as the sum of the identity map-ping x and the residual mapping F(x).
- Mathematically, $H(x) = F(x) + x$.

Identity Path

- The identity path represents the direct mapping from input to output without any transformation.
- It preserves the input features and serves as a baseline for comparison with the output of the residual path.

Residual Path

- The residual path consists of one or more convolutional layers followed by batch normalization and activation functions.
- It learns the residual mapping F(x), which represents the difference between the input and output of the block.

Skip Connection

- The skip connection, also known as the shortcut connection, directly connects the input x to the output H(x) of the residual block.
- It provides an alternative path for the gradient flow during backpropagation, mitigating the vanishing gradient problem.
- The skip connection enables the network to learn residual mappings efficiently by focusing on learning the difference between input and output.

Stacking Residual Blocks

- ResNet consists of multiple residual blocks stacked on top of each other to form a deep neural network.
- Each residual block learns increasingly complex features, allowing the network to capture hierarchical representations from the input data.

Final Layers

- The final layers of ResNet typically include global average pooling (GAP) fol-lowed by fully connected layers for classification.
- GAP reduces the spatial dimensions of the feature maps to a single value per channel, followed by softmax activation for multi-class classification or sig-moid activation for binary classification.

Mathematical Explanation

- Let x denote the input feature map of a residual block.
- The output H(x) of the block is computed as H(x) = F(x) + x, where F(x) represents the residual mapping learned by the block.
- The network learns to optimize the parameters of F(x) such that the difference between H(x) and x is minimized, effectively learning the residual function.
- By focusing on learning residual functions, ResNet facilitates the training of very deep networks by enabling the direct propagation of gradients through skip connections.

ResNet's mathematical architecture is characterized by the incorporation of skip connections and residual blocks, which allow the network to efficiently learn residual mappings and alleviate the challenges associated with training deep neural networks. These architectural innovations have contributed to the widespread adop-tion and success of ResNet in various computer vision tasks.

1.14 Applications of ResNet

ResNet, with its innovative skip connections and residual learning, has found widespread applications in various domains of computer vision and deep learning. Some of the key applications of ResNet include:

1.14.1 Image Classification

ResNet has demonstrated state-of-the-art performance on numerous image clas-sification benchmarks, such as ImageNet, CIFAR, and MS-COCO. Its ability to train very deep networks effectively has led to significant improvements in image recognition tasks.

1.14.2 Object Detection

ResNet's feature extraction capabilities have been leveraged in object detec-tion frameworks, such as Faster R-CNN and YOLO, to achieve high-accuracy object localization and classification.

1.14.3 Semantic Segmentation

ResNet's deep, hierarchical features have been successfully incorporated into fully convolutional networks (FCNs) for pixel-wise semantic segmentation tasks, where the goal is to assign a class label to each pixel in an image.

1.14.4 Transfer Learning

The pre-trained weights of ResNet models, trained on large-scale datasets like ImageNet, can be effectively transferred to other domains and tasks, enabling efficient training and high-performance models with limited data.

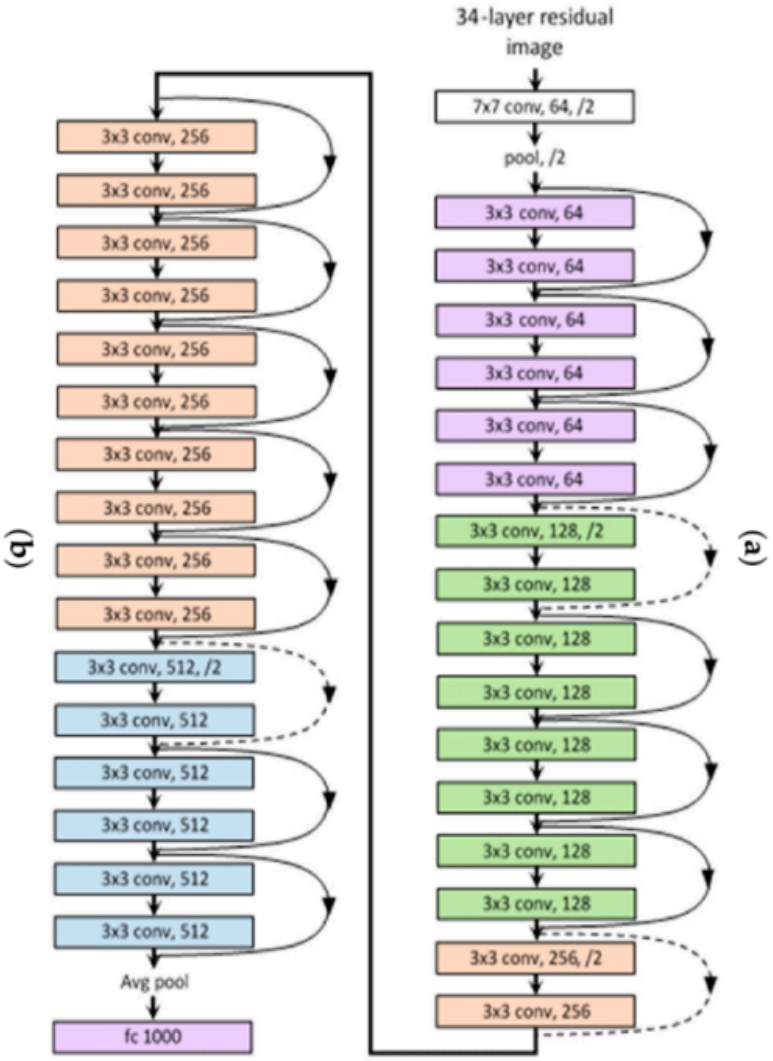


Figure 1.9: Resnet Architecture

1.15 Role of ResNet in Pneumonia Detection

ResNet plays a crucial role in the detection of pneumonia from chest X-ray images. Its unique architectural design and residual learning approach make it an effective choice for this medical imaging task.

1.15.1 Chest X-ray Image Classification

In the context of pneumonia detection, ResNet is employed to classify chest X-ray images into two classes: pneumonia and non-pneumonia. The deep, hierarchical features learned by the ResNet model are well-suited for capturing the complex visual patterns associated with pneumonia in chest X-ray images.

1.15.2 Residual Learning for Pneumonia Detection

The residual learning approach of ResNet is particularly advantageous for pneumonia detection. By learning residual mappings, the network can effectively model the differences between healthy and pneumonia-affected lung regions, which are often subtle and require a deep understanding of the underlying visual patterns.

1.15.3 Handling Vanishing Gradients

Pneumonia detection from chest X-rays often involves complex visual fea-tures and relationships, requiring very deep neural networks to capture these nuances. ResNet's skip connections and residual blocks mitigate the vanishing gradient prob-lem, enabling the training of sufficiently deep models for this task.

1.15.4 Interpretability with Class Activation Maps

ResNet's architecture, combined with techniques like Class Activation Maps (CAM), can provide valuable interpretability for pneumonia detection. By visualizing the regions of the chest X-ray image that contribute most to the model's prediction, clinicians can gain insights into the decision-making process and validate the model's performance

1.15.5 Transfer Learning for Pneumonia Detection

Pre-trained ResNet models, trained on large-scale image datasets, can be ef-fectively fine-tuned for the specific task of pneumonia detection from chest X-rays. This transfer learning approach leverages the generic visual feature representations learned by ResNet, significantly reducing the required training data and computa-tional resources.

ResNet's innovative architecture, residual learning, and interpretability fea-tures make it a highly effective and widely adopted choice for the task of pneumonia detection from chest X-ray images. Its ability to train deep models, handle complex visual patterns, and provide interpretable results has contributed to its success in this medical imaging domain.

CLASS ACTIVATION MAP

1.16 Class Activation Map (CAM) - *Detailed Explanation*

Class Activation Map (CAM) is a visualization technique used to understand the areas of an image that contribute the most to a particular class prediction made by a convolutional neural network (CNN). CAM provides insights into the network's decision-making process by highlighting the regions of the input image that are most relevant to a specific class.

1.16.1 Purpose and Implementation

Purpose

- CAM is implemented to enhance the interpretability of CNNs, particularly in image classification tasks.
- It helps in understanding where the model is focusing its attention within an image to make a classification decision.
- CAM provides valuable insights into the discriminative regions of the input image, aiding in model debugging, validation, and trustworthiness assessment.

Implementation

- CAM is implemented by extracting the feature maps from the last convolutional layer of a CNN and then applying global average pooling (GAP) to obtain a feature vector for each channel.
- These feature vectors are then multiplied by the corresponding weights of the fully connected layer responsible for the target class.
- The resulting weighted feature maps are summed to produce the CAM for the target class.
- Finally, the CAM is upsampled to the original image size to visualize the class-specific activation regions.

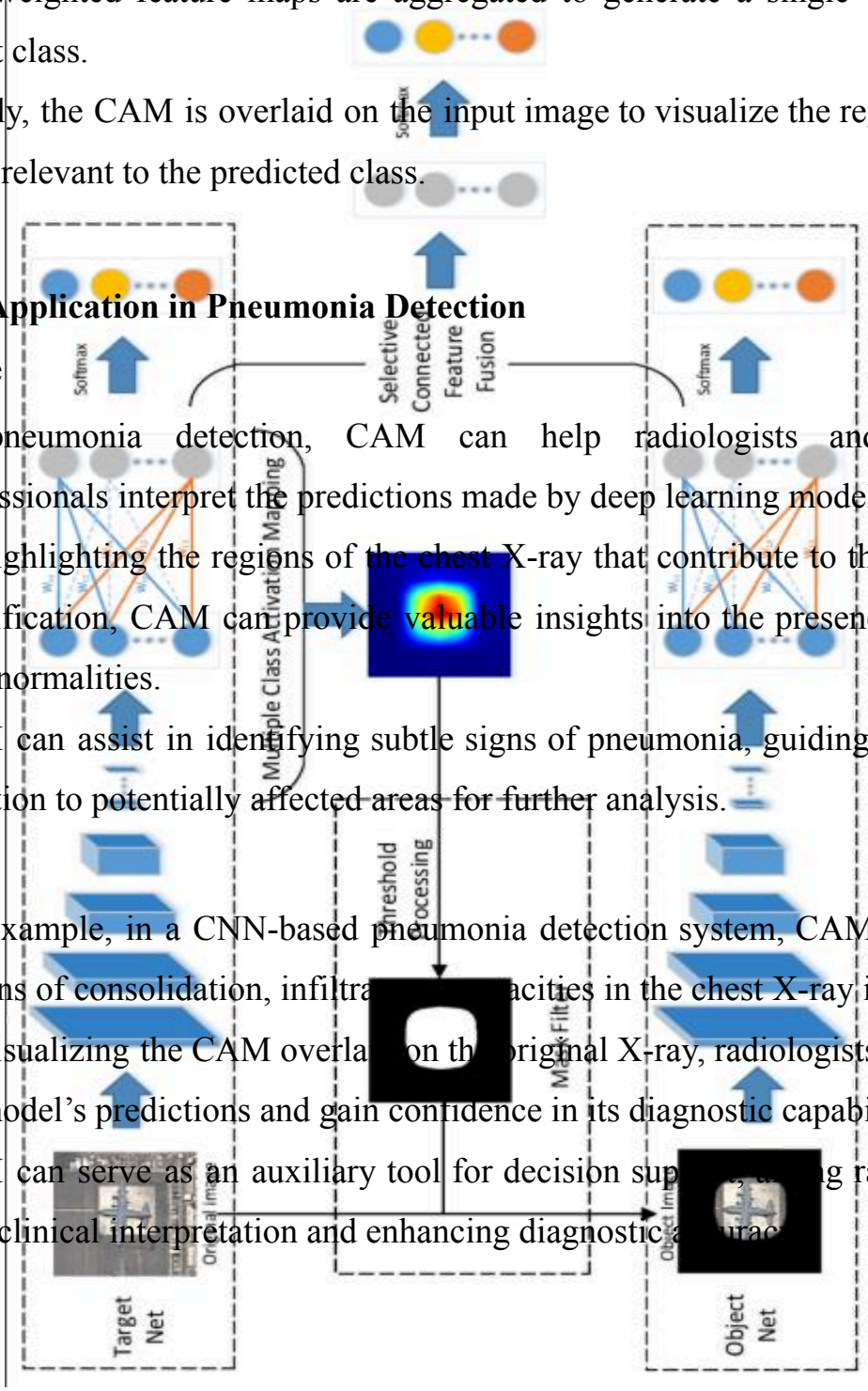
1.16.2 Logic and Approach

Logic

- CAM leverages the inherent hierarchical feature learning capability of CNNs.
- It exploits the fact that deeper convolutional layers capture increasing abstract and discriminative features.
- By examining the activations of these layers, CAM can localize the areas of the input image that contribute the most to the prediction of a particular class.

Approach

- CAM starts by extracting feature maps from the last convolutional layer of a CNN, which captures high-level semantic information.
- It then computes the importance of each feature map by weighting it with the corresponding class-specific weights from the fully connected layer.
- The weighted feature maps are aggregated to generate a single CAM for the target class.
- Finally, the CAM is overlaid on the input image to visualize the regions that are most relevant to the predicted class.



1.16.3 Application in Pneumonia Detection

Relevance

- In pneumonia detection, CAM can help radiologists and healthcare professionals interpret the predictions made by deep learning models.
- By highlighting the regions of the chest X-ray that contribute to the pneumonia classification, CAM can provide valuable insights into the presence and extent of abnormalities.
- CAM can assist in identifying subtle signs of pneumonia, guiding radiologists' attention to potentially affected areas for further analysis.

Use Case

- For example, in a CNN-based pneumonia detection system, CAM can identify regions of consolidation, infiltra acities in the chest X-ray images.
- By visualizing the CAM overla on th riginal X-ray, radiologists can validate the model's predictions and gain confidence in its diagnostic capabilities.
- CAM can serve as an auxiliary tool for decision sup g radiologists in their clinical interpretation and enhancing diagnostic a urac

Figure 1.10: Class Activation Map

1.17 Mathematical Concept of Class Activation Maps (CAM)

The mathematical concept behind Class Activation Maps (CAM) involves ex-tracting the class-specific activation maps from the final convolutional layer of a con-volutional neural network (CNN) and combining them to visualize the regions of an input image that contribute most to the prediction of a particular class.

1.17.1 Feature Extraction

- CAM starts by passing an input image through the CNN until the final convo-lutional layer. This layer typically captures high-level semantic features of the input image.

1.17.2 Class Activation Weights

- For a specific target class c, CAM computes the class activation weights w_c. These weights represent the importance of each feature map in the final convo-lutional layer for predicting class c.

1.17.3 Activation Map Extraction

- CAM extracts the activation maps A^L from the final convolutional layer. Each activation map represents the response of a particular feature to the input image.

1.17.4 CAM Calculation

- To compute the CAM for class c, CAM multiplies each activation map A^L_k by its corresponding class activation weight $w_{c,k}$ and sums the weighted activation maps:

$$CAM_c = \quad w_{c,k} \times A^L_k$$

- Here, k represents the index of each feature map in the final convolutional layer.

1.17.5 Visualization

- The resulting CAM for class c highlights the regions of the input image that are most relevant to the prediction of class c.
- CAM can be overlaid on the original input image to visualize the areas that contribute most significantly to the predicted class.

Class Activation Map (CAM) is a powerful technique for visualizing and inter-preting the decisions of convolutional neural networks in image classification tasks. By highlighting the discriminative regions of input images, CAM enhances the

trans-parency and interpretability of deep learning models, making them more accessi-ble and trustworthy for real-world applications like pneumonia detection in medical imaging.

CHAPTER 2: CONCLUSION

In conclusion, the integration of Class Activation Maps (CAM) into the interpretability framework of convolutional neural networks (CNNs) provides a valuable tool for understanding and validating the decision-making process of these models in image classification tasks. By highlighting the regions of input images that contribute most significantly to the classification outcome, CAM enhances the interpretability and transparency of CNNs, aiding in model validation, debugging, and improvement.

CAM's mathematical foundation, implementation steps, and visualization tech-niques offer intuitive insights into the features learned by CNNs, facilitating model interpretation and building trust in AI systems. However, it's essential to recognize CAM's limitations, such as its dependency on specific CNN architectures and the need for cautious interpretation of results in real-world applications.

Overall, CAM represents a significant advancement in AI interpretability, em-powering researchers and practitioners to gain deeper insights into CNN behavior and ensure the reliability and robustness of AI systems across various domains, including healthcare, autonomous driving, and more.

2.1 FUTURE WORK

Future work in the realm of interpretability and model validation could ex-plore several avenues to enhance the effectiveness and applicability of Class Activa-tion Maps (CAM) and similar techniques. Firstly, researchers could investigate the development of CAM-compatible architectures for different types of neural networks beyond the standard ResNet-like structures, enabling broader applicability across var-ious network architectures. Additionally, efforts could focus on refining CAM visu-alization methods to provide more detailed and precise insights into the

decision-making process of complex models, potentially incorporating multi-scale analysis or attention mechanisms to capture finer-grained features.

Class Activation Mapping (CAM) holds promise for enhancing our under-standing of deep learning models. There's potential to combine CAM with other interpretation techniques to gain richer insights. Furthermore, researchers should de-velop CAM-inspired methods for analyzing models that don't deal with images, like text or time series data. Investigating how CAM can be applied in real-world domains like healthcare is crucial to assess its practical value. Ultimately, future advancements in CAM should aim for increased capabilities, robustness, and accessibility, fostering the development of trustworthy and transparent AI systems.

APPENDIX A: Supplementary Plots

A.1 Supplementary Plots

A.1.1 Data Distribution Plots

Histograms or violin plots depicting the distribution of features in the dataset, such as age, gender, or any other relevant demographic information. These plots can help understand the characteristics of the dataset and potential biases.

A.1.2 Model Performance Across Different Metrics

Line plots or bar charts showing the performance of the developed model across different evaluation metrics, such as accuracy, precision, recall, F1-score, and area under the curve (AUC). These plots can provide a comprehensive overview of the model's performance beyond a single metric.

A.1.3 Learning Curves

Plots showing the training and validation loss or accuracy over epochs during the model training process. Learning curves can help visualize the model's conver-gence and identify issues such as overfitting or underfitting.

A.1.4 Confusion Matrices

Heatmaps illustrating the confusion matrices for the classification model, show-ing the distribution of true positive, true negative, false positive, and false negative predictions. These plots provide insights into the model's performance across differ-ent classes and can help identify any specific class-related challenges.

A.1.5 Precision-Recall Curves

Curves illustrating the trade-off between precision and recall for different clas-sification thresholds. These plots are particularly useful when dealing with imbal-anced datasets and can help evaluate the model's performance at varying decision thresholds.

A.1.6 Feature Importance Plots

If applicable, plots depicting the importance or contribution of different fea-tures in the classification task. These plots could be generated using techniques such as

permutation importance, SHAP values, or feature importance scores from tree-based models.

A.1.7 Model Interpretability Plots

Visualizations such as Class Activation Maps (CAM), Grad-CAM, or Inte-grated Gradients showing the regions of the input images that are most relevant for the model's predictions. These plots can enhance the interpretability of the model's decision-making process.

A.1.8 Error Analysis Plots

Plots showing examples of misclassified instances or instances with high prediction uncertainty. These plots can help identify common patterns or challenges faced by the model and guide further improvements.

A.1.9 Model Comparison Plots

Plots comparing the performance of different models or model variants (e.g., baseline vs. optimized model). These plots can help demonstrate the effectiveness of proposed enhancements or modifications.

REFERENCES

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for im-age recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[2] Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convo-lutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR.

[3] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Langlotz, C. (2017). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225.

[4] Wang, S., Kang, B., Ma, J., Zeng, X., Xiao, M., Guo, J., ... & Cai, M. (2020). A deep learning algorithm using CT images to screen for Corona Virus Disease (COVID-19). medRxiv.

[5] Liu, L., Luo, T., Nie, S., Yin, X., Zhou, Z., & Hou, L. (2021). Deep learning for pulmonary disease detection in chest x-rays: A systematic review and meta-analysis. Journal of Medical Internet Research, 23(2), e23498.

[6] Cires¸an, D., Giusti, A., Gambardella, L. M., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In 2012 IEEE conference on computer vision and pattern recognition (pp. 3642-3649). IEEE.

[7] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[8] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[9] Li, L., Qin, L., Xu, Z., Yin, Y., Wang, X., Kong, B., ... & Xu, X. (2020). Us-ing artificial intelligence to detect COVID-19 and community-acquired pneumo-nia based on pulmonary CT: Evaluation of the diagnostic accuracy. Radiology,296(2), E65-E71.

[10] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In Pro-ceedings of the IEEE conference on computer vision and pattern recognition (pp. 2097-2106).

[11] Madani, A., Arnaout, R., Mofrad, M. R., & Arnaout, R. (2020). Fast and ac-curate view classification of echocardiograms using deep learning. npj Digital Medicine, 3(1), 1-8.

[12] Rajpurkar, P., Park, A., Irvin, J., Zhu, K., Yang, B., Mehta, H., ... & Ng, A. Y. (2018). Deep learning for chest radiograph diagnosis: A retrospective com-parison of the CheXNeXt algorithm to practicing radiologists. PLoS Medicine, 15(11), e1002686.

[13] Mooney, P., Zhao, Q., & Pederson, T. (2016). Deep learning for healthcare ap-plications based on physiological signals: A review. Computer Methods and Programs in Biomedicine, 138, 55-63.

[14] Shan, F., Gao, Y., Wang, J., Shi, W., Shi, N., Han, M., ... & Li, Y. (2020). Lung infection quantification of COVID-19 in CT images with deep learning. arXiv preprint arXiv:2003.04655.

[15] Ardakani, A. A., Kanafi, A. R., Acharya, U. R., Khadem, N., & Mohammadi, A. (2020). Application of deep learning technique to manage COVID-19 in routine clinical practice using CT images: Results of 10 convolutional neural networks. Computers in Biology and Medicine, 121, 103795.