Convoluted Neural Networks (CNNs) using Python

Hari Kishan Gullapalli,

ghk235@gmail.com

Disclaimer

I prepared this document collecting information from various sources. These include research publications from different scientists, online blogs, Super Data Science courses and multiple sources from Google search. Although the text is in my own words, I do not own any intellectual property on this material and am not gaining any monetary benefit out of this. Secondly, I prepared this material for my own consumption and may or may not be 100% accurate. Lastly, this material will be revised as and when needed and may or may not be communicated to whoever is reading. If anyone got their hands on this, kindly use it as a secondary or tertiary reference text and nothing more than that.

Context

This material is being prepared as a personal reading material with necessary theoretical knowledge required to work on the SDS Community collaborative project CP#21 – Medical X-ray image classifier: Pneumonia Detection. So, all references, knowledge and information may revolve around this project initially. Later, when working on other projects, the scope of this document will be expanded.

Project Brief

Predict if the patient has pneumonia or not by analyzing the X-ray image of the patient's lungs. We will be using CNN deep learning neural network to execute this project. We will train a CNN with a bunch of images from a publicly available X-ray imagery data from a hospital in China. Finally, we will use Streamlit to deploy the CNN and GitHub to manage the code repository.

Future Scope

Using the similar approach, try to classify other serious medical conditions from CT scans, MRI scans or Contrast MRI scans. This will solidify the learning acquired on this project. These exercises are subject to the availability of the required datasets.

Medical Background

What is a neuron?

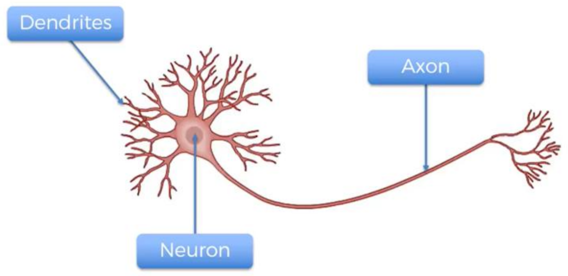
A nerve cell that transmits information throughout the body. Neurons are the basic units of the brain and nervous system, and they allow us to think, feel and move.

Neurons process the input signals and send output signals to the spinal cord and from there to muscles. There are two types of neurons, sensory neurons and motor neurons.

There are three parts of a neuron. Cell body (nucleus), axon (long tail like structure sending signals) and Dendrites (branch like structure receiving signals).

Synapse is the nerve junction where signal transmission happens from one neuron to another. It is essentially a small gap between the axon of one neuron to the dendrites of another.

What we do in neural networks is similar in architecture to the human body and neuron structure.



What is Pneumonia?

Pneumonia is a lung infection caused by bacteria, viruses or fungi. It can be caused to one lung or both the lungs. It is a serious infection where the air sacs in the lungs are filled with fluids, pus and other such liquids. Pneumonia can be treated.

It occurs in 4 stages.

a. Congestion – Occurs within 24 hours, lungs may be heavy, boggy and red. Early symptoms include cough and fatigue
b. Red Hepatization – 2nd stage and occurs after 2 to 3 days of 1st stage. Lungs will appear red and gray like liver, hence called hepatization
c. Grey Hepatization – 3rd stage and occurs 2 to 3 days of 2nd stage. It also appears like liver.
d. Resolution – final and 4th stage where it starts around 10th to 12th day. Symptoms improve and patients begin to feel better.

Pneumonia can be very serious and can cause death. Complications include respiratory failure, sepsis, and lung abscess.

Based on the size, it can be classified into two categories:

a. Lobar Pneumonia (Non-Segmental Pneumonia) – This type of the pneumonia affects an entire lobe of a lung, resulting in a larger visible area on imaging
b. Bronchopneumonia – Smaller patches scattered throughout the lungs, appearing as smaller areas of consolidation on imaging.

The right lung has 3 lobes, and the left lung has 2 lobes. If most of the lobe or entire lobe is filled with Pneumonia, it is called a lobar Pneumonia. For our purposes, we can conclude that the Pneumonia is Lobar Pneumonia if it is spread 33% or more in a lung.

Using CNN, we can further identify the subtype, but there should be enough data for the same. May not be possible in the current project.

CNN Complete Theory

A Convolutional Neural Network (CNN) is a type of deep learning neural network architecture often used in solving problems using computer vision. Computer vision is a field where the computer learns and processes images and video feed data.

The CNN architecture uses the method of convolution instead of matrix multiplication to arrive at the result like in the case of normal neural networks. CNN uses convolution which combines two functions to show how one changes the shape of the other.

If we ignore the mathematical components of how a CNN works, it is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good perception.

X-ray classification for Pneumonia detection is a classification problem. Practical use case is to classify new x-rays and confirm with reasonable accuracy that the patient has Pneumonia or not. This strengthens the radiologist's analysis and will act as an additional confirmation to the patient.

Other use cases are self-driving cars, facial recognition, process speech and audio, classify time-series data among others.

Structure of an Image

For a machine, image is nothing but numbers in a certain pattern. An RGB (color) image is nothing but a matrix of pixel values in 3-planes corresponding to each color (Red, Green, Blue). Each pixel value is made up of 3 RGB components ranging from 0 to 255. So, for example, if we must obtain the RGB pixel value of pinkish blue, it would be RGB (0,25,51).

For greyscale (black and white) images, there will only be one number denoting the darkness of the cell. In all pixel notations, 0 is the darkest and 255 is the lightest. Hence, if we got to denote black in RGB setup, it would be RGB (0,0,0) and white color would be (255,255,255).

"Kernel" in CNN

Kernel is the term used to describe small matrix of weights that is used to perform convolution operations on input data. It can also be termed as a filter which slides over the input matrix to extract features, making Kernel more accurate term to use than weights. Kernels are fundamental to CNNs because they help in detecting edges, textures, patterns and complex structures in data.

Kernels are typically small matrices with dimensions like 3x3 or 5x5.

We will start with some kernel values and we finetune those values as the algorithm runs leading to less delta between the expected value and the actual value. This step is called backpropagation which will be understood later.

The result of applying a kernel to an image is called a feature map (activation map).

Important Parameters of Kernels

There are 3 important parameters of Kernels.

Kernel Size

a. Typical sizes are 3x3,5x5,7x7. Always odd sizes are used typically to maintain a central reference pixel.
b. Smaller Kernels capture finer details and reduce computational cost.
c. Larger Kernels capture more global patterns but require more computation.

Stride

The step size of the kernel as it moves across the image. Usually, 1 step is used preserving more details. If 2 or more stride is used, it reduces output leading to downsampling.

Padding

As the kernel reaches to the end of the image pixel matrix, there may not be pixel values to convolve using the kernel that we chose. So, we must decide on how the kernel handles the image borders in CNNs.

There are 2 common types of padding used in CNNs:

1. Valid Padding (No Padding) – The kernel only applies where it fully fits, reducing output size.
2. Same Padding (Zero Padding) – Adds extra pixels (usually zeros) around the image to preserve size.

Common Types of Kernels

There are 3 categories of kernels usually used in CNN applications.

a. Edge Detection Kernels
    a. Sobel Kernel – Detects horizontal and vertical edges [(-1,0,1), (-2,0,2), (-1,0,1)]
    b. Prewitt Kernel – Alternative to Sobel, less computationally expensive [(-1,0,1), (-1,0,1), (-1,0,1)]
    c. Laplacian Kernel – Detects both horizontal and vertical edges in one go. [(0,-1,0), (-1,4,-1), (0,-1,0)]
b. Blurring/Smoothing Kernels
    a. Box Blur (Average Filter) – Replaces each pixel with average of its neighbors. 1/9[(1,1,1), (1,1,1), (1,1,1)]
    b. Gaussian Blur Kernel (Weighted Smoothing) – Gives more weight to central pixels to retain structure while reducing noise. 1/16[(1,2,1), (2,4,2), (1,2,1)]
c. Sharpening Kernels
    a. Basic Sharpening Kernel – The central pixel is amplified, and surrounding pixels are subtracted. [(0,-1,0), (-1,5,-1), (0,-1,0)]
    b. Unsharp masking kernel (High Boost Filtering) – Enhances edges while preserving details. [(-1,-1,-1), (-1,9,-1), (-1,-1,-1)]
d. Embossing Kernels – Used to highlight edges and give 3D effect to images. [(-2,-1,0), (-1,1,1), (0,1,2)]

Custom learnable kernels in CNNs

Unlike manually defined kernels, CNNs learn kernels automatically through training.

In early layers, they detect simple edges and textures

In deeper layers, they capture complex patterns like shapes, objects, faces.

The backpropagation algorithm updates kernel weights based on gradient descent to optimize feature extraction for the given dataset.

The following are the factors that should be considered before picking a kernel.

a. **Nature of the Input Data**

Different types of data require different kernel operations:

- Images → Edge detection, texture extraction, smoothing.
- Time-series data → Trend detection, smoothing filters.
- Text (NLP with CNNs) → 1D convolutions for n-gram feature extraction.

Example: A facial recognition system may need edge and contour detection kernels, while speech processing may need spectrogram-based filters.

b. **Type of Features to Extract**

What characteristics in the data are most important?

- Edges & contours → Sobel, Prewitt, Laplacian kernels.
- Textures & patterns → Gabor filters.
- High-frequency details → Sharpening kernels.
- Low-frequency structures → Smoothing/Blurring filters.

Example: In biometrics (fingerprint recognition), ridge and valley detection kernels are essential.

c. **Scale & Size of Objects in Data**

What size of patterns need detection?

- Small features → Use small kernels (3×3, 5×5) for fine details.
- Larger structures → Use bigger kernels (7×7, 9×9) for more general patterns.

Example: Satellite image processing requires larger kernels (e.g., 7×7) to detect large geographical features.

d. **Edge Detection Sensitivity**

If detecting sharp boundaries, choose edge-sensitive kernels:

- Low sensitivity (soft edges) → Sobel, Prewitt.
- High sensitivity (sharp edges) → Laplacian, Canny Edge Detectors.

Example: Medical imaging (e.g., detecting tumors in MRI scans) needs high-sensitivity edge detection.

e. Noise Levels in Data

If the data contains noise, use filters to remove it:

- High noise → Use Gaussian blur, Median filters.
- Low noise → Use sharpening and high-pass filters.

Example: Security camera footage needs denoising filters to remove low-light artifacts.

f. Computational Cost & Efficiency

Complex kernels require more computation:

- Smaller kernels (3×3, 5×5) → Faster processing, but may miss global patterns.
- Larger kernels (7×7, 9×9) → Capture larger structures but are computationally expensive.

Example: Real-time object detection (self-driving cars) uses 3×3 and 5×5 kernels for quick processing.

g. Symmetry & Orientation of Features

Different kernels detect features in specific orientations:

- Vertical edges → Sobel-X, Prewitt-X.
- Horizontal edges → Sobel-Y, Prewitt-Y.
- Diagonal patterns → Gabor filters.

Example: Handwriting recognition (OCR) benefits from direction-sensitive filters to detect slanted letters.

h. Multi-scale Feature Extraction

Some applications require multi-scale kernels:

- Small kernels (3×3) for fine details.
- Medium kernels (5×5) for intermediate textures.
- Large kernels (7×7, 9×9) for broad patterns.

Example: Face detection uses multiple kernel sizes to detect facial features at different scales.

i. Learnability & Trainability in CNNs

Instead of manually selecting kernels, let CNNs learn them:

- Fixed kernels (Sobel, Gaussian) → Useful in traditional image processing.
- Learnable kernels (in deep CNNs) → Automatically optimized via training.

Example: Deep learning for object detection (YOLO, Faster R-CNN) learns optimal kernels from data instead of using predefined ones.

Note: Modern CNNs, do not require manual selection of kernels. Instead, they learn the optimal kernels automatically during training via backpropagation and gradient descent.

Random Initialization:

- CNNs start with randomly initialized learnable kernels (filters).

Feature Extraction Through Convolution:

- These kernels slide over the image and detect low-level features (edges, textures) in shallow layers.
- Deeper layers extract complex features (shapes, objects, semantic meaning).

Backpropagation & Gradient Descent:

- The CNN updates kernel values based on loss function feedback.
- Useful kernels for the task emerge automatically over multiple training epochs.

Example: A CNN trained for X-ray diagnosis may learn edge-detecting filters like Sobel filters in early layers, but deeper layers will specialize in detecting diseases, fractures, or tumors.

When is manual kernel used?

Traditional Image Processing

- Before deep learning, edge detection and noise filtering used manually chosen kernels like Sobel, Laplacian, and Gaussian filters.

Medical & Scientific Imaging

- Some applications still apply domain-specific filtering (e.g., X-ray enhancement using predefined kernels).

Feature Engineering in Low-Data Scenarios

- If a dataset is small, handcrafted kernels help highlight important patterns before CNN training.

Hybrid Models

- Some modern architectures combine handcrafted filters + CNN-learned filters for improved performance.

Example: Fingerprint recognition systems sometimes use Gabor filters for preprocessing before using CNNs.

Layers in CNN

The following are the layers in CNN:

a. Input Layer – Accepts raw data (ex. Images)
b. Convolutional Layer – Extracts spatial features using filters
c. Activation Layer – ReLU, Leaky ReLU etc – Introduces Non-Linearity
d. Pooling Layer – Max/Avg Pooling – Reduces dimensionality
e. Normalization Layer – BatchNorm, LayerNorm etc – Stabilizes learning
f. Dropout Layer – Prevents Overfitting
g. Fully Connected (Dense) Layer – Makes final predictions
h. Output Layer (Softmax, Sigmoid etc) – Produces Class probabilities.

a. Input Layer

The input layer is the first layer in a Convolutional Neural Network (CNN), responsible for receiving and processing raw image data before passing it to the next layers for feature extraction.

The input layer does not perform computations but acts as a data feeder to the network. It takes images in pixels and converts them into a format that CNN can process.

The input image is represented as a grid of pixel values. A grayscale image has 1 channel and an RGB image has 3 channels.

Also, CNNs work best when the inputs are normalized. Pixel values (0 to 225) are scaled 0-1 or -1 to 1 for better performance.

Common normalization techniques: Min-Max Scaling, Mean Normalization.

CNNs process images as tensors. The image data is reshaped into (Batch, Height, Width, Channel) format.

For example: A batch of 32 images with size 224x224 and having 3 channels (RBG) would be represented in the tensor format as (32,224,224,3).

Input Layer Dimensions for Different Image Types

Image Type	Shape (H × W × C)	Example
Grayscale	(512, 512, 1)	X-ray, MRI, Handwriting
RGB Color	(224, 224, 3)	Face Recognition, Object Detection
Multi-Spectral	(128, 128, 10)	Satellite Images (Hyperspectral)
3D Medical Scan	(256, 256, 256, 1)	CT scans (Depth Dimension)

b. Convolutional Layer

The Convolutional Layer applies filters (kernels) to the input image to detect patterns such as edges, shapes, and textures. It slides a small filter (e.g., 3×3, 5×5) across the image and performs element-wise multiplication and summation. The result is a Feature Map, which highlights important regions in the image.

CNNs stack multiple convoluted layers to learn complex patterns.

c. Activation Layer

The Activation Layer in a Convolutional Neural Network (CNN) is responsible for introducing non-linearity into the model. Non-linearity allows the network to learn complex patterns, making it possible to approximate any function (i.e., learn from data with complex relationships). Without activation functions, a neural network would essentially be a linear regression model, regardless of how many layers it has.

It does – Introducing non-linearity, allowing neurons to fire, enabling learning

Types of activation functions:

a. Sigmoid (Logistic) Function

Range – 0,1. Primarily used for binary classification problems, easy to compute but with extreme values of x, causes slow learning (vanishing gradient problem).

b. Hyperbolic (Tanh) Tangent

Range -1 to 1. Like sigmoid but centered around 0 making it more effective. Reduced vanishing gradient problem, but still there is this problem.

c. Rectified Linear Unit (ReLU)

Range – 0 to Infinity. The most popular activation function for hidden layers in CNNs. Solves the vanishing gradient problem and is fast and computationally efficient. Dying ReLU problem – Neurons can become inactive if their inputs are always negative.

d. Leaky ReLU

Range -infinity to +infinity. A variant of ReLU to avoid dying ReLU problem. Allows small negative values for xxx to maintain some gradient during backpropagation.

e. Softmax

Range 0 to 1 for each output. Mainly used in the output layer for multiclass classification problems.

Converts outputs into probabilities, making it useful for multi-class classification. Can be computationally expensive due to the exponentiation and summing over the entire output.

Activation Function	**Usage**	**Pros**	**Cons**
Sigmoid	Output layer for binary classification.	Output values between 0 and 1.	Susceptible to vanishing gradients.
tanh	Hidden layers for deep networks.	Zero-centered, mitigates some vanishing gradient issues.	Still suffers from vanishing gradients.
ReLU	Hidden layers for deep networks.	Fast convergence, no vanishing gradients.	Can "die" and stop updating.
Leaky ReLU	Hidden layers for deep networks.	Avoids dying ReLU problem.	Choice of alpha is important.

PReLU	Hidden layers (same as Leaky ReLU but learnable).	Flexible, learnable leaky factor.	More parameters to learn.
ELU	Hidden layers (especially for very deep networks).	Smooth gradients, avoids vanishing gradient.	Computationally expensive.
Softmax	Output layer for multiclass classification.	Converts outputs into probabilities.	Computationally expensive.
Swish	Hidden layers (alternative to ReLU).	Smooth and better performance in some networks.	More computationally expensive.

d. Pooling Layer

Pooling Layer reduces the spatial dimensions while retaining the important features. It reduces computation, makes CNN invariant to small transitions, and helps in preventing overfitting.

A pooling function summarizes the features in a region of the feature map. The size of the region is determined by the pooling filter.

There are 3 types of pooling layers:

Max Pooling (Most Commonly Used) - Selects the maximum pixel value in a window. Preserves sharp features like edges while reducing the image.

Average Pooling – Computes the average value of pixels in a window. This method is used when exact feature details are less important.

Global Average Pooling (GAP) – Takes average of entire feature map and reduces it to 1x1 per channel. Used in modern architectures like ResNet, MobileNet instead of fully connected layers.

e. Normalization Layer

Normalization stabilizes training, prevents large variations in gradients and helps faster convergence.

It scales activations to ensure they are in a well-defined range.

There are three types of normalizations:

1. Batch Normalization
2. Layer Normalization
3. Instance Normalization

f. Dropout Layers

A Dropout Layer is a regularization technique used in Convolutional Neural Networks (CNNs) and other deep learning models to prevent overfitting.

- It works by randomly dropping (disabling) a fraction of neurons in a layer during each training iteration.
- The disabled neurons do not participate in forward or backward propagation during that iteration.
- This forces the network to learn more robust and generalized features instead of relying on a few dominant neurons.

Overfitting in CNNs

- CNNs have millions of trainable parameters.
- They can memorize training data instead of learning generalizable features.
- This leads to overfitting, where the model performs well on training data but poorly on unseen test data.

How Dropout Helps?

- Dropout prevents neurons from relying too much on specific features.
- Each neuron learns to contribute independently.
- It introduces stochasticity in learning, making the model more robust.

During training:

A fraction (e.g., 20%-50%) of randomly selected neurons in the layer are temporarily disabled.

Disabled neurons do not pass information forward or receive updates during backpropagation.

The remaining active neurons learn more robust representations.

During inference (testing/prediction):

All neurons are used, but their outputs are scaled by the dropout rate to maintain expected activation levels.

For example, if 50% dropout was used during training, then during inference, the neuron activations are multiplied by 0.5 to balance the network.

Where to apply?

After fully connected (Dense) layers, especially before the final output layer.

Between convolutional blocks, but with a lower rate (e.g., 0.2-0.3).

The dropout rate p is a hyperparameter that must be tuned.

Dropout Rate	Effect
0.1 - 0.3	Mild regularization (for large datasets).
0.3 - 0.5	Stronger regularization (for medium datasets).
>0.5	Very aggressive (can cause underfitting).

Start with 0.5 for Dense layers and 0.2-0.3 for Conv layers.

Adjust based on validation loss trends.

Relevance for X-ray imaging:

YES, when used properly!

- Helps generalization in small datasets (common in medical imaging).
- Reduces the risk of overfitting to artifacts (like scanner noise).
- Can be combined with data augmentation for better results.

NO, if overused!

- Can cause underfitting, leading to poor model performance.
- Too much dropout in early layers removes important spatial features.

Best Practice for X-ray Classification CNNs:

- Use SpatialDropout2D (0.2-0.3) in convolutional layers.
- Use standard Dropout (0.5) in fully connected layers.
- Monitor validation loss to adjust dropout rates.

g. Fully Connected Dense Layer

Fully Connected (FC) layers and Output layers play a crucial role in decision-making in Convolutional Neural Networks (CNNs). They come after convolutional and pooling layers and are responsible for final feature interpretation and classification or regression tasks.

A Fully Connected (FC) layer, also known as a Dense layer, connects every neuron from the previous layer to every neuron in the current layer.

- Purpose: Extracts high-level features from CNN layers and makes predictions.
- Structure: Each neuron is connected to every neuron in the next layer.
- Activation Functions: Uses ReLU, sigmoid, softmax, etc., to introduce non-linearity.

How it Works in CNNs

1. Convolutional and Pooling layers extract spatial features from images.

2. These feature maps are flattened into a 1D vector.
3. The Fully Connected layers learn complex relationships between features.
4. The final output layer makes predictions based on FC layer activations.

h. Output Layer

The output layer is the last layer of a CNN, responsible for producing final predictions.

Key Aspects of the Output Layer

- Number of neurons: Equal to the number of classes (e.g., 1 for binary classification, 10 for MNIST digits).
- Activation Function: Determines the type of prediction:
    - Sigmoid → Binary classification.
    - Softmax → Multi-class classification.
    - Linear (No Activation) → Regression tasks.

How to choose output layers?

Binary Classification – 1 Neuron – Sigmoid Function – Best for problems like Pneumonia detection.

Multi-class Classification – No. of classes, Softmax Function – Handwritten Digit Recognition

Regression (Continuous Output) – 1 Neuron – Linear – Predicting age from X-rays.

Step by Step detailed checklist to implement a CNN for Pneumonia X-ray classification

Step 1 – Setup and Preparation

Install Necessary Libraries

We may need the following libraries and packages to get started with the CNN implementation.

Core Libraries: TensorFlow/Keras, PyTorch (choose one).

Data Handling: Pandas, NumPy, OpenCV, Pillow (PIL).

Visualization: Matplotlib, Seaborn.

Data Augmentation: Albumentations, TensorFlow ImageDataGenerator.

Model Evaluation: Scikit-learn (for classification reports, confusion matrix).

Step 2 – Obtain X-ray Dataset

Load the X-ray dataset and organize it into folders.

```
dataset/
  train/
    pneumonia/
    normal/
```

```
val/
   pneumonia/
   normal/
test/
   pneumonia/
   normal/
```

Step 3 – Image Preprocessing

a. Resize Images – Standard CNN images are 224x224, 256x256
b. Convert to greyscale – Optional
c. Normalize pixel values 0 to 1 or -1 to 1
d. Augment Data – Rotation, Flipping, Zooming, Contrast Adjustment

Step 4 – Define CNN architecture

a. Input Layer: 224x224x1 (Greyscale), 224x224x3 (RGB Scale)
b. Convolutional Layers: Use 3x3 kernels with ReLU activation. Add Batch normalization after convolution.
c. Pooling Layers: Max Pooling (2x2) to reduce dimensionality. Average Pooling can be used for feature smoothing.
d. Dropout Layer: Apply Dropout (0.3-0.5) to prevent overfitting
e. Fully Connected (Dense) layers: Flatten feature maps, Use 128 or 256 Neurons in dense layers. ReLU activation for hidden layers
f. Output Layer: Sigmoid activation (for binary classification). SoftMax for multiclass classification.

Step 5 – Model Compilation and Training

a. Loss Function – Choose loss function – Binary Crossentropy and Categorical Crossentropy.
b. Optimizer – Adam, SGD with momentum (for fine-tuning)
c. Metrics – Accuracy, Precision, Recall, F1-Score, AUC-ROC (to measure clinical performance).
d. Split Data into training, validation and test sets – Train (70%) for model training, Validation (15%) for Hyperparameter tuning, Testing (15%) for final model evaluation.
e. Train model – Set batch size of 32 or 64, Train for 10 – 50 epochs, observing loss and accuracy.
f. Use callbacks – Early Stopping (Prevents Overfitting)
g. Model Checkpoint – Saves Best Model

Step 6 – Model Evaluation and Improvement

a. Use the confusion matrix (for false positives and false negatives)
b. Check Precision, Recall and F1 score
c. Compute ROC-AUC for classification confidence
d. If overfitting – Add dropout (increase from 0.3 to 0.5), Add L2 Regularization to FC Layers, Use Data Augmentation more aggressively.
e. If Underfitting – Increase CNN depth (more convolutional layers), Use larger dataset or pretrained models (transfer learning).

Step 7 – Deployment and Integration

a. Use Streamlit App for a simple UI.
b. Can also use Flask/Fast API to create a Web API
c. Deploy to a cloud platform
d. Implement Edge AI for faster diagnosis on devices
e. Use Grad-CAM or LIME to visualize decision-making regions

Special Case – Handling Imbalanced Datasets (Like in this project's case)

There are different ways to handle the imbalanced datasets.

1. Data-level techniques (At Preprocessing stage)
    a. Oversampling (Increasing Minority Class Samples)
    b. Undersampling (Reducing Majority Class Samples)
    c. Data Augmentation
2. Algorithm based techniques (Training-based solutions)
    a. Class Weighing (Binary Loss Function)
    b. Focal Loss
    c. Transfer Learning

Let us look at each in detail:

Data-level techniques

a. Oversampling (Increasing Minority Class Samples)

Random Oversampling

In this method, it increases the number of instances in the minority class by either duplicating existing samples or generating synthetic ones. So, we copy the same images again and again until the imbalance is resolved and train the model on these images.

Problem in this approach is it can lead to overfitting as we are not training on new images, and the model learns these only these patterns leading to overfitting.

Synthetic Minority Oversampling Technique

In this method, it generates synthetic images by interpolating between existing minority class samples. This method reduces overfitting by creating slight variations in the images but can create unrealistic images.

If the output images turn to be highly unrealistic, it will lead to incorrect training on entirety.

b. Undersampling (Reducing Majority Class Samples)

Reduces the number of instances in the majority class by randomly removing samples.

Random Undersampling

Randomly deletes images from the majority class to balance the dataset. This prevents the model from being dominated by the majority class.

But this can lead to loss of valuable data, reducing the overall model performance.

Cluster Based Undersampling

Instead of randomly removing images, it clusters similar images and selects representative ones. This helps in preserving diverse samples. But this is computationally expensive.

c. Data Augmentation

Applies transformations to increase data variability in the minority class.

Common Augmentations are:

h. Rotation (+- 10 degrees)
i. Flipping (Horizontal/Vertical)
j. Zooming (+-10%)
k. Brightness/Contrast Adjustments
l. Gaussian Noise Addition

Best method used in medical imaging like X-rays and MRIs where it is challenging to obtain new data points.

Algorithm-level techniques

a. Class Weighting (Biasing Loss function)

Assigns a higher penalty for misclassifying the minority class.

This prevents bias towards the majority class. But this method needs careful tuning to prevent instability.

b. Focal loss

Modifies cross-entropy loss to focus on hard-to-classify cases. This is a bit better than class weighting method. Class weighting treats all misclassifications equally.

Focal loss prioritizes difficult-to-classify samples by down-weighing easy ones.

c. Transfer Learning

Uses pre-trained CNN architectures (eg. ResNet, VGG16) trained on large datasets (eg. ImageNet). Pretrained models already have strong feature extraction capabilities. They need less data to perform well.

Evaluation Techniques

There are certain metrics and evaluation techniques which can be used in determining the model performance instead of using accuracy which is misleading in the case of an imbalanced dataset.

Three metrics which are predominantly used:

a. Precision
b. Recall
c. F1 – Score

To understand these better, we need to understand the confusion matrix first.

Confusion Matrix

	Predicted Positive (Pneumonia)	Predicted Negative (Normal)
Actual Positive (Pneumonia)	True Positive (TP)	False Negative (FN)
Actual Negative (Normal)	False Positive (FP)	True Negative (TN)

True Positive (TP) → Model correctly predicts Pneumonia.

False Negative (FN) → Model falsely predicts normal when it is Pneumonia

False Positive (FP) → Model wrongly predicts Pneumonia when it is actually normal

True Negative (TN) → Model correctly predicts normal

Let's look at each metric in detail.

a. Precision

Simply put, Precision tells us how many Pneumonia cases are truly Pneumonia?

It answers, out of all the cases my model predicted as Pneumonia, how many are correct?

Precision = TP/(TP+FP)

How to read precision?

High Precision (closer to 1.0) → Fewer false positives (FP)

Low Precision (closer to 0) → Many false positives (FP), meaning the model is misclassifying normal cases as Pneumonia

This metric is highly useful in medical diagnosis, fraud detection, spam filtering where false positives must be low (i.e, we don't want to classify a normal patient as Pneumonia).

b. Recall (Sensitivity or True Positive Rate)

Recall measures, how many of the actual Pneumonia cases were correctly identified.

Out of all actual Pneumonia cases, how many did the model detect?

Recall = TP/(TP+FN)

How to read recall?

High Recall (closer to 1.0) → fewer false negatives.

Low Recall (closer to 0) → many false negatives, meaning the model is missing real Pneumonia cases.

We don't want to miss the real Pneumonia cases, and this metric does tell that.

c. F1-Score (Harmonic mean of precision and recall)

F1-score balances Precision and recall, useful when one is too high and other is too low.

How good is my model at making balanced predictions?

Best when we need both high precision and high recall.

F1-Score = 2x(Precision x Recall)/(Precision + Recall).

How to read F1-score?

High F1-score (closer to 1) → Good balance of precision and recall

Low F1-score (closer to 0) → Either Precision or recall is too low.

Generally, when using CNN for medical imaging, high recall should be favoured.

When the use-case is spam detection or fraud detection, high precision should be favoured.

In general classification tasks, F1-score should be high (Both should be balanced).