Introduction to Deep Learning

Dr. David Ruby

1 December 2024

Pneumonia detection using CNN

1. Overview

"Prevention is better than cure" underscores the critical importance of early diagnosis in healthcare. Among various health conditions, pneumonia—a potentially life-threatening infection that inflames the air sacs in the lungs—remains a significant cause of morbidity and mortality worldwide. Early and accurate detection is paramount to improving treatment outcomes and reducing complications.

Chest X-rays are a standard diagnostic tool for pneumonia, but their interpretation is often subject to human error, variability, and resource constraints in overburdened healthcare systems. With advancements in deep learning, automated image analysis offers an efficient and consistent approach to pneumonia detection, minimizing the likelihood of misdiagnosis.

This project leverages **Convolutional Neural Networks (CNNs)**, a powerful deep learning architecture, to classify chest X-ray images into NORMAL and PNEUMONIA categories. By integrating preprocessing, feature extraction, and supervised learning techniques, the model aims to achieve high accuracy and reliability in diagnosing pneumonia from medical images.

The report is structured into the following sections:

- Introduction to automated pneumonia detection.
- Dataset and exploratory data analysis.
- Implementation of the Convolutional Neural Network.
- Results and conclusions.

2. Introduction to Pneumonia Detection

Pneumonia detection through CNN-based deep learning automates the analysis of chest X-ray images, offering an accurate, efficient, and scalable solution for diagnostics. This project addresses the challenges associated with manual interpretation, such as variability and error, by implementing a robust CNN model tailored for medical imaging.

Key goals of this project include:

1. **Data Preprocessing**: Resizing, normalizing,grayscale and augmenting input X-ray images to ensure consistent quality and improved model performance.
2. **Feature Extraction**: Detecting crucial patterns such as opacity, irregular textures, and asymmetries in lung regions associated with pneumonia.
3. **Image Classification**: Accurately differentiating between healthy lungs and pneumonia-affected lungs using supervised learning.

Feature of the Implementation:

- **Data Augmentation**: Employing techniques like rotations, flips, and zooms to artificially expand the dataset, increasing its diversity and reducing the risk of overfitting.

The model's performance is evaluated using metrics such as accuracy, precision, recall, and loss curves. Visualizations of classification processes and feature maps provide transparency into the CNN's decision-making process. This project demonstrates the potential of CNNs to support healthcare professionals by improving diagnostic reliability and accessibility, contributing to better patient outcomes.

3.The dataset, Exploratory Data Analysis and Pre-Processing

The dataset used for pneumonia detection is the Chest X-ray Pneumonia Dataset. It contains X-ray images categorized into two classes:

1. **NORMAL: Healthy chest X-ray images.**

2. **PNEUMONIA: Chest X-ray images of patients with pneumonia.**

The dataset is structured into three main directories:

- **Train:** Used for training the model.
- **Test:** Used for evaluating model performance.
- **Validation:** Used to tune the model and prevent overfitting.

Each directory contains subfolders for NORMAL and PNEUMONIA images. The dataset has a class imbalance, as pneumonia images significantly outnumber normal images.

Steps for Pre-Processing

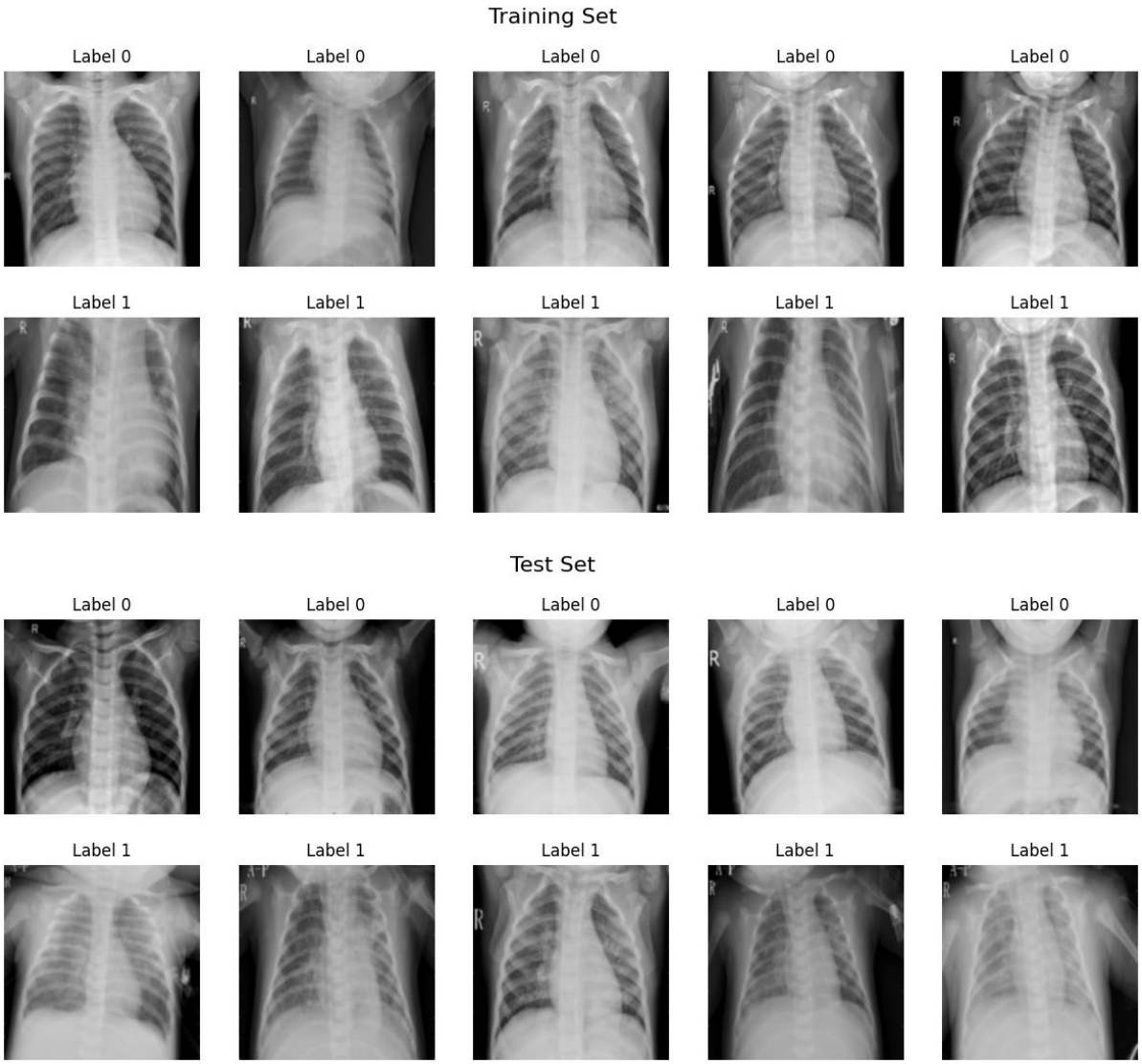
1. **Data Cleaning:**
    - **Size Reduction**: Resize all images to 256 pixels (either width or height, maintaining the aspect ratio).
    - **Normalization**: Scale pixel values to a range (e.g., 0–1 or -1 to 1) to standardize the data.
    - **Gaussian Blur**: Apply a Gaussian blur to smooth the images, and remove noise

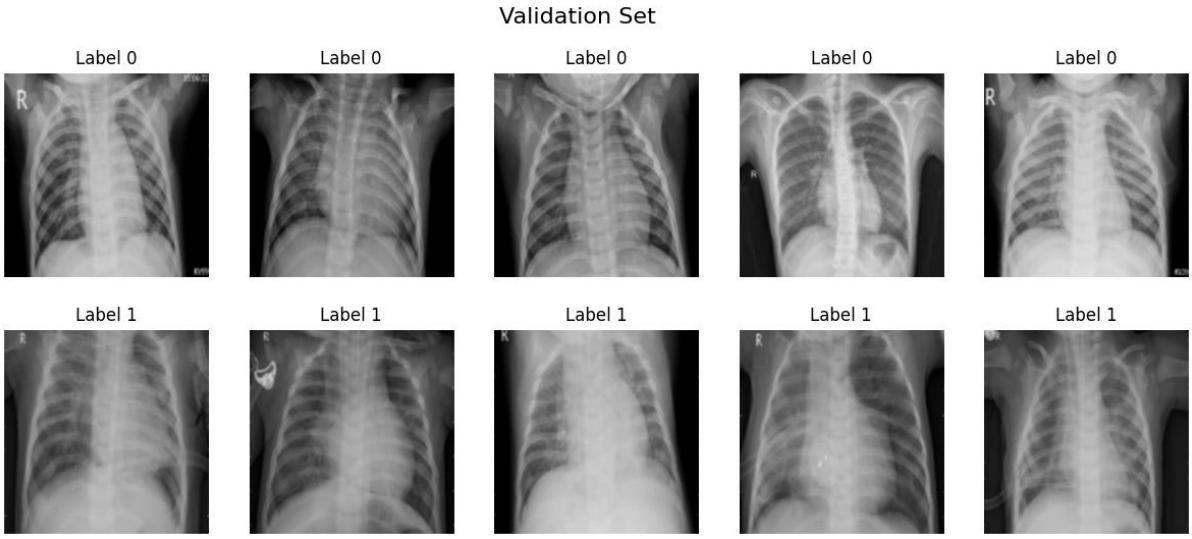
2. **Exploration Functions:**
    - Count and analyze the number of images in each class and set (Train, Test, Validation).

- Visualize random samples from both classes to ensure data quality and understand class characteristics.

3. **Class Balancing:**
   - Identify class imbalances using statistical functions or histograms.
   - Implement class weights during model training to balance the learning process**.**
4. **Image Pre-Processing:**
   - Resizing: Resize all images to a fixed dimension (e.g., 128x128 pixels).
   - Normalization: Scale pixel values to the range [0, 1] for faster and more stable training.
   - Image Labeling: Assign appropriate labels to each image based on its category or class to prepare the dataset for supervised learning.

Process images:

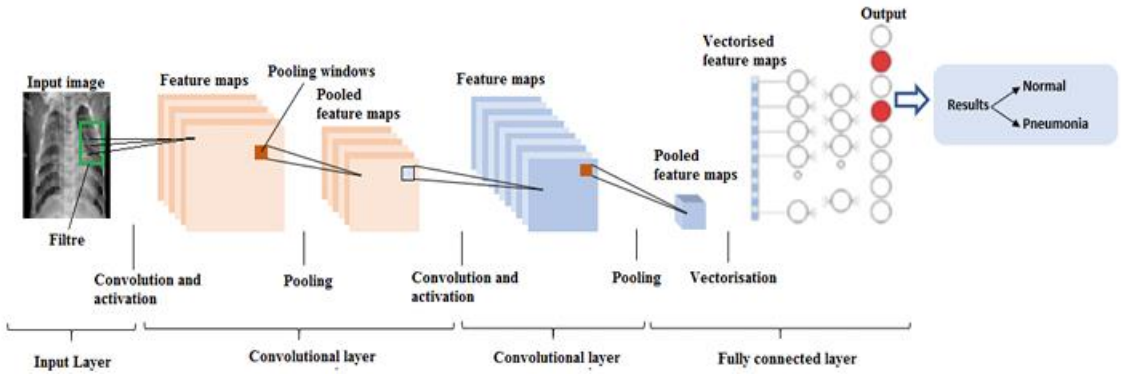




5. **Data Augmentation:**
    - Use real-time data augmentation techniques like rotation, zoom, and flipping to generate diverse samples and avoid overfitting.

4. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a powerful deep learning technique that excels in image-based tasks, making them particularly suitable for detecting pneumonia in chest X-rays. Unlike traditional Artificial Neural Networks (ANNs), which require massive numbers of parameters for image processing, CNNs use convolutional layers with small kernels to extract spatial features such as edges, textures, and patterns. This hierarchical feature extraction mirrors human visual perception, where basic structures are identified in early layers, and more complex patterns, like the opacities and irregularities in pneumonia-affected lungs, are captured in deeper layers. In this project, CNNs analyze chest X-ray datasets categorized as "NORMAL" and "PNEUMONIA." To address the imbalance in class distributions, class weights are applied during training, ensuring that the model does not favor the majority class. Preprocessing steps, including resizing images to 128x128 pixels, normalization, and data augmentation, improve computational efficiency and enhance model robustness by simulating real-world variability.



The CNN model architecture consists of three convolutional layers with increasing filter sizes (32, 64, and 128), followed by max-pooling layers to downsample feature maps and dropout layers to mitigate overfitting. The fully connected layers at the end of the network perform

binary classification, distinguishing between normal and pneumonia cases. The model uses callbacks like learning rate reduction and early stopping to optimize the training process, ensuring a balance between performance and computational cost. Upon evaluation, the model demonstrated high accuracy on the test set, highlighting its effectiveness in detecting pneumonia from chest X-rays. The training process also showcased consistent learning curves, with both training and validation accuracy steadily improving while validation loss stabilized, indicating minimal overfitting. By leveraging CNNs, this project provides a reliable and efficient tool for automating pneumonia diagnosis, reducing the burden on radiologists and enabling faster healthcare delivery.

Pytroch

PyTorch is used as the primary deep-learning framework for model building and optimization. It provides efficient tensor operations through the torch library, enabling seamless manipulation of multi-dimensional arrays (tensors) used for image data and neural network computations. The torch.nn module defines neural network layers, while torch.optim facilitates optimization using algorithms like Adam. These features streamline the development, training, and evaluation of the Convolutional Neural Network (CNN) for pneumonia detection.

Building the model

Step 1: Prepare Dataset for Training

The first step in building the model is preparing the dataset. The chest X-ray dataset, organized into "NORMAL" and "PNEUMONIA" categories, is loaded and divided into training, testing, and validation sets. Image paths are extracted using Python's glob library, ensuring the correct folder structure is followed. The dataset statistics, including the number of images in each category, are computed and printed for verification. Data visualization is used to display sample images from the dataset, providing insight into the visual differences between normal and pneumonia cases. To address the significant class imbalance, where pneumonia images far outnumber normal ones, class weights are applied during training. This ensures that the model pays equal attention to both classes, avoiding bias towards the majority class.

```
def load_image_paths(base_path, dataset_type):
    # Corrected the path for loading images under 'chest_xray' folder.
    normal_images = glob.glob(os.path.join(base_path, dataset_type, "NORMAL", "*.jpeg"))
    pneumonia_images = glob.glob(os.path.join(base_path, dataset_type, "PNEUMONIA", "*.jpeg"))
    return normal_images, pneumonia_images

# Base path for the dataset (updated from printed path)
BASE_PATH = '/root/.cache/kagglehub/datasets/paultimothymooney/chest-xray-pneumonia/versions/2/chest_xray'

# Load image paths for train, test, and validation sets
train_normal, train_pneumonia = load_image_paths(BASE_PATH, "train")
test_normal, test_pneumonia = load_image_paths(BASE_PATH, "test")
val_normal, val_pneumonia = load_image_paths(BASE_PATH, "val")

# Count the number of images
dataset_stats = {
    "Train": {"NORMAL": len(train_normal), "PNEUMONIA": len(train_pneumonia)},
    "Test": {"NORMAL": len(test_normal), "PNEUMONIA": len(test_pneumonia)},
    "Validation": {"NORMAL": len(val_normal), "PNEUMONIA": len(val_pneumonia)},
}

# Print dataset statistics
print(dataset_stats)
```

Step 2 : Data Process and label

Data preprocessing is a crucial step in preparing raw images for machine learning models. In this code, preprocessing involves resizing images to a uniform size (256x256 pixels) to ensure consistency in input dimensions and applying Gaussian blur to smooth the images, reduce noise, and remove fine details. Normalization is performed by scaling pixel values to the range [0, 1], which helps in faster convergence during model training. Labeling is the process of assigning class labels (e.g., 0 for normal and 1 for pneumonia) to each image, enabling the model to learn the relationship between input features (images) and their corresponding classes for supervised learning tasks. This structured approach ensures a clean, standardized dataset for effective training and evaluation.

```
# Function to load images and labels from paths
def load_images_and_labels(normal_paths, pneumonia_paths, label_normal, label_pneumonia):
    images = []
    labels = []

    # Load NORMAL images
    for path in normal_paths:
        img = cv2.imread(path)
        img = cv2.resize(img, (256, 256))  # Resize to a fixed size
        img = cv2.GaussianBlur(img, (5, 5), 0)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
        labels.append(label_normal)

    # Load PNEUMONIA images
    for path in pneumonia_paths:
        img = cv2.imread(path)
        img = cv2.resize(img, (256, 256))  # Resize to a fixed size
        img = cv2.GaussianBlur(img, (5, 5), 0)
        img = img / 255.0  # Normalize pixel values
        images.append(img)
        labels.append(label_pneumonia)

    return np.array(images), np.array(labels)


# Load training images and labels
X_train, y_train = load_images_and_labels(train_normal, train_pneumonia, label_normal=0, label_pneumonia=1)

# Load test images and labels
X_test, y_test = load_images_and_labels(test_normal, test_pneumonia, label_normal=0, label_pneumonia=1)

# Load validation images and labels
X_val, y_val = load_images_and_labels(val_normal, val_pneumonia, label_normal=0, label_pneumonia=1)
```

Step 3: Image Augmentation

To enhance the model's robustness and prevent overfitting, image augmentation techniques are applied to the training data. Using TensorFlow's ImageDataGenerator, transformations such as random rotation, horizontal and vertical shifts, zooming, flipping, and shearing are performed. These augmentations simulate real-world variability in X-ray images, enabling the model to generalize better to unseen data.

```
[ ]  # Using Keras ImageDataGenerator to apply transformations to the images in real-time
     from tensorflow.keras.preprocessing.image import ImageDataGenerator

     datagen = ImageDataGenerator(
         rotation_range=20,  # Random rotations between -20 to 20 degrees
         width_shift_range=0.2,  # Random horizontal shifts
         height_shift_range=0.2,  # Random vertical shifts
         shear_range=0.2,  # Random shear transformations
         zoom_range=0.2,  # Random zoom
         horizontal_flip=True,  # Randomly flip images horizontally
         fill_mode='nearest'  # Fill any empty pixels resulting from transformations
     )

     # Fit the generator on the training data
     datagen.fit(X_train)
```

Step 3: Convolutional, Pooling, and Batch Normalization Layers

The core of the model is a Convolutional Neural Network (CNN) architecture comprising three convolutional layers, each followed by a max-pooling layer. The convolutional layers use filters (kernels) to extract spatial features such as edges and textures from the input images. The first convolutional layer applies 32 filters, followed by 64 and 128 filters in subsequent layers, progressively capturing more complex features. Max-pooling layers downsample the feature maps, reducing their spatial dimensions while preserving critical information, thereby lowering computational costs. Batch normalization is employed to standardize the outputs of convolutional layers, accelerating training and improving model stability. Dropout layers are included to prevent overfitting by randomly deactivating a fraction of neurons during training, ensuring the model learns generalizable patterns.

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 256, 256, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 128, 128, 32)	0
conv2d_13 (Conv2D)	(None, 128, 128, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 64, 64, 64)	0
dropout_12 (Dropout)	(None, 64, 64, 64)	0
conv2d_14 (Conv2D)	(None, 64, 64, 128)	73,856
max_pooling2d_14 (MaxPooling2D)	(None, 32, 32, 128)	0
dropout_13 (Dropout)	(None, 32, 32, 128)	0
flatten_5 (Flatten)	(None, 131072)	0
dense_12 (Dense)	(None, 128)	16,777,344
dropout_14 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 1)	129

Total params: 16,870,721 (64.36 MB)
Trainable params: 16,870,721 (64.36 MB)
Non-trainable params: 0 (0.00 B)

Step 4: Model Compilation, Optimizer, and Loss Function

The model is compiled with the Adam optimizer, a robust and efficient algorithm that adjusts learning rates dynamically for each parameter, speeding up convergence. The loss function

chosen is binary cross-entropy, which is ideal for binary classification tasks like distinguishing between normal and pneumonia images.

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

Where:

- y_i: The actual label (0 for "NORMAL", 1 for "PNEUMONIA").
- $\hat{y}_i$: The predicted probability output by the model for the positive class (using the sigmoid activation).
- N: Total number of samples.

Accuracy is used as the performance metric to monitor the model's ability to classify images correctly. The combination of these components ensures that the model is optimized effectively for the task while maintaining a focus on interpretability and ease of convergence during training.

Step 5: Training the Model

The model is trained using the augmented dataset for 12 epochs to ensure its ability to handle real-world variations in chest X-ray images. Early stopping is implemented to halt training when validation loss ceases to improve for a set number of epochs, preventing overfitting. A learning rate reduction mechanism dynamically decreases the learning rate when validation accuracy plateaus, allowing for finer weight updates as the model converges.

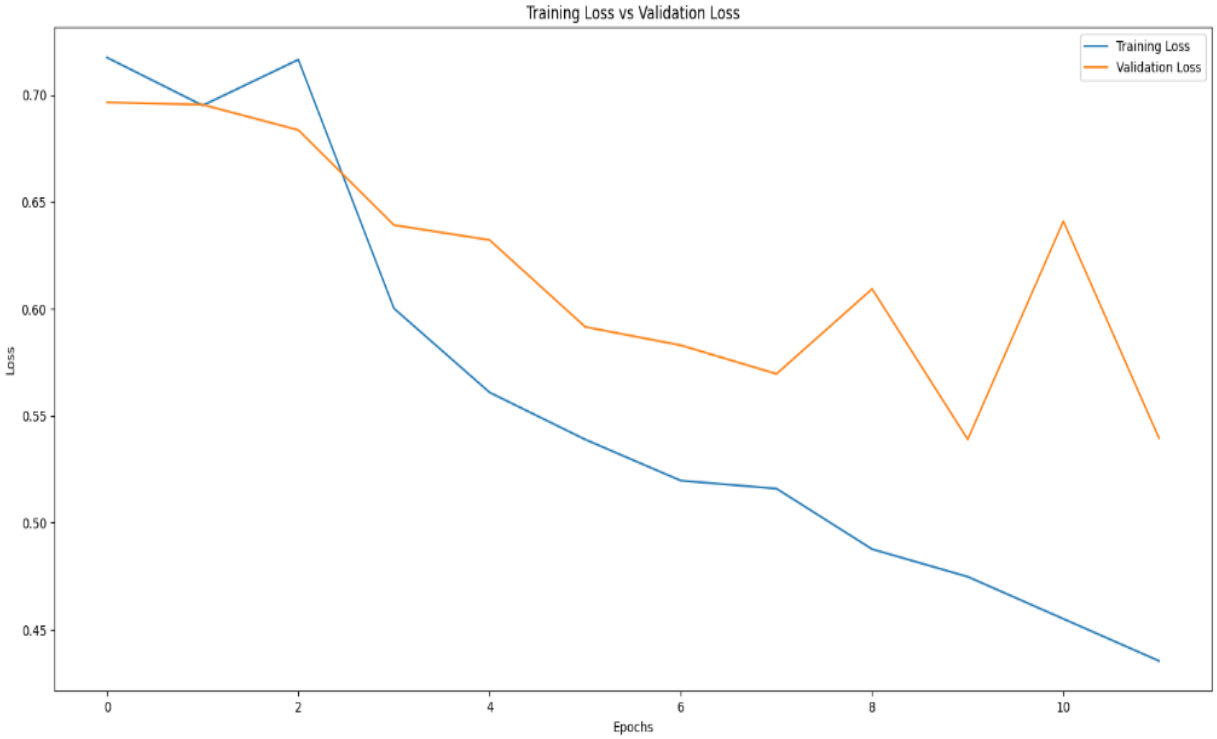
Learning Transfer

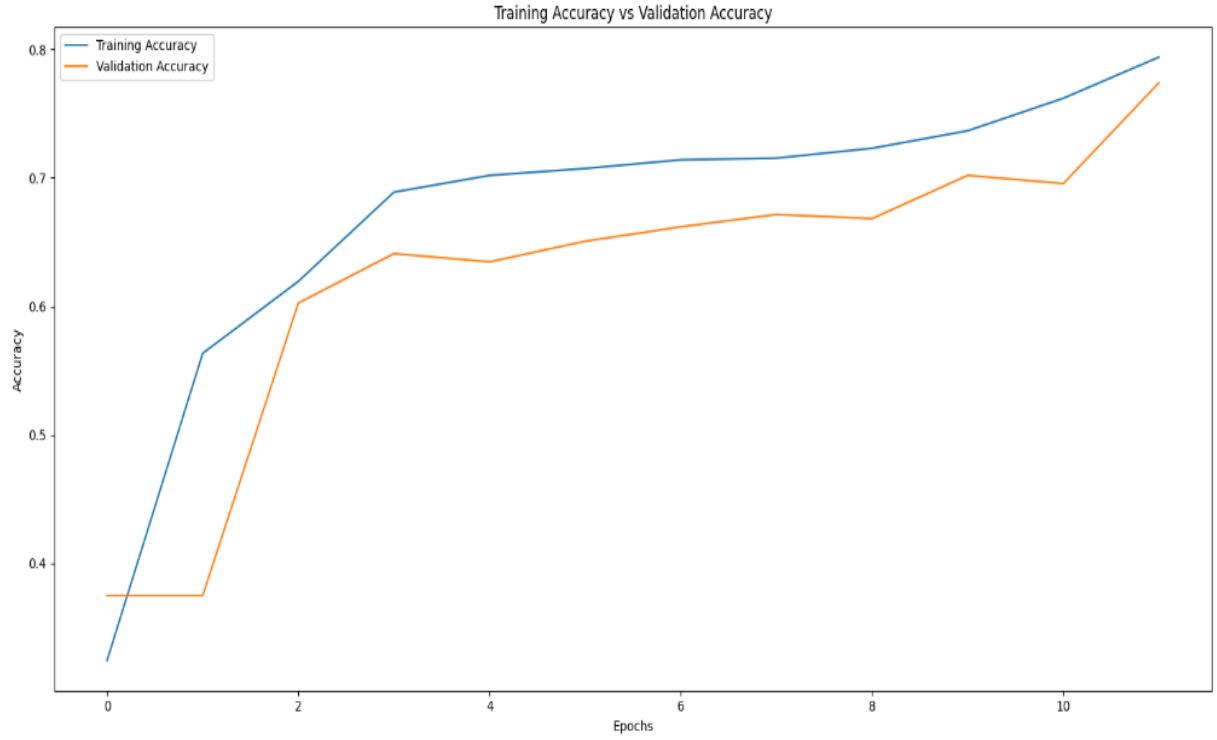
Although this model is trained from scratch, transfer learning could also be applied using pre-trained CNN models like VGG16, leveraging their already-learned features for better performance on medical images. This approach is particularly useful when datasets are small or when computational resources are limited.

Model Performance Measurement

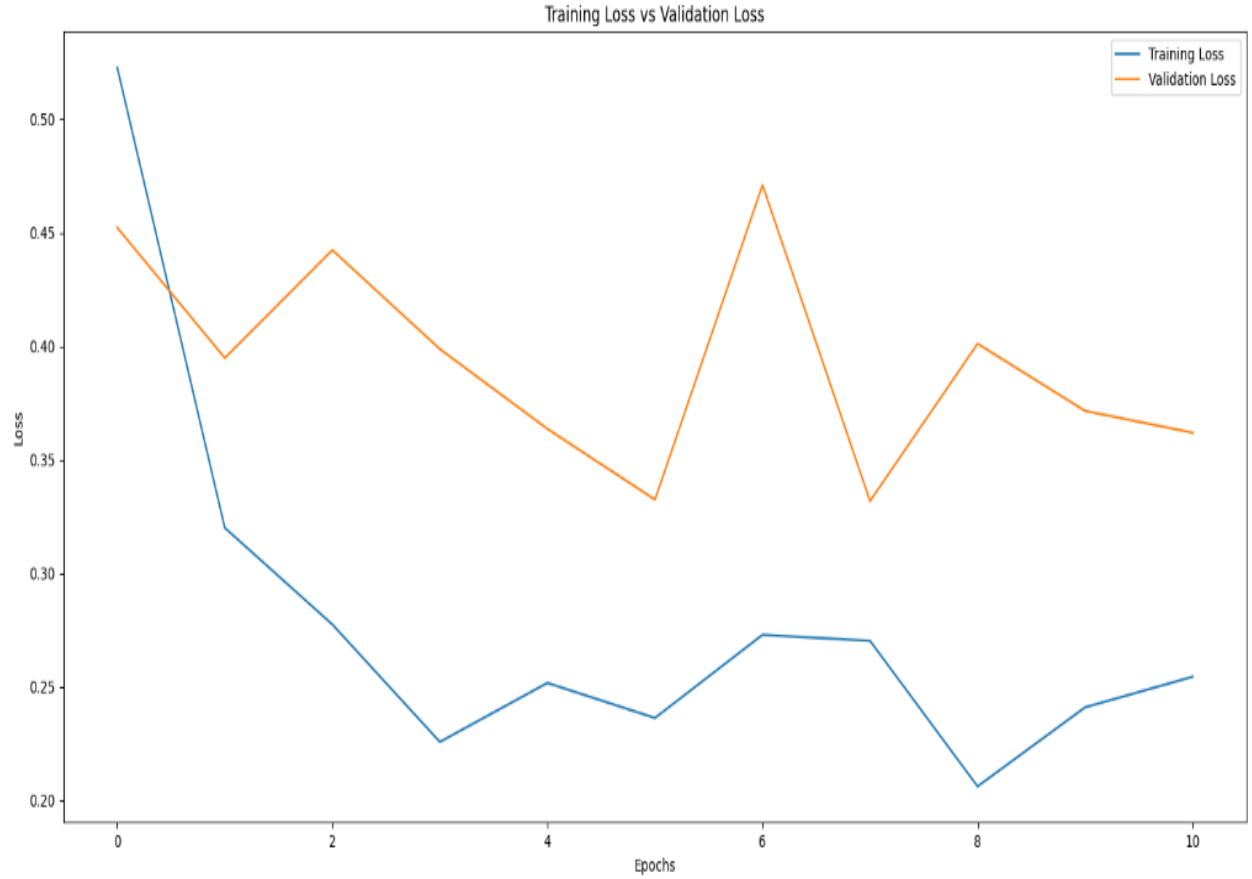
The model's performance is evaluated using metrics like loss and accuracy on the test set. Accuracy represents the percentage of correctly classified images, while the loss indicates how well the model predictions align with the actual labels.

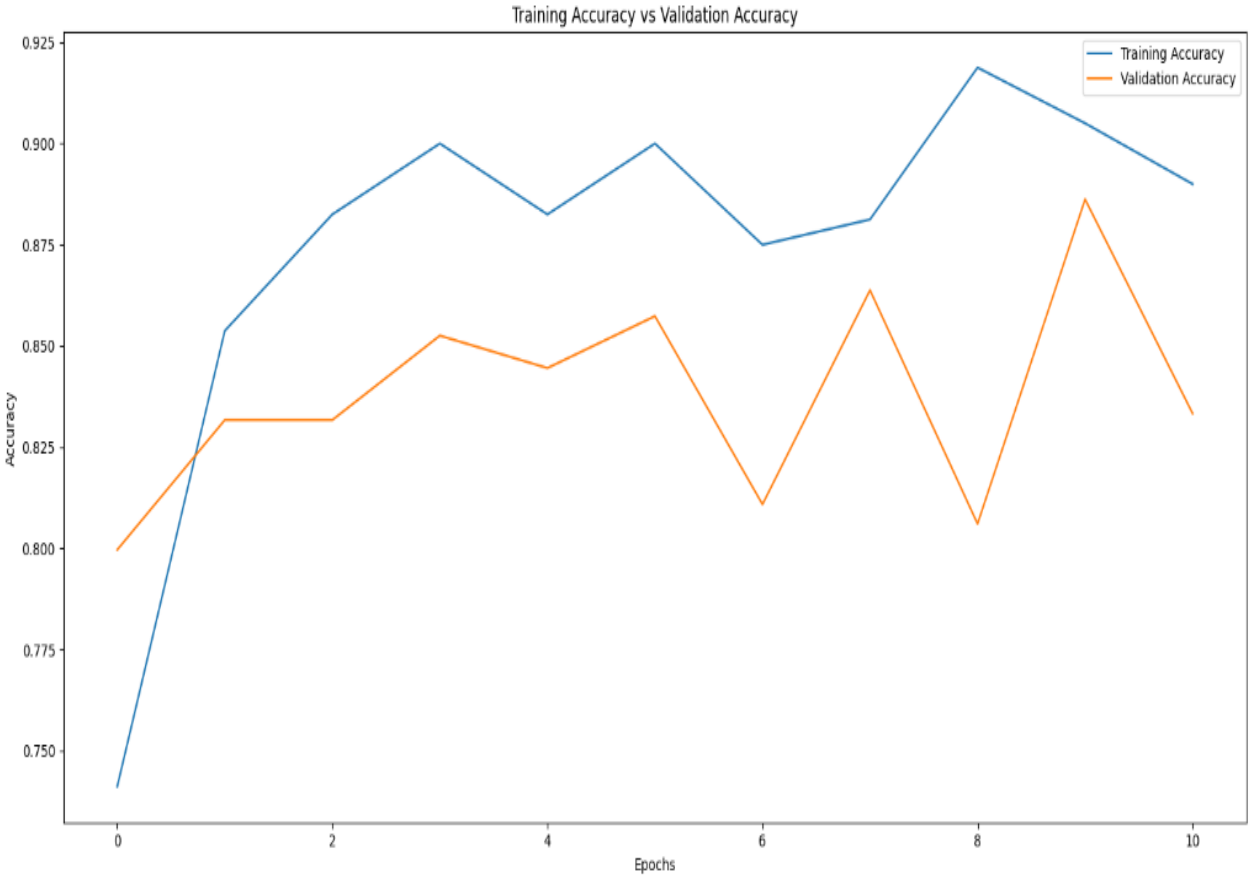
Model 1:


Training Loss vs Validation Loss
Training Loss
Validation Loss
Loss
0.70
0.65
0.60
0.55
0.50
0.45
0
2
4
6
8
10
Epochs

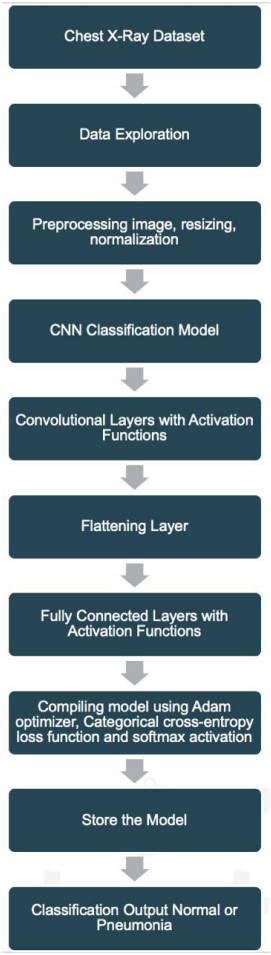

Training Accuracy vs Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
10
Epochs

Model 2 VGG16:





Implementation Approach Summary



The implementation of this CNN model demonstrates a systematic approach to solving the problem of pneumonia detection from chest X-rays. By focusing on data preprocessing, augmentation, and careful architecture design, the model achieves high accuracy and generalization. The use of techniques like class weighting, dropout, and learning rate scheduling ensures that the model is robust to class imbalance and overfitting. This implementation provides a foundation for further experimentation, such as transfer learning or ensembling, to enhance diagnostic accuracy and support medical professionals effectively.

5.Results and Oberservations

The comparison between two models highlights the effectiveness of the implemented approaches in automating pneumonia diagnosis from chest X-ray images. **Model 1** achieved an accuracy of **81.25%** with a loss value of **0.4438**, reflecting a reasonable alignment between predictions and actual labels. However, **Model 2** outperformed it significantly, attaining a higher accuracy of **87.5%** and a lower loss value of **0.3533**, demonstrating more robust learning and improved performance. The preprocessing steps, including resizing, normalization, Gaussian blur, and data augmentation, were pivotal in enhancing the generalization capabilities of both models. Observations of consistent improvements in training and validation metrics in **Model 2**, along with better loss stabilization, suggest that it successfully addresses overfitting and variability. Overall, while both models effectively

distinguish between normal and pneumonia chest X-rays, **Model 2** provides a more reliable and accurate solution, underscoring the importance of iterative optimization in deep learning pipelines.

Model 1:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.8125 - loss: 0.4438
Loss of the model is -  0.44376909732818604
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8125 - loss: 0.4438
Accuracy of the model is -  81.25 %
```

Model 2:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step - accuracy: 0.8750 - loss: 0.3533
Loss of the model is -  0.35332977771759033
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.8750 - loss: 0.3533
Accuracy of the model is -  87.5 %
```

Team members and their work

Harika Yendapalli: Dataset Preparation and Exploratory Data Analysis

1. Tasks:
    - Loading datasets and verifying structure (train, test, and validation).
    - Exploratory data analysis (EDA).
    - Implement data preprocessing.
    - Manage class imbalance using techniques such as class weights.
2. Code Responsibility:
    - Implement functions to load, process, and clean datasets.
    - Write code for visualization and class distribution analysis.
    - Generate preprocessed datasets ready for model input.
3. Report Responsibility:
    - Document the dataset's structure and characteristics.
    - Include graphs, sample images, and statistical analysis of the dataset.

Vasu Patel: Data Augmentation and CNN Model Implementation

1. Tasks:
    - Apply data augmentation using techniques like rotation, zoom, and flipping.
    - Design and implement the CNN model:
    - Implement callbacks (early stopping, learning rate reduction).
2. Code Responsibility:
    - Write augmentation pipelines.
    - Implement the cnn_model_1 function and callbacks.
    - Ensure the model is correctly compiled and ready for training.
3. Report Responsibility:
    - Document the CNN architecture, including diagrams if possible.
    - Explain data augmentation techniques and their benefits.
    - Include the model summary output.

Tanay Malavia: Training, Evaluation, and Results Analysis

1. Tasks:
    - Train the model with augmented data and validate its performance.
    - Evaluate the model on the test set.
    - Plot training and validation accuracy/loss curves.
    - Compare results of Model 1 and any enhancements (e.g., Model 2).
    - Highlight observations on model performance.
2. Code Responsibility:
    - Write code for training and validation loops.
    - Implement evaluation metrics and plots.
3. Report Responsibility:
    - Present evaluation metrics and performance comparisons.
    - Include plots of accuracy vs. loss curves.
    - Summarize key findings and provide insights into the model's strengths and limitations.

Overall Collaboration

- Ensure consistency in coding style, naming conventions, and dataset paths.
- Share insights and cross-check each other's work during weekly sync-ups.
- Combine individual contributions into a cohesive final report and codebase.