Using Convolutional Neural Network for the diagnosis of Covid-19 related or non-related pneumonia from chest X-ray images

Hasanberk Cakmak[†], Bedriye Gulce Kaya[‡]

***Abstract*—Chest X-ray is a medical imaging technique that plays an important role in the diagnosis of lung diseases such as pneumonia which is an inflammatory condition caused by infection with viruses or bacteria and also COVID-19 which is a contagious disease caused by the virus SARS-CoV-2 that quickly spread worldwide, resulting in the COVID-19 pandemic starting from December 2019. In this study, we used different custom CNN models with increasing complexity for the recognition and classification of the X-ray images from the dataset. The dataset containes pneumonia, Covid-19 positive, and normal lung images to help us in developing the models with a higher accuracy rate.**

***Index Terms*—Convolutional Neural Network, Machine Learning, Image Processing, Deep Learning, COVID-19, Pneumonia**

I. Introduction

In recent years, deep learning algorithms, particularly Convolutional Neural Networks (CNNs), have revolutionized the field of medical imaging analysis. CNN-based models have demonstrated remarkable success in accurately classifying diseases by extracting certain features from the image to help physicians with diagnosis.

With the emergence of the COVID-19 pandemic and the prevalence of pneumonia cases, radiological imaging tools such as X-ray imaging has played a crucial role in the visualization of the internal structures of the human body. Additionally, CNN-based models offer a valuable second opinion and can assist healthcare professionals in challenging cases. They can serve as decision support tools, providing insights and highlighting potential abnormalities that may have been overlooked. By leveraging the power of CNN models, healthcare providers can enhance their diagnostic accuracy and reduce the likelihood of misinterpretation or missed diagnoses.

One of the primary advantages of CNN-based models for lung disease classification is their ability to learn and extract complex features automatically from X-ray images. Traditional image analysis methods often relied on handcrafted features, which were time-consuming and limited in their ability to capture the subtle patterns and nuances present in medical images. CNNs, on the other hand, can automatically learn hierarchical representations of images, enabling them to identify relevant patterns and features associated with specific lung diseases.

In this study, we will discuss the construction of multiple custom Convolutional Neural Networks or CNNs including attention layer and eventually a custom Densenet structure. Custom CNN with different optimizers are compared for their efficiencies with accuracy and loss results.

This report is structured as follows. In Section II we describe the related CNN models used in this field, the system and data models are respectively presented in Sections III and IV. The proposed signal processing technique is detailed in Section V and its performance evaluation is carried out in Section VI. Concluding remarks are provided in Section VII.

II. Related Work

Following the emergence of the novel coronavirus disease (COVID-19) in December 2019, the utilization of Deep Learning models for diagnosing airway disorder diseases gained significant traction, becoming a prevalent method in the medical field. This global crisis prompted a surge in studies and the development of various deep learning modalities.

DenseNet, short for Dense Convolutional Network, is a deep learning architecture proposed by Huang et al. in 2017. It is a type of convolutional neural network (CNN) that aims to address some of the limitations of traditional CNN architectures [1].

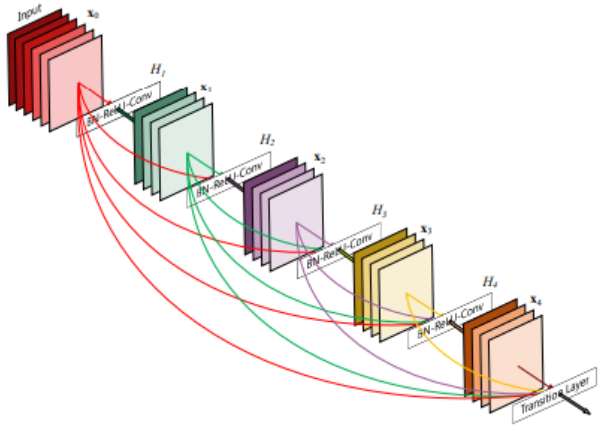

Fig. 1: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input [1].

[†]Department of Information Engineering, University of Padova, email: hasanberk.cakmak@studenti.unipd.it

[‡]Department of Information Engineering, University of Padova, email: bedriyegulce.kaya@studenti.unipd.it

Ovy Rochmawanti and Fitri Utaminingrum used DenseNet-121 in their study to diagnose Tuberculosis, Pneumonia, Cardiomegaly, and COVID-19 with the highest accuracy of 0,892, 0,904, 0,898, and 0,986, respectively [2].

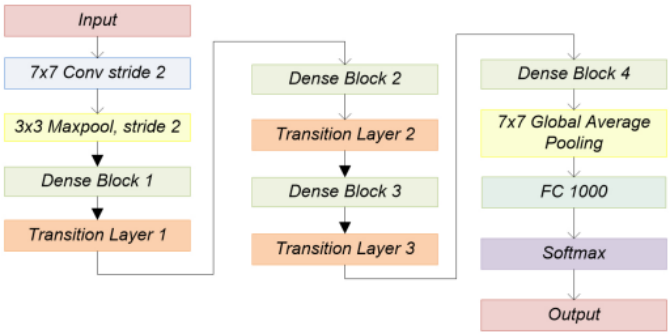

Fig. 2: Original Architectures of DenseNet-121 [2].

Ibrahim, A.U., Ozsoz, M., Serte, S., et al. used the AlexNet model as a DL model which utilizes rectified linear unit (ReLu) in place of the Sigmoid function which is used in traditional neural networks. The model achieved 84% accuracy in the prediction of Covid-19 Pneumonia [3].

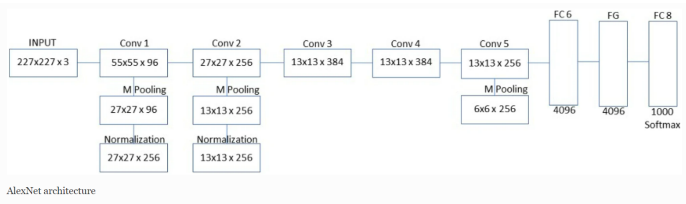

Fig. 3: AlexNet architecture [3]

III. Processing Pipeline

Over the course of this paper, different models were constructed from the ground up using Tensorflow-Keras. These models are as follows:

- **Model 1:** A custom CNN model with regular sequential layers
- **Model 2:**A custom CNN model with regular sequential layers
- **Model 3:** A custom CNN model with Attention layer implantation
- **Model 4:** A pre-trained Dense121 model for comparison
- **Model 5:** A custom CNN with DenseNet architecture

Before establishing any models to feed the data, a data separation and pre-processing phase took place to ensure the same and optimal conditions for each model since the condition of the data and the size of train, test, and validation datasets plays a crucial role in machine learning and data analytics.

Our specific path in this project is to initially construct a custom CNN which archives high accuracy by manipulating hyperparameters, optimizers, and layer architecture utilizing the sequential design. After achieving relatively high performance on a custom CNN with general layers adding an attention layer into the same CNN without changing hyperparameters to compare performance and how an attention layer affects different aspects of the performance in terms of training loss, validation loss, training accuracy, and validation categorical accuracy. The same has been done by implementing a DenseNet architecture to custom CNN to observe changes in the performance for the classification of chest x-rays.

Firstly a custom deep Convolutional Neural Network (CNN) architecture suitable for multi-class classification problems. It is not based on a pre-trained or pre-existing model, the model is designed by manually stacking multiple layers, including convolutional layers, batch normalisation layers, max-pooling layers, dropout layers, and dense layers, in a specific sequence to form a deep neural network. The choice of layer types, their configurations, and the number of layers are constructed to meet the requirements of the classification problem at hand.

While the construction of custom CNN (Model 1) the importance of the selection of optimization algorithms arised. Two popular options of Adam and RMSprop optimizer algorithms were considered the difference between them being the use of momentum and the learning rate adaptation. We wanted to experiment with both optimizers in use to determine the different characteristics firsthand. Hence a secondary custom CNN model was trained with the same hyperparameters and the same number of epochs at hand with the only difference being the change of optimizer from RMSprop to Adam.

After the training and evaluation were completed with custom CNN models an Attention layer was established to be integrated into the model. The attention layers are inspired by the human idea of attention mostly known for their performance in NLP applications. When used on training a custom CNN model they improve the models ability to focus on more relevant features. By incorporating attention mechanisms the model can assign varying levels of importance to different spatial regions, enabling it to prioritize features.

The same approach has been followed for the introduction of DenseNet architecture. A DenseNet is known for its efficiency and strong performance on image classification tasks. The important point here is that even though a Dense121 pre-trained model is present in the study, referenced Model 5 is a custom-made from-scratch Densenet model. Every model has been trained with the same dataset and appropriate epoch times to ensure a valid comparison.

IV. Signals and Features

A. Gathering and Partition of the Dataset

The dataset provided to us has 3 folders as Covid, Pneumonia, and Normal, and metadata.csv which contain chest X-ray posteroanterior (PA) images. X-ray samples of COVID-19 were collected from the following websites: GitHub, Radiopaedia, The Cancer Imaging Archive (TCIA), and the Italian Society of Radiology (SIRM). Then, instead of data being independently augmented, a dataset containing 912

already augmented images was collected from Mendeley. From the Kaggle repository and NIH dataset, 1525 images of pneumonia cases and 1525 X-ray images of normal cases were collected. 4575 images were used in this experiment, which has 1525 samples for each case. [4]

After acquiring the dataset, we split the 4575 images into 3 parts: training, testing, and validation datasets. The training dataset is 60% of the whole data and the testing, and the validation data are 20% and 20% divided. We used 2745 images as training data, 915 for testing, and 915 for validation.

B. Pre-processing the Dataset

After the partitioning of the dataset, a Data Augmentation phase took place. Before training the models three data generators were created for train, validation, and test. For this Keras ImageDataGenerator is used. Every image is scaled to 1/255 as a common normalization to image pixel values to improve convergence during training. Randomly applied rotations to images in order to help with robustness. Also shearing, horizontal flip, width, and height shiftings were applied. In our application we experimented with different batch sizes in order to optimize the system in the end we decided on 128 as the batch size. The class_mode is set to 'categorical' as the data is expected to be organized in subdirectories by class, and the labels will be one-hot encoded categorical values. The shuffle=True ensures that the data is shuffled after each epoch to further enhance training robustness.

V. Learning Framework

After the dataset is partitioned and preprocessed the training is initialized. The structure of each model is as follows:
Custom CNN architecture: A custom CNN architecture consists of several layers with each of them in charge of a specific operation. The model is created as a sequential model which means the layers are sequentially stacked. This model will be the fundamental structure of all the other models mentioned in this paper. The Custom Convolutional Neural Network of **Model 1** is defined as:

- **Input layer:** The initial layer to receive input. Images are resized as 128x128 pixels in pre-processing with the three color channels of RGB. As an activation function Rectified Linear Unit activation function (ReLU) will be applied. It is selected due to the simplicity and efficiency it provides. ReLU will replace all values that are negative with zero at the tensor. The 'padding' parameter is set to 'same', which means zero-padding will be applied to the input so that the output has the same spatial dimensions as the input.
- **Convolutional Layers:** In the architecture, there are multiple and repeating convolutional layers. The first convolutional layer has 32 filters with a 3x3 size. They are small windows that slide through the image matrix to detect local patterns and features. Over the course of the model, the filter size got systematically increased to capture more complex and higher-level features from the data. As the tensor moves to deeper layers the filter numbers have been increased from 32 to 64, 128, 256, 512 and finally 1024 in convolutional layers, the filter size hasn't been changed from 3x3. Every convolutional layer with a specific number of filters has been used twice and back to back with only a normalization between them.
- **Batch normalization:** Batch normalization is a method used to normalize the tensor and stabilize the process. It is applied after each convolutional layer until fully connected layers to achieve more efficient and regularised training.
- **Max pooling:**Max pooling is used to reduce the spatial dimensions of the feature maps. The window size is selected to be 2x2 which will reduce the size of the feature map by half. It is added after every two convolutional layers.
- **Dropout layers:** In order to prevent overfitting some dropout layers are introduced. A dropout layer with a dropout rate of 0.5 will randomly set half of the units to zero which prevents overfitting. However, to not increase computational complexity unnecessarily it is not used after every convolutional block.
- **Fully connected layers:** When the tensor reaches the fully connected layer it is flattened by tf.keras.layers.Flatten() layer means the conversion of a 2D feature map into a 1D vector by multiplying the dimensions. But since the model is already reduced to 1D after the last max pooling the Flatten layer is redundant. In this fully connected layer (tf.keras.layers.Dense(2048, activation='relu')), each neuron is connected to every neuron in the previous layer, it has 2048 neurons in this case, meaning it will have 2048 weights to learn from the flattened input data. After extracting meaningful patterns the output of the fully connected layer is fed into the output layer where the problem's requirement will be satisfied.
- **Output layer:** Since the problem of the project is classifying Covid, Pneumonia, and healthy Chest Xray Data, the final neuron numbers are set to 3 for multi-class classification. The softmax activation function is used to convert the final output into a probability score for 3 neurons. As softmax converts the weights into probabilities for these three classes.
- **Model compile:** The problem being a multi-class classification problem, the categorical cross-entropy loss function is used to compile the model with the RMSprop optimizer function. Since the model is constructed from the ground up it is without any pre-trained weights usage thus a considerable amount of epochs are necessary. To prevent overfitting an epoch number of 30 is selected for the model.

As explained in the Processing Pipeline part, on Model 2 the optimizer function has been changed from RMSpromp to Adam. Apart from that the structure remained the same with multiple convolutional blocks and a fully connected layer at

the end.

Fig. 4: Custom CNN layered image.

We wanted to experiment with more complex architectures to study how the additional complex layers or blocks affect the overall performance. To do so **Model 3** is introduced with an attention layer implantation between the last dropout layer and the fully connected layer of the initial custom CNN architecture constructed at **Model 1**.

For the attention layer, a combination of Channel Attention and Spatial Attention mechanisms has been used. In order to implement an attention layer a class called AttentionLayer is created with build and call methods:

- **Build:** This method is called when the layer is initiated. It extracts the number of filters from the last dimension of the input shape. Through which initializes avg_pool, max_pool, and shared_layers. They will be used in the call method. The build method also introduces a sigmoid activation layer that puts weights between a value of 1 and 0.
- **Call:** Call method is the actual computational method of the attention mechanism implemented. It is called when the layer is applied to an input tensor. It follows the below procedure:

1) Applies the ”avg_pool” to input to result in a spatial average of each channel
2) Take the results through two ”shared_layers” dense layers
3) Applies the ”max_pool” to input to result in the spatial maximum of each channel
4) The result will be reshaped to (1,1, self. filters)
5) Passes the reshaped results through the same dense layers for feature transformation
6) Adds the transformed average pooling and max pooling results element-wise.
7) Combined outputs will pass through ”attention_layer” to get the weights of each channel at hand which will represent their importance of them relative to others.
8) Original inputs will be multiplied with the attention weights of the previous step element-wise to obtain a final output of the layer.

When the AttentionLayer is executed it will point out the important channels while suppressing others. With the implantation of AttentionLayer, the Model 3 was expected to have better results but it wasn't the case. This will be discussed in the Results chapter.

Fig. 5: Custom CNN with AttentionLayer visualized

For the final model a more complex architecture of Densenet using Tensorflow/Keras is implemented as Model 5. The Densenet architecture is known for its efficiency and dense connections between layers. Initially, the hyper-parameters of input shape, number of classes, filter numbers, kernel size, padding, and number of layers have been adjusted accordingly to correspond to the custom CNN architecture at hand. In order to implement a DenseNet structure three blocks are created:

1) **”conv_block”:** This block is created to define the fundamental building block of a CNN, convolutional layers. It applies the layer with a given filter number and kernel size. To be able to compare the results with previous models these parameters and activation function, which is ReLU, were chosen accordingly. After the convolutional layer batch normalizations were added as well. The output of this block is the feature maps with the number of filters.
2) **”dense_block”:**A dense block represents a group of stacked convolutional blocks. Input tensor is taken and repeatedly applied to conv_block. This stacking of convolutional blocks with shared parameters enhances feature reuse and benefits the network in terms of learning more useful feature representations. The output of this block is the tensor concatenated feature maps of all conv_blocks inside the dense_block it has been. Concatenation allows for building a richer representation of input for the network.
3) **”transition_block”:**At this function combination of 1x1 convolution and average pooling is applied to reduce the spatial dimensions of the feature map. At the same time reduces the number of filters by half by (int(x.shape[-1]) // 2). After the convolutional layer, it applies average pooling with a pool size of 2x2 which downsamples the spatial dimensions. At the output of this block, we have reduced spatial dimensions, and half of the filters are dropped.

The working process of this model is as follows:

```
Input image  size                                                    (128.128.3)  =>
Convolutional Block (filters=32, ReLU, kernel size = 3x3) + BNorm    (128, 128, 32) =>
Dense Block (num_layers=2, growth_rate=32)                           (128, 128, 32)  =>
Transition Block                                                        (64, 64, 16)=>
Dense Block (num_layers=2, growth_rate=64)                               (64, 64, 64)=>
.........                                  feature maps size   =      (16, 16, 256) =>
GlobalAveragePooling2D =>
Fully Connected Layers:
A dense layer with 2048 units and ReLU   =>   The final dense layer with 3 units and softmax
```

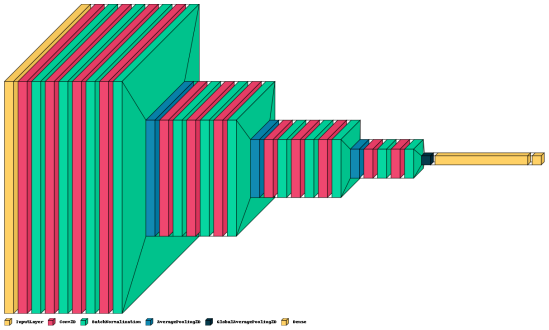

Fig. 6: Custom DenseNet layer visualized

VI. Results

In this part, the collection of all the results obtained from the custom CNN models is presented. Accuracy and loss graphs of the models are shown in the figures respectively: Custom CNN with RMSprop, Custom CNN with Adam Optimizer, Custom CNN with Attention Layer, and Custom DenseNet. As represented in the graphs the result of the first comparison of RMSprop and Adam optimizers has been very close. By looking at the loss graphs RMSprop shows more loss than Adam but has a more consistent accuracy as the model2 fluctuates a lot. By looking at the metrics Model2 has a slight edge over Model1 as a result we continued the training of the other models with Adam optimizers. The results of Model 3 and Model 4 created some confusion since we expected the addition of DenseNet architecture and attention layer to improve overall performance. We observed a slight decrease in Training loss and a slight increase in Training accuracy between Model 2 and Model 5, both validation accuracy and test accuracy suffered. We suspect the implementation of hyperparameters or the number of epochs might be insufficient for these models to thrive The same case can be seen for Model 3 which has the AttentionLayer. However, when given the choice of a slight decrease in Training loss and a slight increase in Training accuracy Model 3 or Model 5 can be chosen.

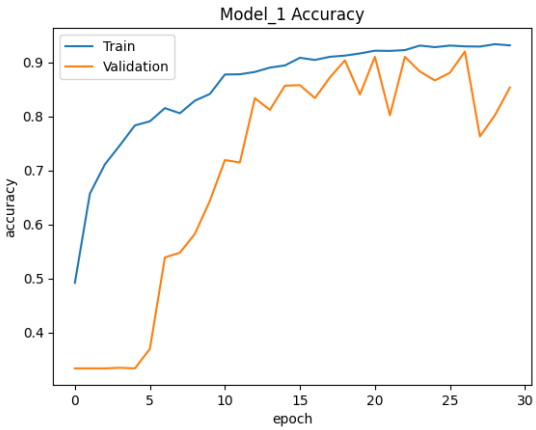

Fig. 7: Custom CNN with RMSprop accuracy graph

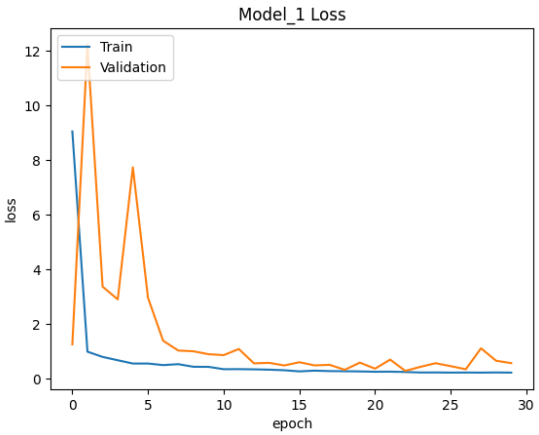

Fig. 8: Custom CNN with RMSprop loss graph

Evaluation Metrics for Custom CNN with RMSprop

- **Training Loss:** 0.2430
- **Training Accuracy:** 0.9275
- **Validation Loss:** 0.5683
- **Validation Accuracy:** 0.8535
- **Test Loss:** 0.5041
- **Test Accuracy:** 0.8579

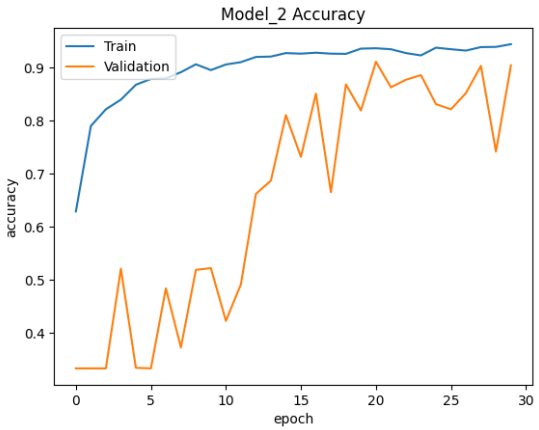

Fig. 9: Custom CNN with Adam Optimizer accuracy graph

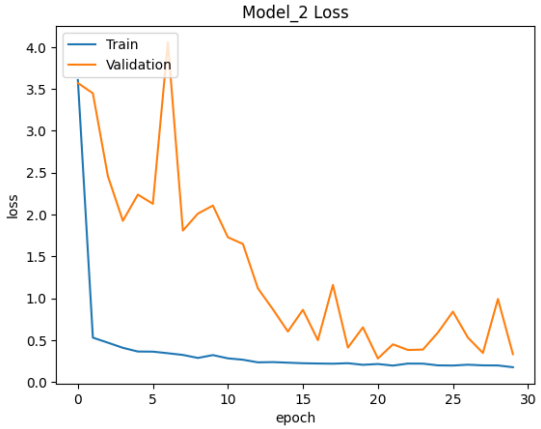

Fig. 10: Custom CNN with Adam Optimizer loss graph

Evaluation Metrics for Custom CNN with Adam Optimizer

- **Training Loss:** 0.2169
- **Training Accuracy:** 0.9271
- **Validation Loss:** 0.3321
- **Validation Accuracy:** 0.9049
- **Test Loss:** 0.3248
- **Test Accuracy:** 0.9038

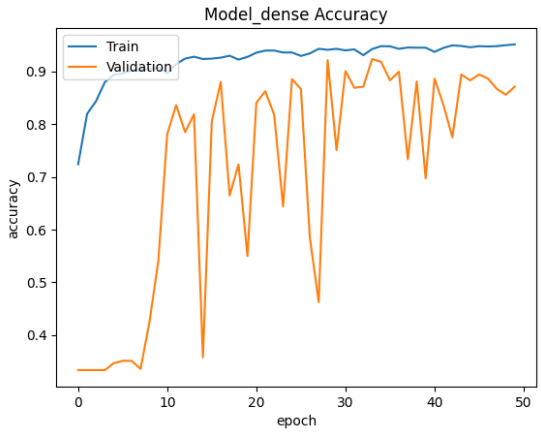

Fig. 11: Custom DenseNet accuracy graph

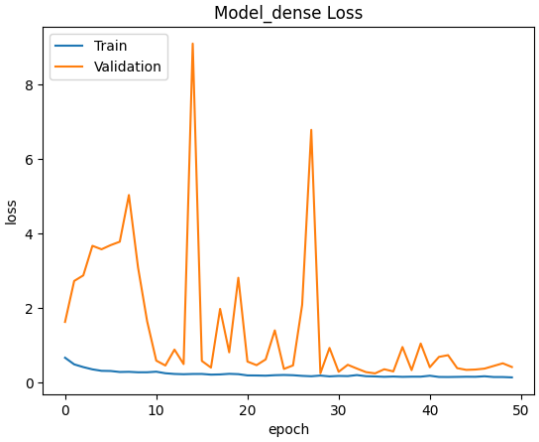

Fig. 12: Custom DenseNet loss graph

Evaluation Metrics for Custom DenseNet

- **Training Loss:** 0.1970
- **Training Accuracy:** 0.9322
- **Validation Loss:** 0.4159
- **Validation Accuracy:** 0.8710
- **Test Loss:** 0.4112
- **Test Accuracy:** 0.8841

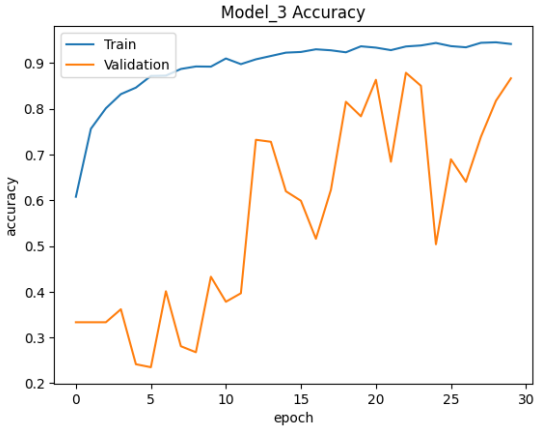

Fig. 13: Custom CNN with Attention Layer accuracy graph

Evaluation Metrics for Custom CNN with Attention Layer

- **Training Loss:** 0.2764
- **Training Accuracy:** 0.9071
- **Validation Loss:** 0.4288
- **Validation Accuracy:** 0.8666
- **Test Loss:** 0.4560
- **Test Accuracy:** 0.8513

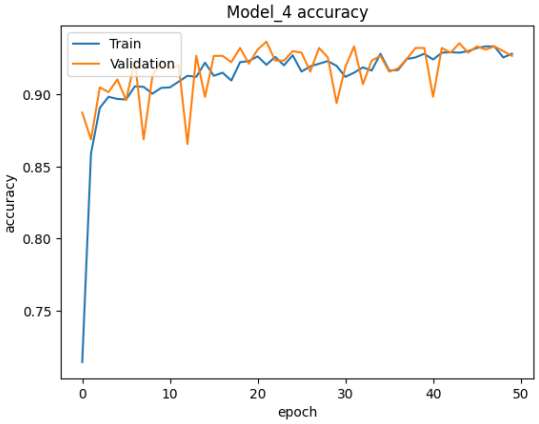

Fig. 14: DenseNet-121 accuracy graph

Evaluation Metrics for DenseNet-121

- **Training Loss:** 0.1460
- **Training Accuracy:** 0.9446
- **Validation Loss:** 0.2462
- **Validation Accuracy:** 0.9267
- **Test Loss:** 0.2433
- **Test Accuracy:** 0.9147

VII. Concluding Remarks

In conclusion, our aim was to develop the best CNN-based model for the detection of lung diseases which are COVID-19 and pneumonia. We used DenseNet-121, Adam Optimizer, Custom RMS, and Attention Layer models and compared the results by looking at their validation, test, and training accuracy and loss rates. The accuracy rates we obtained from the datasets varied between 0.94 and 0.85. For future studies, the amount of data that can be obtained will be more and this will help us to have more accurate and reliable solutions in medical diagnosis.

References

[1] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[2] O. Rochmawanti and F. Utaminingrum, "Chest x-ray image to classify lung diseases in different resolution size using densenet-121 architectures," *Association for Computing Machinery*, 2021.

[3] A. U. Ibrahim, M. Ozsoz, S. Serte, F. Al-Turjman, and P. S. Yakoi, "Pneumonia classification using deep learning from chest x-ray images during covid-19," *Cognitive Computation*, 2021.

[4] Z. Asraf, Amanullah; Islam, "Covid19, pneumonia and normal chest x-ray pa dataset," *Mendeley Data, V1*, 2021.