Pneumonia Disease Detection using Machine Learning (CNN)

Abdelrahman Wael Abdelrahman Rashad Ammar

(Technical Assistant - ML Specialist)

June 2, 2023

Abstract

Pneumonia is a common respiratory infection that poses a significant threat to public health. Timely and accurate diagnosis is essential for effective treatment. Recently, artificial intelligence (AI) has shown great potential in automating pneumonia detection from medical images, particularly chest X-rays (CXRs).

This project aims to create an AI-based system for detecting pneumonia using deep learning models. It also underscores the collaborative role of AI in healthcare, offering a second opinion and aiding in decision-making.

The primary goal of this project is to harness machine learning algorithms to analyze CXR images and provide accurate and efficient pneumonia diagnosis. The project employs a dataset of CXR images, including both healthy and pneumonia cases, to train and evaluate deep learning models. The method involves implementing the DenseNet deep learning architecture for pneumonia detection.

The models are trained on a large dataset of CXR images and fine-tuned for optimal performance. The results demonstrate the AI system's effectiveness in detecting pneumonia, with the models achieving high accuracy, precision, and recall rates, delivering reliable and consistent diagnoses. The project achieves an overall accuracy of 90% through rigorous evaluation, showing promising performance in detecting pneumonia from CXR images.

Future work includes expanding the dataset to enhance the models' robustness and generalization. Additionally, incorporating more clinical data and exploring other imaging modalities could improve the system's overall accuracy and reliability.

Ongoing research and development in this field hold the potential to advance pneumonia diagnosis and facilitate early intervention and treatment for affected individuals.

Chapter (1)

Introduction & Background

Chapter (1)
Introduction & Background

1.1. Overview

Pneumonia is an infection that affects one or both lungs, causing symptoms such as a cough with phlegm or pus, fever, chills, and breathing difficulties. Unfortunately, it can spread quickly. The causes of pneumonia include bacteria, viruses, and fungi. Each year, around 450 million people (approximately 7% of the global population) are diagnosed with pneumonia, resulting in about 4 million deaths. In the wake of the COVID-19 pandemic, there is a heightened awareness of the importance of rapid pneumonia diagnosis, as the condition can sometimes be fatal.

When patients undergo imaging tests such as X-rays, Computed Tomography (CT), or Magnetic Resonance Imaging (MRI), these images need to be reviewed by skilled doctors or specialists. This process is critical because misdiagnoses can occur; for example, a true pneumonia case might be incorrectly identified as normal. This complexity arises because pneumonia symptoms can vary widely, making early detection challenging.

One significant advancement transforming the medical field is the rapid development of artificial intelligence (AI). AI has proven highly beneficial, offering time savings and increased accuracy in surgical procedures and diagnostic predictions. This project introduces a specialized convolutional neural network (CNN) model designed to verify pneumonia infection through X-ray images. Given that diagnosing pneumonia in real life requires the expertise of professionals, along with laboratory tests, vital signs, and clinical history, this AI model aims to save time and enhance the accuracy of diagnoses, ultimately saving more lives.

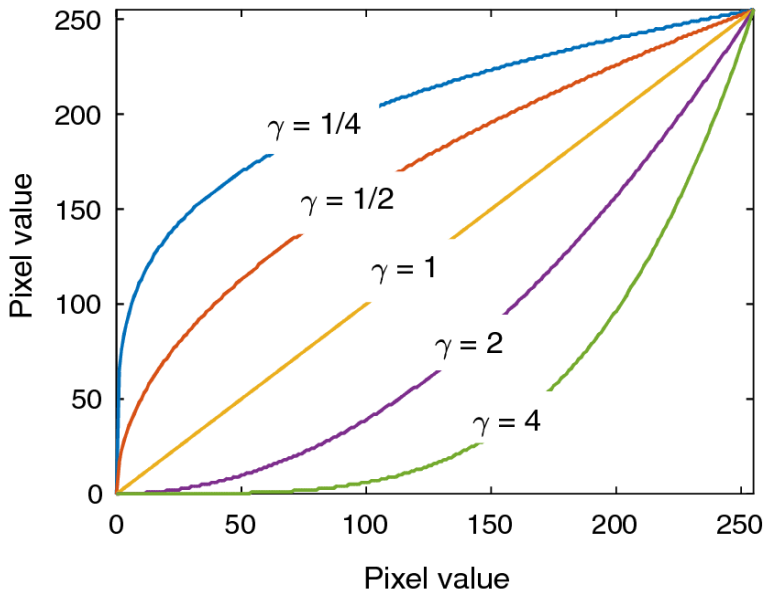
The focus of this project is to develop a robust and reliable AI model capable of determining whether a patient has pneumonia based on chest X-ray (CXR) images. The model is trained on data from numerous previous cases. The system enhances the visibility of features in CXR images through various modifications during the pre-processing stage. It processes these images as two-dimensional digital representations composed of finite digital values, or pixels.

1.2. Image processing

Typically, pixel values represent attributes like gray levels, hues, heights, and opacities. Chest X-rays (CXRs) are usually stored digitally in grayscale format, which simplifies their use with image transformation algorithms. To standardize all images, they are first processed through a grayscale filter, which uses only a single channel. In grayscale images, each pixel value solely indicates light intensity.

Image enhancement, a common application of digital image processing (DIP) techniques, aims to improve quality and reduce noise. Gamma correction is often applied to enhance image contrast and reveal more details. For an 8-bit grayscale image, with 256 levels of pixel intensity, pixel values are normalized to range from 0 to 1. These values are then adjusted by raising them to a power greater than or less than one, depending on the required correction. If lower intensity levels need to be darkened, the power value must be greater than one. This technique is particularly useful for X-rays to highlight bright features against a darker background.

Gamma transformation curves for different gamma values show that gamma values less than one increase image brightness, while gamma values greater than one darken the image.

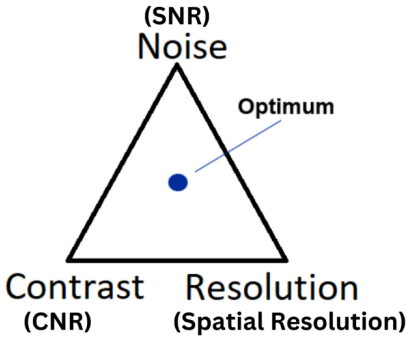


1.3. The X-ray technique

It is quick and painless. This method usually gives quick, clear imaging results to show whether pneumonia is present or not and if there are other abnormalities associated with it. Inexpensive and fast compared to other examinations and compared to CT scans.

1.3.1. X-ray Image Characteristics

The three most common parameters used to measure the quality of an image are the signal-to-noise ratio (SNR), the spatial resolution (R), and contrast to noise ratio (CNR). An image ideally has a high value of each of these parameters, however, there are frequent trade-offs among the parameters, and compromises must be made. The decision will depend on the usage of the X-ray. You can reach the optimum case at the center of the three-parameter triangle.



.

1.3.1.1. Contrast-to-Noise Ratio (CNR)

CNR measures the ease with which low-contrast objects may be differentiated from their surroundings. Objects with a higher CNR are more easily separated.

1.3.1.2. Spatial Resolution (R)

Frequently measured in line pairs per millimeter (lp/mm), spatial resolution (R) is the smallest separation between two high-contrast objects so that they appear as two separate objects.

1.3.1.3. The signal-to-noise ratio (SNR)

The suggestion states that the higher the SNR, the better the degraded image has been reconstructed to resemble the original image and the more effective the reconstructive algorithm. For the reliable detection of an object within a picture, SNR of 5 or higher is necessary.

Chapter (2)

Methodology, Theory, and Analysis

This chapter aims to provide comprehensive information on the medical aspects of X-ray image diagnosis and the various chest cases. It then goes deeply into AI and deep learning in order to start understanding the mathematical building blocks of neural networks with their various operations and algorithms.

2.1. Diagnosis of CXRs images

CXRs are a vital tool for doctors to diagnose a wide range of lung conditions. However, the interpretation of CXRs can be challenging for some healthcare professionals. In particular, lung opacities can be difficult to interpret because of the variable appearance of these opacities on a CXR. When doctors try to analyze a CXR image, they go through a number of steps to determine if the image shows the presence of a lung condition. The first step for a CXR image examination is to observe the costophrenic angle (CPA) which in the normal case is called "free" with air that appears black in the X-ray image, but if it is obliterated this means that there is a white color "some fluid" in it. And this can be divided into 2 types:

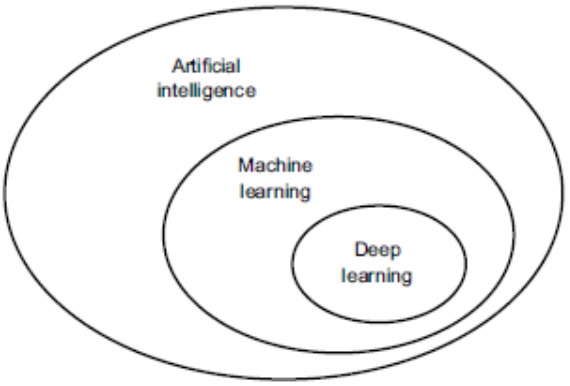
- The fluid is a sign of pleural effusion, when there is fluid only, the case can be diagnosed and categorized according to the position of the fluid and its effect on the heart and trachea position in the image as mild, moderate or massive.
- Hydropneumothorax is diagnosed when the image displays a mixture of fluid and air. Besides, the lung appears to be collapsed and it is divided into two halves; one jet-black and the other clear white, as a consequence of a fluid leak or rupture of the pleural membranes.

(a) **(b)** **(c)**

Fig. 2.2. Pleural effusion (a. mild, b. moderate, c. massive)

2.2. AI and deep learning

Initially, it is important to establish a precise and explicit understanding of the terms used when referring to AI. This involves defining AI, machine learning, and deep learning and exploring their interconnections.



2.2.1. Artificial Intelligence:

In brief, AI can be defined as the attempt to mechanize cognitive activities that are typically executed by humans. Therefore, AI constitutes a broad area that encompasses machine learning and deep learning, among other techniques that may not necessarily involve any learning. It is worth noting that prior to the 1980s, the majority of AI instructional materials did not make any reference to the concept of "learning".

2.2.2. Machine Learning:

Typically, the conventional method of enabling a computer to perform beneficial tasks involves a human programmer creating a set of instructions, or a computer program, that specifies how to process input data and generate relevant outputs.

In contrast, machine learning operates differently: it examines both the input data and the corresponding outcomes and determines what rules should be applied. Instead of being explicitly programmed, a machine learning system is trained. It is presented with numerous relevant examples for a specific task, and it identifies statistical patterns within these examples, which eventually allows the system to generate rules for automating the task. For instance, if someone desired to automate the tagging of their vacation photos, they could provide a machine-learning system with multiple samples of pictures that have already been tagged by people. The system would then learn statistical rules for linking particular images to particular tags.

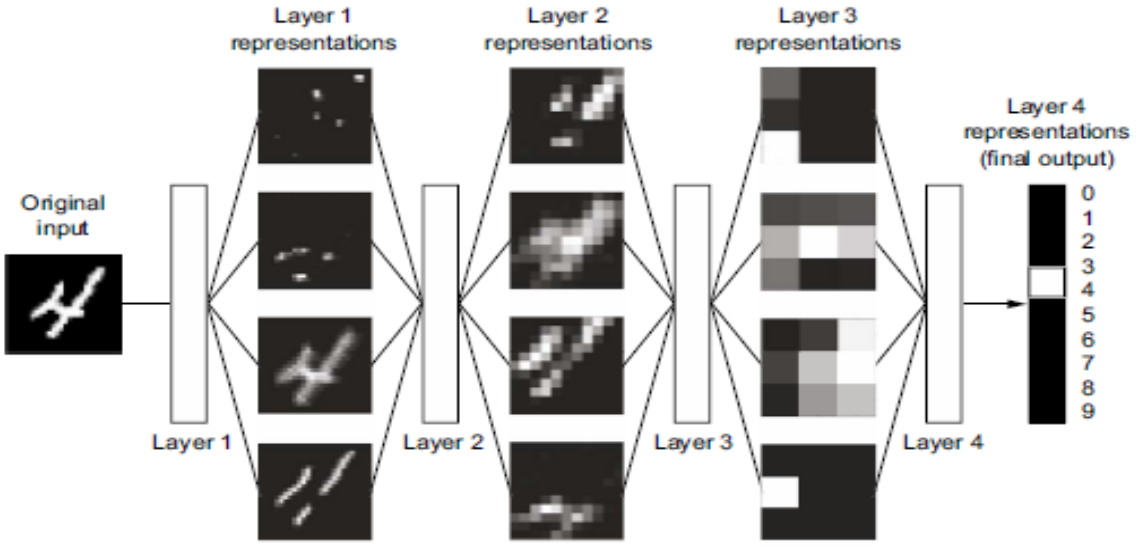
2.2.3. Deep Learning

Deep learning is a specific type of machine learning that focuses on learning a series of increasingly meaningful representations from data. The term "deep" in deep learning does not refer to a higher level of understanding achieved by the method but instead refers to the idea of learning successive layers of representations.

The number of layers that contribute to a model of the data is called the depth of the model. Other appropriate names for this field could have been layered representations learning or hierarchical representations learning. In modern deep learning, tens or even hundreds of successive layers of representations are typically involved, and they have all learned automatically from exposure to training data.

Conversely, other machine learning approaches tend to concentrate on learning only one or two layers of representations of the data; thus, they are sometimes referred to as shallow learning. Deep learning uses neural networks to learn layered representations, which are structured as literal layers stacked on top of each other.

Although the term "neural network" is inspired by neurobiology, deep learning models are not intended to be models of the brain.



2.3. Neural network building blocks in Mathematics

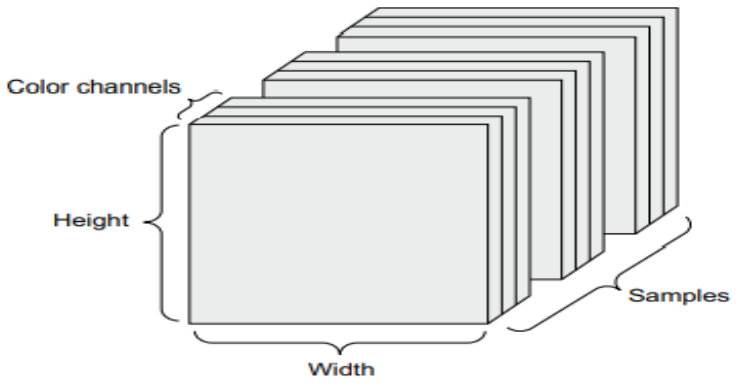
Tensors, tensor operations, differentiation, gradient descent, and other basic mathematical ideas must be understood in order to comprehend deep learning. Without getting into more complex technical details, the goal of this section is to explain these ideas. The section will specifically avoid using mathematical notation because it is unnecessary to introduce complex mathematical backgrounds in this work.

2.3.1. Neural network data Representations

A tensor is fundamentally a container for data, typically numerical data. It is therefore a container for numbers. Tensors are a generalization of matrices to any number of dimensions (it should be noted that a dimension in the context of tensors is frequently referred to as an axis).

2.3.2. Image data Representation

Height, breadth, and color depth are the three dimensions that most images have. Although rank-2 tensors could be used to store grayscale images because they only have a single-color channel, by convention image tensors are always rank-3 and only have a one-dimensional color channel for grayscale images. Thus, a tensor of shape (128, 256, 256, 1) could be used to store a batch of 128 256 256 grayscale images, and a tensor of shape (128, 256, 256, 3) could be used to store a batch of 128 256 256 color images.



2.3.3. Gradient-based optimization

As previously demonstrated in the preceding section, each neural layer in our basic model example modifies its input data in the manner shown in equation (2.1):

$$output = Relu(dot(input, W) + b) \quad \textbf{(2.1)}$$

The kernel and bias properties are referred to as the layer's weights or trainable parameters, respectively. The knowledge that the model discovered after being exposed to training data is contained in these weights. These weight matrices are initially initialized (also known as random initialization) using small random numbers. Naturally, there is no reason to believe that equation (2.1) will produce any useful representations when W and b are random.

Although meaningless, the resulting representations serve as a starting point. On the basis of a feedback signal, these weights will then be gradually adjusted. The learning that underlies machine learning is this gradual modification, sometimes referred to as training. This occurs within a system known as a training loop, which functions as follows. Follow these instructions repeatedly until the loss looks acceptable low:

- Draw a batch of training samples, **X**, and corresponding targets, **y_true**.
- Run the model on **X** (a step called the forward pass) to obtain predictions, **y_pred**.
- Compute the loss of the model on the batch, a measure of the mismatch between **y_pred** and **y_true**.
- Update all weights of the model in a way that slightly reduces the loss on this batch.

When forecasts, **y_pred**, and expected objectives, **y_true**, are closely matched, a model with a noticeably low loss on its training data will eventually be attained. The model has "learned" to map the right targets to the right inputs. It may appear to be magic from a distance, but when broken down into its simplest components, it is actually rather basic. Just writing I/O code in Step 1 seems simple enough. Steps 2 and 3 only involve the use of a few tensor operations, therefore they could be implemented using only the ideas covered in the preceding section. Updating the model's weights in step 4 is the challenging step.

2.3.4. Backpropagation Algorithm

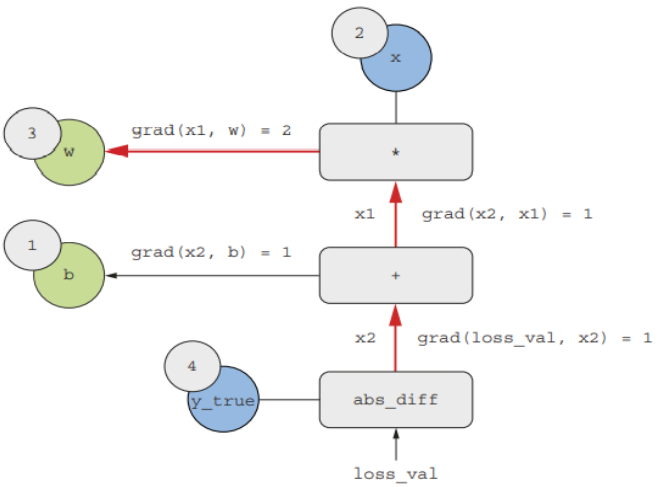
In practice, computing the gradient of complex functions can be a non-trivial task, and cannot be done easily in closed form. The Backpropagation algorithm is a widely used method for computing gradients of functions that are represented as a composition of simpler functions, such as in the case of neural networks. In the context of a two-layer model, the gradient of the loss with respect to the weights can be computed using the Backpropagation algorithm.

This algorithm computes the gradients by recursively applying the chain rule of calculus to propagate the gradient backward through the layers of the network. Specifically, it first computes the gradient of the loss with respect to the output of the last layer and then uses this to compute the gradient with respect to the weights in the last layer.

This gradient is then propagated backward to compute the gradient with respect to the weights in the preceding layer, and so on until the input layer is reached.

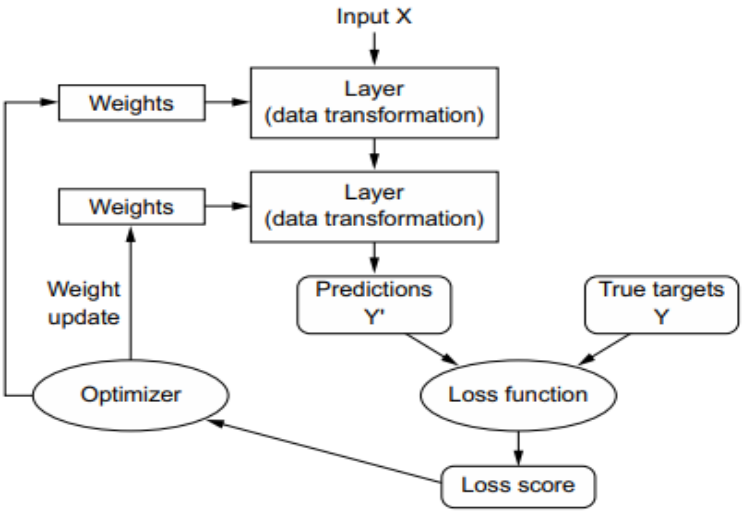
2.3.4.1. The chain rule

Backpropagation is a technique for quickly computing the gradient of arbitrary complicated combinations of these atomic operations by using the derivatives of basic operations (like addition, Relu, or tensor product). Importantly, a neural network is made up of numerous chained tensor operations that each have a straightforward, well-known derivative. Calculus informs us that the following identity, often known as the chain rule, can be used to derive such a chain of functions. According to the chain rule, you may get a node's derivative with respect to another node by multiplying the derivatives of each edge along the path connecting the two nodes.



2.3.5. Relationship between the network, layers, loss function, and optimizer

The model, composed of layers that are chained together, maps the input data to predictions. The loss function then compares these predictions to the targets, producing a loss value: a measure of how well the model's predictions match what was expected. The optimizer uses this loss value to update the model's weights.



2.4. Anatomy of a neural network

2.4.1. Layers Deployment

The layer is the basic unit of data in neural networks and is considered the basic building block of deep learning. A layer is a type of data processing unit that accepts one or more tensors as input and produces one or more tensors as output.

Some layers are stateless, but more often than not, layers have a state, which consists of the layer's weights (one or more tensors that were learned by stochastic gradient descent and collectively hold the knowledge of the network). Different tensor formats and forms of data processing call for different kinds of layers.

For instance, densely connected layers, also known as fully connected or dense layers (the Dense class in Keras), are frequently used to handle simple vector data that is stored in rank-2 tensors of shape (samples, features).

2.4.2. The “Compile” Step

Once the model architecture is defined, still three more things are needed to be chosen:

- **Loss function (objective function):** The quantity that will be minimized during training. It represents a measure of success for the task at hand.
- **Optimizer**: Determines how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent (SGD).
- **Metrics**: The measures of success monitored during training and validation, such as classification accuracy. Unlike the loss, training will not optimize directly for these metrics. As such, metrics don’t need to be differentiable.

2.4.3. Picking a loss function

It is crucial to match the right loss function to the right problem because your network will use any shortcut it can to reduce loss. If the objective does not fully correlate with success for the task at hand, the network may end up acting inadvertently or generating undesirable results.

"**Maximizing the average well-being of all living humans**" is a poor choice for the objective function of a foolish, omnipotent AI trained by SGD. Because the average well-being is unaffected by the number of humans left, this AI may decide to kill all humans except a small number in order to make its job easier and concentrate on the welfare of those who are still alive.

This might not have been meant to be! It must be taken into consideration that every constructed neural network will be equally ruthless in lowering its loss function, so the objective has to be picked carefully to avoid unintended consequences. Fortunately, there are easy rules to follow when selecting the right loss for common issues like classification, regression, and sequence prediction. For example, binary cross entropy is used for problems involving two classes, categorical cross entropy is used for problems involving several classes, and so on.

2.4.4. The Fit () method

Fit () follows compile (). The training loop itself is implemented by the fit () method. These are its main justifications:

- **The data (inputs and targets) to train on.** It will typically be passed either in the form of NumPy arrays or a TensorFlow Dataset object.
- **The number of epochs to train for:** how many times the training loop should iterate over the data passed.
- **The batch size to use within each epoch of mini-batch gradient descent:** the number of training examples considered to compute the gradients for one weight update step.

2.4.5. Monitoring loss and metrics on validation data

Creating models that perform well on the training data, which is simple, is not the aim of machine learning. The objective is to produce models that perform well overall, but especially when applied to data points that the model has never seen before.

A model may perform well on its training data, but it is not a guarantee that it will perform well on data that it has never seen before! It is feasible, for instance, that the model will just end up memorizing a mapping between training samples and their goals, which would be useless for the task of predicting targets for data the model has never seen before.

It is common practice to set aside a portion of the training data as validation data; the model will not be trained on this data, but it will be used to calculate a loss value and metrics value. This allows the monitoring of how the model performs on new data.

To distinguish it from the "**Training Loss**," the value of the loss on the validation data is referred to as the "**Validation Loss**". Considering that it is crucial to maintain rigorous separation between the training and validation sets of data because the latter is used to check whether the model's newly learned skills are applicable to new data.

Chapter (3)

Results & Discussion

Chapter (3)
Results & Discussion

In this chapter, the used data set, and the performance of the different AI models with it will be discussed. The results of the proposed model will be interpreted. Finally, the mobile application depending on the proposed model will be introduced.

3.1. The Dataset Used To Train The Model

For every experiment in this investigation, a set of predetermined parameters will be employed. Each model was specifically trained using a batch size of **20 samples** for a total of **50 epochs**.

7000 photos made up the dataset, which was divided into training, validation, and testing sets using a **70/15/15 ratio**. To reduce the cross-entropy loss function during training, the well-known stochastic gradient descent algorithm:

Adam optimizer was utilized. A number of trials were done to achieve a fair comparison of the performance of each model on the pneumonia detection task by utilizing fixed parameters. It is important to keep in mind that these settings might not be the best ones for all datasets or models. It has been demonstrated that choosing **50 epochs** for training is a typical deep learning practice and helps achieve good performance on a range of tasks.

20 samples were chosen as the batch size for every iteration to strike a balance between computational effectiveness and model stability. Smaller batch sizes may result in shorter training times and less memory use, but they may also provide less stable convergence and noisier gradients. The decision to use **7000 photos** for the dataset most likely reflects the size of the pneumonia detection dataset that is currently available.

Even though larger datasets occasionally result in better performance, obtaining or compiling a larger dataset may not always be feasible or practical.

It is standard procedure to divide training, validation, and test sets into **70/15/15 ratios** in order to assess the effectiveness of machine learning models.

3.2. Import All The Crucial Libraries

The following step involves applying several pre-processing techniques to the input photos. The goal of image pre-processing is to enhance the visual information included in each input image (by removing or reducing noise from the original input image, increasing contrast to enhance image quality, removing low- or high-frequency information … etc.).

Specific data preprocessing tasks can be carried out using the predefined Python libraries. The second step in machine learning's data preprocessing is importing all essential libraries. For this data preprocessing in machine learning, the three main Python libraries are [23]:

- **NumPy:** The foundational Python library for scientific computation. As a result, it is utilized to add any kind of mathematical procedure to the code. You can also incorporate sizable multidimensional arrays and matrices into your programming by using it.
- **Pandas:** is a top-notch open-source Python data analysis and manipulation library. It is frequently used to manage and import datasets. It includes Python data analysis tools that are high-performing and simple to use.
- **Matplotlib:** This Python 2D chart-plotting package can be used to create any kind of chart in Python. It can deliver publication-quality figures in a variety of hard copy formats and in cross-platform interactive settings.

Finally, using test photos and the trained model, the performance of the proposed method is assessed. Each experiment is run three times, and the average results are presented.

3.3. Testing Models

3.3.1. VGG16

The model has achieved state-of-the-art results on several benchmark datasets, including ImageNet, which contains over one million images. However, VGG16 also has some disadvantages. It is a very deep model with a large number of parameters, which can make it slow to train and computationally expensive. Additionally, VGG16 requires a large amount of training data to achieve good performance. The architecture of VGG16 is characterized by small convolutional filters (3×3) and a high number of filters per layer, which results in a large number of trainable parameters. The model consists of 16 layers, which are composed of convolutional layers, pooling layers, and fully connected layers.

3.3.2. VGG, Resnet50 and CNN

Transfer learning is a powerful technique for training deep neural networks using pre-trained models. In this project, it is applied with three popular models: VGG, ResNet50, and CNN. One of the advantages of using transfer learning with pre-trained models such as VGG, ResNet50, and CNN is that it reduces the amount of training data needed to achieve good results.

This is because the pre-trained models have already learned to extract meaningful features from images, which can be used as a starting point for training a new model on a smaller dataset. Another advantage is that these models have been extensively studied and fine-tuned, resulting in state-of-the-art performance on many image recognition benchmarks. However, one potential disadvantage of transfer learning is that it can lead to overfitting if the new dataset is too small or too different from the original dataset used to train the pre-trained model. In addition, the pre-trained models may not be optimized for the specific task at hand, so fine-tuning may be necessary to achieve the best performance.

The transfer learning model with VGG, Resnet50, and CNN was also implemented. This model achieved an accuracy of 91% with a low false negative rate that can be shown in the confusion matrix below in Fig. 3.1. for detecting pneumonia patients as normal cases of 25.

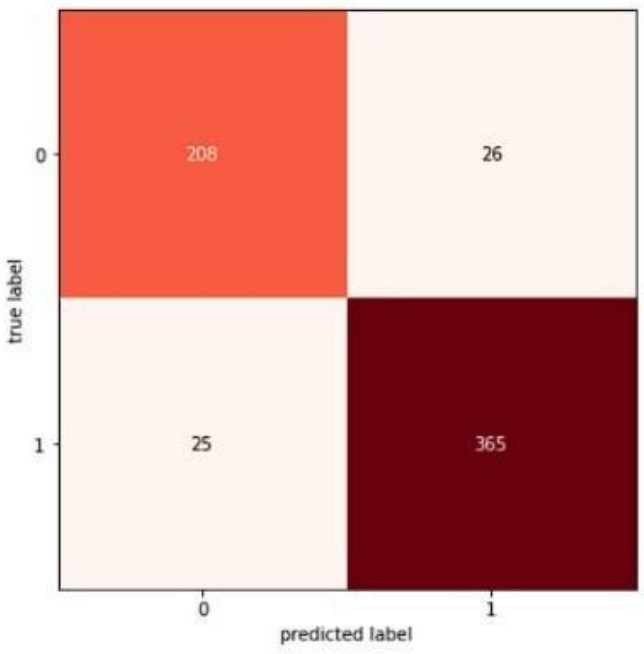


Fig. 3.1. Convolution Matrix of VGG, Resnet50 and CNN

3.3.3. Inception

While Inception has been shown to be effective for a variety of computer vision tasks, in our experiments it did not perform as well as the other models evaluated for the task of pneumonia detection. In particular, after running with the same dataset and parameters, it gave a lower accuracy of 63 % with a large number of false negatives, and longer training times were observed compared to other models, indicating that it can't be the most suitable choice for this specific application.

3.3.4. Sequential Application Programming Interface (API)

One of the main advantages of the Sequential API is its ease of use and implementation. The API is a good choice for simple image classification tasks and can be used to build a wide variety of models, including both shallow and deep networks. However, the Sequential API may not be suitable for more complex image classification tasks that require more advanced architectures and techniques. Additionally, the simplicity of the Sequential API can sometimes result in suboptimal performance compared to more complex models. The Sequential API is a simple way to build a deep learning model in Keras. The API allows you to stack layers in a linear fashion, with each layer feeding into the next. You can choose from a variety of different layer types, including convolutional layers, pooling layers, and fully connected layers. Sequential API was experimented with and achieved an overall accuracy of 92% in experiments.

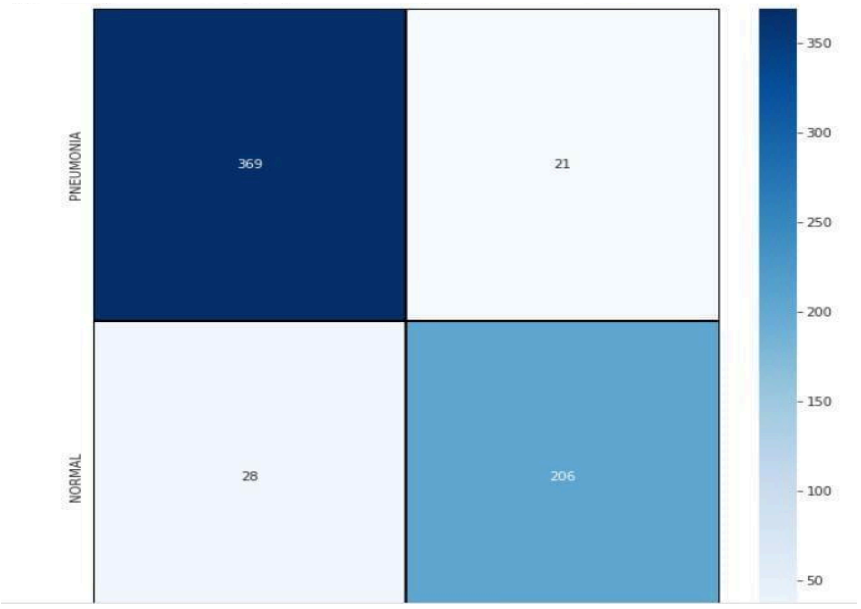


Fig. 3.2. Confusion Matrix of Sequential API (predicted number of cases to the true ones)

3.3.5. DenseNet

The DenseNet model is a convolutional neural network architecture that was introduced as an improvement over traditional deep neural networks.

One of the main advantages of DenseNet is its state-of-the-art performance on several benchmark datasets, including ImageNet. The model has achieved high accuracy with fewer parameters than other deep neural networks, which can make it more computationally efficient. Additionally, DenseNet can be fine-tuned for a wide variety of image classification tasks.

However, DenseNet can be more complex to understand and implement compared to simpler models like the Sequential API. Additionally, the dense connections between layers can result in a larger number of trainable parameters, which can make the model slower to train and computationally expensive.

The architecture of DenseNet is characterized by dense connections between layers, where each layer is connected to all subsequent layers in the network. This approach can help to alleviate the vanishing gradient problem, where gradients can become very small as they propagate through deep networks, by providing multiple paths for information to flow through the network. Several previous studies have used DenseNet for pneumonia detection, and the model has achieved good results with high accuracy and sensitivity.

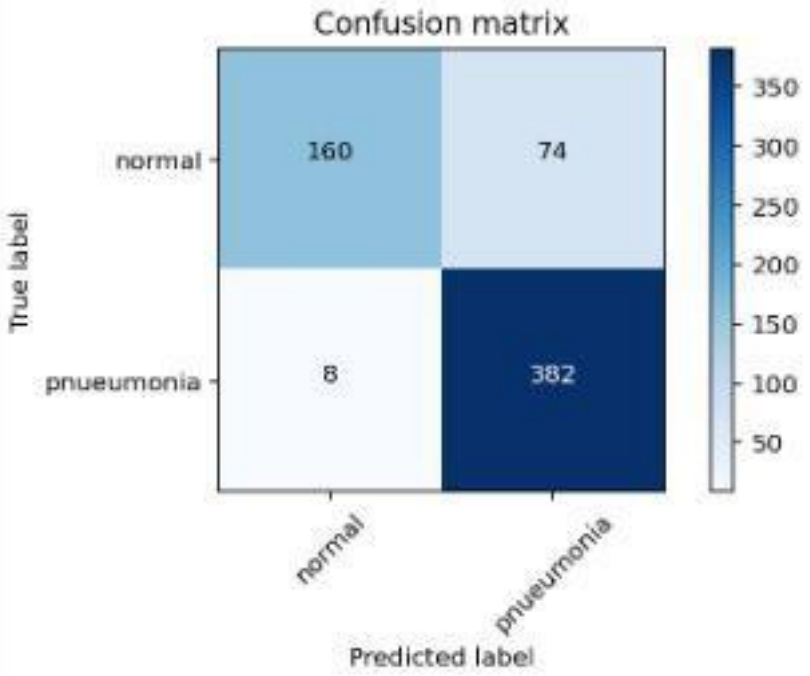


Fig. 3.3. Convolution Matrix of DenseNet

3.4. Model Testing And Results

In that study, The DenseNet model was used and trained on a larger dataset of 14000 images, The main parameters (batch size and epochs) were adjusted to make the model get the number of false negative cases as least as possible which can be known as the testing stage with a number of images of 2563 image and in each testing, results can show the cases prediction through the confusion matrix at each trial.

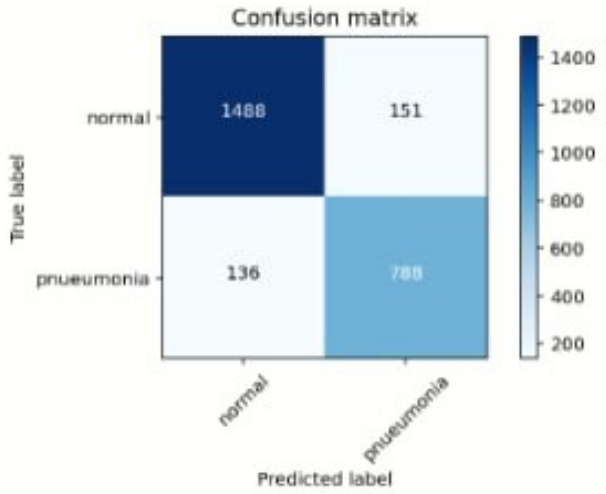


Fig. 3.4. Convolution Matrix of DenseNet at batch size of 16 with 5 epochs

Batch Size	No. of epochs	Accuracy	False Negatives
16	5	89%	136
10	10	87%	158
20	50	88%	125
23	50	90%	108
25	50	89%	102
28	50	90%	102
30	50	90%	136
27	55	89%	116
27	60	88%	112
27	65	88%	107
27	70	90%	124

Table 4.1. Comparison of different obtained results from convolution matrices

3.5. Developing A Website Application To Use The Proposed Model

First, the trained model was exported and saved in model format. These types of models can be saved in three different ways: as a Keras model, a saved model, or a collection of specific in-memory functions (TensorFlow). The h5 format is employed in the current study with the incredibly straightforward method model.save("filename.h5"). The SavedModel exports a MetaGraph, a data flow graph that includes a list of classes and classification signatures for prediction because the problem at hand is the challenge of picture classification.

Additionally, using the TFLite converter after that creates the TFLite model from these Keras models, which will be saved in the .tflite format. The TFLite file has finally been saved by writing it to the file system. A flat buffer of the .tflite format is utilized on mobile devices together with the backend, or Graphics Processing Unit (GPU). To obtain an output, this model can therefore be instantiated from the saved Keras model based on how the model is represented with the procedures for flattening the model for simple applications on mobile devices (as shown in Fig. 3.16).

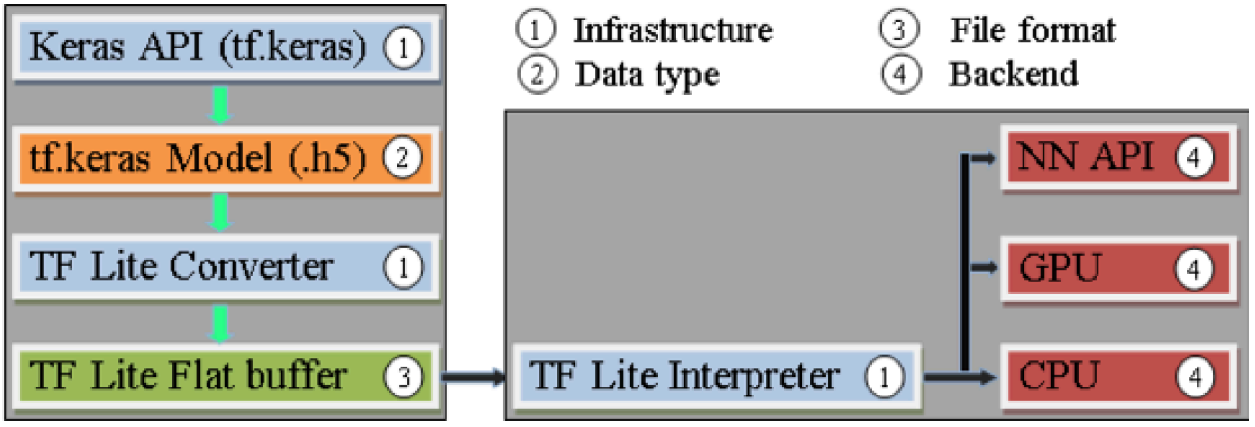


Fig. 3.16. Illustration of TensorFlow Lite conversion process [24].

3.5.1. Usage of Flutter

Due to the increased proliferation of smartphones, mobile applications are available for healthcare and estimating diagnosis in limited circumstances on smartphones.

Flutter, a cross-platform framework for real-time applications, has been developed. It offers a reliable solution with a single codebase for mobile (Android and iOS), web, and desktop apps. It also uses GPU delegates to reduce latency and increase power economy with the APK “Android Package Kit” for the distribution of apps on mobile devices that operate without an Internet connection.

3.5.2. Running/Deploying the Model on Mobile Device

Building a model is only a small part of what is needed, but the critical part is to produce a solution using machine learning or deep learning or to create a real-time application. The model must be constructed, converted, optimized, and saved using TFLite technology by carefully following the previous steps. The converted model has been included in the code of the project cross-platform Flutter mobile app. Cross-platform refers to creating an app with a single codebase that works on both iOS and Android. As the Flutter app is being developed, The following are the steps that had been used to develop an app and releasing it on mobile devices:

- Flutter SDK installation and configuration.
- Adding the TFLite file and the class label file to the app code's asset folder.
- To make the app operational, three crucial functions have to be loaded: the tflite model, taking a picture, and classifying the pictures which were added to the main.dart file.
- Use the TFLite interpreter to perform the inference and make the GPU delegate the backend to enhance the inference performance.
- Once the Shelf-Life application is completed, create the APK to install the app on mobile devices.

3.5.3. User Interface UI Design of the Application and How it Works

While developing the application, our main priority was to design a mobile app and a website that should be consistent, user-friendly, and easy to use and navigate, even for users who are not familiar with AI or machine learning. The UI design can also be visually appealing and engaging when using X-ray images to help users understand the symptoms of pneumonia and the importance of early diagnosis and treatment. So, the app consists of three main interfaces: a registration screen, a login screen (or to go as a guest), and AI-powered pneumonia detection X-Ray screen. The registration screen allows users to register as doctors or patients.

For doctors, they are being asked to provide their medical license number as the system provides them with a percentage of pneumonia cases to help them identify potential cases more easily. And for the patients, they are being advised to visit a doctor if the X-ray indicated possible infection by pneumonia. The login screen allows users to log in with Google and reset their password using a code. This ensures that only authorized users can access the system and protects ‘patients’ data.

Chapter (4)

Conclusion

Conclusion

In this project, we developed a pneumonia detection model which can be utilized as a second opinion in hospitals or other healthcare entities. By providing a reliable automated assessment of CXR images, the model can assist healthcare professionals in improving the accuracy and speed of their diagnoses according to the registered user.

If the user is a doctor, the model can assist him/her by reassuring his detection opinion. Meanwhile, if the user is a patient can assist him/her by giving recommendations for the coming steps after this examination.

Our main priority was to design a mobile application and website that should be consistent, user-friendly, and easy to use and navigate, even for users who are not familiar with AI or machine learning.

Several deep-learning models were tested for pneumonia detection using CXRs images. Transfer learning was used with VGG, ResNet50, CNN, as well as InceptionV3 and DenseNet. Their performance was evaluated by using different dataset sizes and training configurations against the proposed criteria (False negative cases and accuracy).

After careful analysis and comparisons between the results of the suggested deep learning models, DenseNet was found to be the most effective model, achieving an accuracy of 90%. It also showed a smaller number of false negatives, which is crucial in medical image detection. Moreover, the dataset size was increased from 7,000 to 14,000 images, the batch size was adjusted to 27 and the number of epochs was set to 50. This had a significant impact on model performance, leading to an accuracy of 90% with 100 false negative cases.

In conclusion, our study demonstrated the effectiveness of deep learning models in pneumonia detection and provides insights into the advantages and limitations of different models and training configurations.

Further research can be conducted to improve model performance and explore other potential applications in medical image diagnosis.