LUNG DISEASE DETECTION AND LOCALIZATION OF CXR IMAGES USING DEEP LEARNING

Narayana Darapaneni * Anwesh Reddy Sudha B G Sourabh Kumar Sumanth Babu S
Ganesh Pradeep P V Sourabh Jain Siddhartha Pandey Mathan Rajeshwaran

Contents

*Corresponding author

ABSTRACT

The COVID-19 crisis has caused millions of casualties worldwide and posed challenges for doctors and radiologists to differentiate it from other lung diseases due to similar symptoms. Chest X-ray is a reliable method to detect lung infections such as COVID-19, and computer-aided diagnosis (CAD) systems can assist in the early identification of Chest diseases. Previous works in this research area have used complex models, which require significant time and resources compared to the proposed models in this study. We implemented different CNN popular architectures over a public chest x-ray dataset such as VGG16, InceptionV3, XceptionNet, AlexNet, MobileNetV2, EfficientNetV2M, EfficientNetB5, ResNet50, ResNet101, ResNet152V2, DenseNet121, DenseNet169, DenseNet201, and ensemble techniques. EfficientNetB5 with optimization has shown excellent results in classifying COVID-19 pneumonia, viral pneumonia, and lung opacity with a whopping accuracy of 99%. Hence the proposed model is recommended for clinical diagnosis and triaging chest X-ray images as our model offers efficient ways to show the presence of chest disease using Grad CAM techniques.

1 Introduction and Literature Survey

Following is the literature survey carried out to understand the other works published in the area related to this research. The below survey discusses several concepts and algorithms other researches used in their research.

[1]This paper presents CheXNet, an algorithm whose results surpasses practicing radiologists in detecting pneumonia from chest X-rays. CheXNet is a Dense Convolutional Network consisting of 121 layers that was trained on ChestX-ray14, a publicly available dataset containing 112,120 chest X-ray images with 14 diseases. The performance of CheXNet was compared to that of four radiologists who annotated a test set. CheXNet takes a chest X-ray as input and predicts the probability of a pathology, with the heatmap indicating the localized area of the pathology. The Adam optimizer with hyperparameters 0.9 beta1 and 0.999 beta2 was used during training, with a mini-batch size of 16 and an initial learning rate of 0.001. Input images were scaled down to 224*224, normalized based on mean and standard deviation, and augmented with random horizontal flipping before being fed to DenseNet121. The F1 metric shows that CheXNet performs better than the average radiologist F1-score. CheXNet was also extended to detect all 14 diseases in ChestX-ray14 and achieved state-of-the-art results. When given a chest X-ray image, CheXNet correctly identifies pneumonia along with a heatmap indicating the areas of the image that are most indicative of the pathology.

Person & Model	F1 Score (95% CI)
Radiologist 1	0.383 (0.309, 0.453)
Radiologist 2	0.356 (0.282, 0.428)
Radiologist 3	0.365 (0.291, 0.435)
Radiologist 4	0.442 (0.390, 0.492)
Radiologist Avg.	0.387 (0.330, 0.442)
CheXNet	0.435 (0.387, 0.481)

When compared to previous research on all 14 pathologies in the ChestX-ray14 dataset, CheXNet exhibited strong performance. Specifically, in detecting four classes - Mass, Nodule, Pneumonia, and Emphysema - CheXNet outperformed the previous state-of-the-art results by a margin of more than 0.05 AUC ROC.

[2]This study utilized a deep learning method using transfer learning to classify lung diseases from Chest X-ray images. Two different datasets were used, the U.S. National Institutes of Health NIH dataset with three labelled classes $pneumonia, pneumothorax, normal$ and the Cheonan Soonchunhyang University Hospital SCH dataset with four labeled classes $pneumonia, pneumothorax, tuberculosis, normal$. The image sizes ranged from 1024*1024 to 3000*3000, but all images were scaled down to 600*600 and converted to grayscale with three channels 600*600*3. Data augmentation was performed to increase the number of samples. EfficientNetv2 was used as the base model and fine-tuned by adding new layers on top of it. EfficientNet v2 performed very well on the ImageNet dataset. But the ImageNet model can classify up to 1000 classes. However in the current scenario, since the number of classes are 3 in case of NIH dataset & 4 in case of SCH dataset, authors defined new layers on top of the base model and implemented the deep learning model. The model which is trained from scratch of all the layers performed better compared to the model which trained only newly added layers. New layers added are Global average pooling layer, Dropout layer on top of the EfficientNetv2-m. In this paper, a lookahead optimizer was used. The number of epochs it is trained on is 25. Model performance is given in the below table

The testing accuracy of normal, pneumonia, pneumothorax, and tuberculosis classes on SCH dataset was 63.60%, 82.30%, 82.80%, and 89.90%, respectively.

Table 1: EfficientNet v2-M Model Performance

Dataset	Classes	Validation loss	Accuracy	Sensitivity	Specificity
US NIH dataset	Normal,Pneumonia & Pneumothorax	0.6933	82.15%	81.40%	91.65%
SCH data set	Normal, Pneumonia, Pneumothorax & TB	0.7658	82.20%	81.40%	94.48%

[3]This study explores various CNN-based approaches for classifying lung diseases from Chest X-ray images in the ChestX-ray14 dataset. The authors used 5-fold resampling on the dataset, where each split allocated 70% for training, 10% for validation, and 20% for testing. Input images were center-cropped before being fed into the model. The authors started with ResNet-50 as the original architecture and experimented with multiple ResNet depths such as ResNet-101 and ResNet-38, where each ResNet block consisted of 3 sets of Conv layer+Batch Normalization+Relu Activation function. The multilabel classification function used for classifying multiple lung diseases was Sigmoid. Multiple models were tried, including off the shelf (OFS) based on transfer learning and fine-tuned (FT) in which one or more layers were trained from scratch. In both cases, the weights of the ResNet-50 network trained on ImageNet were used as starting points. Different image input sizes were tried, and models with and without non-image features were compared. The authors used Adam Optimizer for optimization and initialized hyperparameters with beta1 = 0.9 and beta2 = 0.999. The learning rate was set to lr = 0.001 and lr = 0.01 for transfer learning and from scratch. Batch sizes 16 and 8 were used for models based on transfer learning and for models trained from scratch with large input size. ROC analysis was performed using AUC metric to compare the performance of several ResNet-based models. The best overall performance was achieved by the from-scratch trained ResNet-37-large-meta model (with non-image features) for 5 out of 14 pathologies.

[4] This study proposes a deep learning architecture for classifying Pneumonia, Lung Cancer, tuberculosis (TB), Lung Opacity, and COVID-19 using chest X-ray images. The dataset consisted of 3615 COVID-19, 6012 Lung Opacity, 5870 Pneumonia, 20,000 Lung Cancer, 1400 tuberculosis, and 10,192 normal chest X-ray images. The images were resized to 224*224*3, normalized into [0,1], and split into 80% for training and 20% for testing. The model used VGG19 as a pre-trained model, followed by three convolutional neural network layers for feature extraction and a fully connected network for classification. Softmax was the activation function used in the classification layer to categorize the images. Adam optimizer was used with a learning rate of 0.000009, and the model was trained and validated for 5000 epochs with eight iterations per epoch and a batch size of 32. The performance metrics used were recall, loss, F1-Score, accuracy, precision, and the AUC. The results showed that the proposed VGG19 + CNN model outperformed existing work with 96.48% accuracy, 93.75% recall, 97.56% precision, 95.62% F1 score, and 99.82% area under the curve (AUC). The confusion matrix revealed the highest ratio for COVID-19, followed by Lung Opacity, normal chest, Lung Cancer, Pneumonia, and TB disease.

[5] In this study, three fully automated approaches are described for the classification of chest X-ray Images acquired by portable equipment. The objective of Chest X-ray image classification is to determine whether an image belongs to normal class, Pathological class and Covid-19 class. Since Covid-19 was highly contagious & traditional methods of chest x-ray scanning had possibility of infecting non-covid/normal people, portable equipment was used to isolate the covid-19, non-covid & normal patients.
A densely convolutional network architecture is built to work on these three approaches.

- The first approach classifies whether a patient X-ray belongs to normal or pathological category
- The second approach classifies whether a patient X-ray belongs to COVID-19 class or normal & existing respiratory conditions other than COVID-19.
- The third approach classifies whether a patient's X-ray belongs to normal or non-covid-19 or covid19 categories.

In this way, the combined response of various methods assists in categorizing and evaluating the COVID-19, normal, and pathological cases that are being considered. The dataset used in this research was supplied by the Radiology Service of Complexo Hospitalario Universitario A Coruna (CHUAC). The chest X-ray images were taken using portable equipment and only provided a single anterior-posterior view, unlike traditional methods which provide two projections (anterior-posterior and lateral views). The quality of the images obtained through portable equipment was lower compared to conventional X-ray capture methods. Two portable equipment models, Agfa dr100E GE and Optima Rx200, were used to capture the images. The dataset contains a total of 1616 chest X-ray images, including 728 from normal patients, 648 from patients diagnosed with other pulmonary diseases similar to COVID-19, and 240 specific COVID-19 cases. Data augmentation was performed to increase the number of training samples to avoid overfitting and obtain more stable models. The dataset was split into 60% for training, 20% for validation, and 20% for testing. The study utilized the DenseNet-161 architecture, where every layer is connected to each other in a feed-forward method

within each dense block to ensure maximum information flow. The classification layer was designed to support the output required for each deep learning approach, and softmax functions were used in the classification layer. The stochastic Gradient Descent optimization function was used during the training stage. According to the researchers, the proposed approaches achieved global accuracy values of 79.62%, 90.27%, and 79.86%, respectively for each approach. [6] This study proposed a deep learning pipeline architecture consisting of three modules: a Chest X-ray standardization module, a Lung disease detection module, and a Pneumonia analysis module. The goal of the Chest X-ray standardization module is to convert any inverted grayscale chest X-ray image into a conventional chest X-ray and detect landmarks of the chest X-ray. A registered chest X-ray image is generated from the detected landmarks. The goal of the Lung disease detection module is to detect chest diseases from the standardized chest X-ray. This module includes a multi-label classification model and can predict multiple diseases of the 15 classes (14 lung disease classes and 1 normal class). The sigmoid activation function was used in the classification layer. The Pneumonia analysis module's architecture has two stages for detecting subtypes of pneumonia and predicting whether it is COVID-19-based pneumonia or not. If it is COVID-19-based pneumonia, the severity of COVID-19 is determined. Three datasets were used in this study: the CheXpert dataset, the MIMIC-CXR dataset, and the Chest X-ray dataset. Data augmentation techniques, such as scaling and translation, were used. The input size of the image to the deep learning pipeline is 512x512. Four DenseNet-121 models were trained to classify chest diseases, detect pneumonia, differentiate viral pneumonia from other types of pneumonia, detect COVID-19, and identify the severity level of COVID-19. The fully connected layer was trained from scratch, whereas the convolutional layers were fine-tuned. The softmax activation function was used in the output layer, and the cross-entropy loss function was used. The Adam optimizer was used with a learning rate of 0.003. The model's performance was evaluated through the Area Under the Receiver Operating Characteristic Curve (AUC). The model differentiating viral pneumonia, other types of pneumonia, and the absence of disease had an AUC of 0.94-0.98. The model differentiating COVID-19 pneumonia and other viral or non-viral pneumonia had AUCs of 0.87-0.97. The model differentiating severe and non-severe COVID-19 had an AUC of 0.87.

[7] In this research paper, a deep learning neural network method to diagnose Covid-19 disease, assess the severity of the covid-19 & follow-up assessment through mobile device-captured Chest X-ray has been described. This method involves Knowledge transfer & a distillation framework which involves three components as Attending Physician (AP) network, Resident Fellow (RF)network & Medical student (MS) network. AP network used to extract chest x-ray imaging features from ChestX-ray8 dataset consists of a huge number of multiple lung disease labeled chest x-ray images.AP network is based on DenseNet-121 architecture. After training the huge number of images, weights are transferred to RF network which is again based on DenseNet-121 deep learning architecture. RF network differentiates Covid-19 from pneumonia and normal cases. A small dataset has been created with 3 classes Covid, Normal & Pneumonia & has been used during training of RF network. Medical Student network employs MobileNetV2 or SqueezeNet models. MS network learns the Covid-19 triage dataset and follow-up dataset which consists of series of longitudinal chest x-ray images. It is used for triaging patients, identifying the Covid-19, its severity level and based on that classifying the condition is improved, stable or worse. Authors in this research came up with a novel loss function which is Probabilistically Compact loss for training the Medical Student network. AUCROC values have been used to assess the performance of the medical student network. According to the authors, resident fellow network has demonstrated an accuracy of 0.935 during classification of Chest X-ray images when AP network pre-trained with the ChestX-ray8 dataset and MS network has demonstrated an accuracy of 0.850 for Chest X-ray image classification when knowledge was transferred from RF network to the Medical Student network. MobileNetV2 achieved a classification accuracy of 0.97 and SqueezeNet achieved classification accuracy of 0.964. For Covid-19 severity Logistic Regression, Gradient Boosting, Random Forest Classifier and FC classifier proposed by the authors were validated out of which FC classifier along with MobileNetV2 model achieved an accuracy of 0.8.MobileNetV2 network achieved a high AUCROC value of 0.883 in identifying the worse severity of the Covid-19 disease. Finally, the authors concluded that the on-device deep learning network based on MobileNetV2 could be used on high-end mobile devices since MobileNetV2 consumes high amount of resources to provide accurate results and SqueezeNet based network could be used on low & mid-range mobile devices as it requires less resources.

[8] In this study various deep learning techniques with transfer learning were proposed to classify chest X-ray images based on inflammation in the lungs, distinguishing between pneumonia, covid-19, and normal cases, and identifying the location of the disease on the image. To achieve this, they used pre-trained convolutional networks such as VGG16, ResNet50, and EfficientNetB0 as feature extractors, achieving high accuracy rates of 90%, 94.3%, and 96.8%, respectively. The authors also developed a generative adversarial framework, specifically a CycleGAN, to augment the dataset, and the Gradient Class Activation Map technique to monitor disease progression. The COVID-19 Image Data Collection on Github was used for training and validating the model, consisting of 802 chest x-ray images categorized into three labeled classes: normal, pneumonia, and covid-19, all of which were Anterior Posterior View. The images were normalized and augmented using the generative adversarial framework because the dataset was small, and five-fold cross-validation was performed to prevent overfitting during training. The model architecture includes a pre-trained model and a fully connected network, using softmax activation function for classification, Adam optimizer,

and backpropagation technique to improve performance. The EfficientNetB0 augmented model achieved an overall accuracy of 0.968, outperforming other pre-trained models. For covid-19 classification, the model reported a precision accuracy of 1, while it achieved a precision of 0.96 for normal and pneumonia cases.

[9] In this research paper, authors propsed multi label chest x-ray classification technique which combines pre-trained Convolution neural networks and problem transformation methods.Multi label classification is different to multi class classification where in the former a chest X-ray image could be assigned with multiple labels during prediction where as in the multi class classification, a chest X-ray image is classified with one or the other label and normal.The technique proposed in this paper has been validated on the Chest X-ray 14 and CheXpert datasets. Chest X-ray14 dataset comprised of only frontal view images whereas CheXpert comprised of frontal and lateral view images. The pre-trained CNNs were used to extract image features and those are given as input to the trainable classifier which used to transform the multi label problem to single label classification. The architecture of the multi-label classification involved Data Exploration, Data pre-processing, Feature Extraction and Predective model stages. DenseNet-121 model has been used in the feature extraction stage. RelU activation functions has been used in each of the Dense block. Mini batch of 8 has been used to train. Number of ephochs up to 110 considered for training.Binary cross entropy loss function was used. Adam optimizer was used. During classification stage, problem transformation methods such as Binary relevance (BR) and Classifier chain(CC) algorithms used which internally uses a linear Classifier.Following metrics are used to assess the performance of the classifier F1 score, Hamming loss & Average AUC. The described method achieved the highest AUC score of 0.882 compared to other state of the art models during the period. Classifier chain and Binary Relevance classifier methods achieved a better AUC score for multiple lables classification compared to the other state of art methods.

[10] In this research paper a new method of cascading deep learning classifiers is proposed which can enhance the Covid-19 and pneumonia disease classification performance through Chest X-ray images. This proposed method consists of two stages. In the first stage multi label classification of CXR is simplified using a series of binary classifiers. In second stage, cascaded architecture of covid-19 and pneumonia classifiers is built in such a way that different fine tuned DL models can be used simultaneously to obtain best performance of postive case detection. Authors worked on evelen pre-trained CNN models based on VGG and ResNet. VGG16, VGG19, MobileNet, MobileNetV2, DenseNet121, DenseNet169, DenseNet201, ResNet50V2, ResNet101V2, ResNet169V2 and Xception are the models validated on the public chest x-ray dataset. Results showed VGG16, ResNet50V2 and DenseNet169 achieved best detection accuracy of Covid-19, viral pneumonia and bacterial pneumonia. As per author's claim their method outperformed state of the art multilabel classification models of Chest X-ray images for three classes Covid-19, viral and bacterial pneumonia. Authors considered the dataset provided by Dr Cohen at University of Montreal. Dataset is a collection of 306 chest X-ray images, out of which 79 belong to normal class, 69 belong to Covid+, 79 belong to viral pneumonia, 79 belong to bacterial pneumonia. Workflow of the method proposed includes image pre-processing i.e., scaling the image to 150*150, one hot encoding of all the labels done against all images and data augumentation is done. Further selection of the deep learning model from the set of models considered and fine tune respective model. Train the chest x-ray images. Pass the images to the Deep Covid-19 classifier. If the prediction is positive, then patient has covid-19. If prediction is negative then pass the images to Deep Pneumonia Classifier. If prediction is positive, then patient has viral pneumonia, else pass the images to Deep Pneumonia Bacterial classifier. If prediction is positive, patient has bacterial pneumonia else the image belongs to normal or other pneumonia category. For this study authors fixed the hyper parameter values as below. Learning rate is 0.001, batch size of 7, Number of epochs are 50 & Stochastic Gradient descent optimizer is used.A Dropout of 50% is considered. Softmax activation function has been used at the classification layer. Dataset is split to 85% for training and 15% for testing. Accuracy , Precision, Recall , Specificity and F1-score are the performance evaluation metrics considered for classification. VGG16 achieved classification accuracy of 0.99 for Covid-19. ResNet50V2 achieved classification accuracy of 0.99 for Viral pneumonia where as for bacterial pneumonia classification accuracy of 0.99 achieved through DenseNet169.

[11] In this research paper, emphasis is given to the usage of HOG (Histogram of Oriented Gradient) technique for extracting key features from chest X-ray images(by focusing on primarily affected cells) followed by a less complex but yet very powerful linear SVM (Support Vector Machine) models on the combined features to detect and classify the disease. The author is trying to address a couple of problems of complex algorithms related to insufficient computing power at times and the lack of the necessary extracting functions from the images which negatively affects the accuracy of even superior models and this is all merely by introducing new techniques such as HOG descriptors (direction of edges and intensity gradient distribution), and Linear SVM which works principally by dividing the data into lines or hyper-planes. Accuracy oscillates decently between 85 and 87% with just 3000 X-ray images where the author could not use the complete dataset due to inadequate computing power (15000 images sourced from Kaggle event, annotations supported by VinBigData web platform, VinLab) however he wishes to do so as he finds it an immensely promising technique in this domain which can beat other popular models in brevity, accuracy, reliability, and optimization, less

training data. A few limitations were seen here as the author could not use all 15000 images due to less computational power, the larger size of HOG cells showed better accuracy.

[12] In this research paper, emphasis is given to the usage of a multi-layer computer vision classification model (which is generalized regression neural network (GRNN) and Grey relational analysis (GRA) based algorithms with optimization) to detect the exact type of lung disease on chest X-ray images with the bounding region of interest (ROI). To remove noise and increase symptomatic features, X-ray image texture should be analyzed using two-dimensional fractional differential order convolution with this fractional parameter (v) from 0.3 to 0.5. Further after enhancing the image quality, the author proposes to use maximum pooling to suppress the dimensions of features to speed up computations. The author combines multi-layer computer vision models with a radial Bayesian network supported by gray relational analysis to filter the images (symptoms) into distinctive diseases. The author has used a dataset from NIH clinical center (chest X-ray database) and was able to display promising results in classifying disease within bounding regions of interest with evaluation metrics such as mean recall, mean precision, mean accuracy, and mean F1 scores of 98.68, 82.42, 83.57, 0.8981 respectively with K-fold cross-validation technique. Not all the X-ray databases contain good-quality images, so the author has proposed to solve this issue in this paper by enhancing the image quality using fractional order convolution and making it part of the original database so that GRA model can self-adapt to this change in data pattern to work better within bounding ROI. This approach is doing far better than traditional GRA and GRNN models. This technique is very quick and adaptive in learning the slow progress of the disease and locating the problem by extracting specific features on even very large datasets. Hence, the proposed multi-layer classifier provided an automatic CADM tool to separate "disease present" from "disease absent" and enabled clinicians to focus on clinical treatments. In terms of opportunities, Cov-Net can prove even better by realizing more parameter optimizations with the help of swarm intelligence-based heuristic algorithms and by concatenating other data pre-processing or post-processing to enable class-dependent robust features.

[13] In this research paper, emphasis is given to the usage of the computer-aided diagnosis model Cov-Net. For feature extraction, a modified residual network supported with asymmetric convolution and attention technique is also fused in it, soon after it, to achieve high-level semantic and low-level detailed information, skip connected dilated convolution with different variation rates applied. Attention technique eliminates the dilemma that arises due to over focus on common symptoms in Covid-19 and other pneumonia and same time eliminates information loss due to less attention on them. This model has performed very well on two public Covid-19 radiography datasets (Kaggle) with 99.6 and 99.01% respectively for Covid-19, Normal and Viral pneumonia diseases on D1 and Covid-19, Normal, Lung opacity, and Viral pneumonia diseases. The author proposes this technique for relatively fewer training datasets. This model has shown very generalization ability over all other advanced computer vision algorithms (ResNet50, MobileNetV2, DarkNet53, GhostNet, InceptionV4, DenseNet161) and is known for its robustness, reliability, and accuracy. Con-Net was also seen performing better than Cov-Net* (without data augmentation) in all the metrics as without data augmentation, data imbalance worsens the performance. Opportunities lie in optimizing the architecture and integration with other data processing techniques. Also, Cov-Net should be trained on the private data of hospitals to get clinical validity.

[14] There are many popular Deep learning classification models which are used in the medical domain however for Chest X-Ray image datasets to detect the critical issues and especially localization of those come with the high cost of labeling or annotations of such images in terms of pixel-level labels or bounding box regions and lack of explainability of such models. To overcome these issues, this research focuses on the combination of both classification and localization using multi-instance learning (weekly supervised problem). This model has proven competitive on pneumothorax, pneumonia, and pulmonary edema coming from three different datasets (). Deep learning techniques use the saliency method (Gradient Class activation mapping, Grad-CAM) by providing pixel-wise heat maps to predict the disease class. Such models explain using network weights for each class and until the class is predicted they are not determined. Apart from that, feature maps are created using low-resolution filters and then projected back to the original input size. Hence, first, down-sampling is done to meet the sizes of pre-trained models which might degrade the quality of detection and localization of the issue. Finally, with the help of regional bounding boxes or pixel-wise classes, detection and segmentation work. However, in the medical industry, such luxury of crowd-sourced annotated images is not there. Here, MIL comes as a savior where images are broken into numerous patches or instances which are classified as critical findings (positive) or not (negative) with the help of the CNN model and patch scores as output which is used as an image-level label in the loss function. The average AUC was 0.93 in MIL against 0.96 in CNN and 0.92 in FCN, hence MIL outperforms others while classifying and localizing same time. This also appears to be advantageous for the same reason. Opportunities lie in this patch-based CNN too in the form of multi-class patch-based localization, regression for criticality measurement, and additional patch-based heatmaps saliency for finer localization detailing.

[15] In this research paper, Vision Transformers (ViT) is presented as a potential alternative to decade-old Deep Learning techniques. Vision Transformers have outdone Deep Learning on several grounds and metrics in the field of diagnosing severe lung diseases. The research was implemented on the very basic machine and common dataset (National Institutes

of Health Clinical Center, Kaggle) where its relative model VDSNet (Hybrid CNN) was compared with the Vision Transformer model. The vision transformer model uses an encoder of a normal Transformer. It considers if patch size decreases and the number of layers increases then training time and accuracy both increase. The image is split into patches, patches are flattened to a single-dimensional vector. Positional embeddings are added to locate patch position a.k.a. 1-dimensional sequences of token embeddings, which go into an encoder made up of several layers. MLP finally helps obtain the categories. For the implementation of ViT, PyTorch framework was used, and for VDSNet TensorFlow was used with machine specifications such as Pop OS (GNU/Linux), Intel Core i5 9th gen, Nvidia 1660Ti, 16GB RAM. The model was trained for 10 and 20 epochs on fractional and full datasets respectively. Accuracy is 70.24% compared to CNN-based VDSNet's 69.86% with a smaller dataset. The training time of such a network increases slightly as the number of layers increases. With more datasets, this model will perform even better.

[16] In this research paper, Data augmentation techniques like Affine and DCGAN (Deep Convolution Generative Adversarial Network) are compared in terms of accuracy, recall, and AUC metrics to determine that DCGAN fetches far better results compared to traditional techniques. The pneumonia dataset contains 5863 chest X-Ray images. Obtaining large amount of Chest X-ray image datasets is not easy which is the backbone of model learning, hence data augmentation techniques are predominantly used for this purpose and to overcome over-fitting issues in the modeling. The traditional affine model applied to input data and performance of various top CNN models (VGG16, Inception V3, ResNet, DenseNet, etc) compared against DCGAN's accuracy for the classification of lung diseases. DCGAN is based on Generator (convolutional layers) and batch normalization (Leaky Relu activation function) with a convolution layer with filters and the tanh activation function to generate images and Discriminator (transpose convolutional layers) on the other hand takes both real and sample images and outputs the image as fake (no disease) or real (disease) for classification and detection of disease. Accuracy is as high as 98% and recall as 100% with AUC as 0.99. DCGAN is computationally pretty fast compared to any top CNN models.

[17] A progressively growing GAN (PGGAN) model was proposed in this paper to solve the classification problem for Covid19, Pneumonia, and healthy lungs as it required high-resolution synthetic X-ray images which are not possible using other advanced predecessors of GAN (Generative Adversarial Network, class of deep learning algorithms) networks like CovidGAN, RANDGAN, ACGAN, DCGAN, CycleGAN. Apart from this, a very deep CNN (complex) model was used to classify the Covid 19 and Pneumonia to identify the differences between the two diseases that are very close to each other in terms of image feature characteristics. A couple of architectures were proposed for very deep CNN – ResNet and DenseNet with very high accuracy of 98.32 and 99.25% respectively compared to other classical CNN models like Covid-NET, YOLO, etc where accuracy for binary classification up to 99% achieved but accuracy dropped to 97% for multi-classification due to lack of high-resolution data. Hence that can be seen as an opportunity. The author has used COVIDx12 publicly available dataset.

[18] GAN networks have been used with a fusion of Deep transfer learning (base part) and LSTM (head part) networks (i.e., proposed DNN model) in this paper to achieve very high classification performance for 2 to 4 classes in 7 different scenarios like healthy, viral, bacterial, COV-19. No need for explicit feature selection or extraction and this proposed DNN can serve directly. Accuracy, recall, etc are found to be 90% with this approach and 99% for binary classification (Cov-19 and normal). This proposed network was compared with Inception-ResNet V2, Inception V4, VGG16, and MobileNet popular in Pneumonia studies). Performance seems to be higher in terms of accuracy, precision, sensitivity, and specificity. Limitations are mainly around COVID-19 limited samples. For the data, six different reliable databases were used on chest X-ray images. All the hyper-parameters related to the proposed DNN are carefully modified in trial-and-error method learning rate as 0.001 and batch size 10 through for the Mean Squared Error cost function and RMSProp optimizer. All the experiments were performed in Google Collab with 14GB RAM and a Tesla K80 GPU graphics card.

[19] In this paper, the author is proposing a deep transfer learning technique, particularly an ensemble of GoogLeNet, ResNet-18, and DenseNet-121 to deal with the scarcity of X-ray image datasets to detect Pneumonia disease. The ensemble technique depends on weighted average probability over key metrics like precision, recall, f1-score, and AUC for binary classification between Pneumonia and normal cases. With 5 cross-validation models achieved an accuracy of 98.81% and 86.86% over Kermany and RSNA datasets respectively which is higher and more robust over many classical approaches and ensemble methods.

[20] In this paper, the concatenation technique of any two different transfer learning models is attempted with two equitized CT scan and X-ray datasets for the classification of Cov19 Pneumonia and healthy cases. Overall many models like DenseNet121, MobileNet, Xception, InceptionV3, ResNet50, and VGG16 models were used in the pair combinations. The best accuracy of 99.87% was achieved with ResNet50 and VGG16 models. This multimodal technique produced far better results compared to the highest single modal classification accuracy of 98% with ResNet50 and CT scan data and 98.93% with VGG16 and XRay images

[21] In this paper, emphasis is put on Normalization-free networks (NFNets), which are trained on the ImageNet dataset and later fine-tuned for the classification of Tuberculosis. Additionally, the Score-Cam algorithm is used to focus the portions of images for detailed inference of the detection of disease. This technique works equally well for both binary and multi-class classification with decent accuracy, AUC, sensitivity, specificity, avg precision , and avg. recall of 98%, 99%,92%,99%, and 96.1% respectively which makes this model quite promising to be used as an alternative tool for radiologists. Kind of Pre-processing and classification techniques introduced here makes this process overall very unique augmentation technique (RandAugment, known for its accuracy), progressive sizing (augmentation based on images sizes), normalization-free networks (instead of error-prone batch normalization to deal with performance issues), adaptive gradient clipping (to deal with exploding gradients), Score-CAM (complete visual analysis from medical domain perspective). Focus lies in mainly 3 sections - differentiating between healthy and TB patients, sick (not TB) and TB patients, and underlining symptomatic TB areas. Datasets TBX11 K, Montomery, Shenzhen, Dataset A + Dataset B (from National institute of TB, New Delhi). This technique makes the model independent of batches (min batch size, discrepancies in training and testing results), and time efficient (by faster prediction time and fewer computations). Modified architectures of EfficientNet-B7 and normalization-free ResNets make this model a superior performer.

[22] This paper differs from other research, which focuses on advanced AI models, by using a classical machine learning model (SVM) to diagnose Covid-19 in a small dataset of 1100 radiographs, consisting of 300 Covid-19, 400 normal, and 400 other pneumonia cases. The images were preprocessed using a PHOG image feature descriptor, which characterizes spatial features through edge detection. The SVM classification was optimized using sequential minimum optimization, with important hyperparameters tuned to avoid overfitting. The model was trained on a basic machine configuration and evaluated using 10-fold cross-validation. The results showed high accuracy in both binary and multiclass classifications. The model is reliable and has better performance, making it suitable for testing several variants with less time and faster operation, but it should not be used for very large datasets.

[23] In summary, this research paper highlights the importance of developing computerized support systems to assist healthcare professionals in diagnosing diseases accurately. The authors employed the CheXNet model to train CNN-based models that perform multi-class, multi-label classification and compared pre-trained CNN models and their performance. The study achieved an overall accuracy of data prediction and successfully predicted several labels, such as Fracture, Lung Lesion, Pleural Other, Pneumonia, and Pneumothorax. However, the class imbalance affected the model's ability to predict positive cases of certain labels, and the researchers recommend using over-sampling or under-sampling techniques to reduce the imbalance.

[24] This study uses deep learning techniques to detect tuberculosis from chest X-ray images, including image preprocessing, data augmentation, image segmentation, and transfer learning with nine different pre-trained CNNs. The proposed method achieved state-of-the-art performance in the computer-aided diagnosis of tuberculosis. The study used two different databases, one for lung segmentation and one for TB classification, and conducted three experiments to evaluate the accuracy, precision, sensitivity, F1-score, and specificity of the proposed method. The results show that the proposed method can achieve high accuracy, precision, and recall for the detection of TB, and that lung segmentation improves classification performance. The study also confirms that CNNs learn from the segmented lung regions, which resulted in higher detection accuracy.

[25] This paper proposes four deep-learning models, including ResNet152V2, MobileNetV2, a CNN, and an LSTM, to detect and classify pneumonia from chest X-ray images. The proposed models improved accuracy, precision, F1-score, recall, and AUC by 99.22%, 99.43%, 99.44%, 99.77%, and 99.77%, respectively. The dataset used for training and validation included 5856 images, with 1583 normal and 4273 pneumonia cases, and was augmented for each classification. The models achieved more than 91% accuracy and recall, and future work will explore the use of other CNNs and RNNs to detect pneumonia from chest X-ray images.

[26] A collection of 6432 Chest X-Ray scans from Kaggle was used to train and validate multiple CNN models to classify normal, Covid-19 affected, and pneumonia cases. The Xception model achieved the highest accuracy for detecting chest X-ray images, with 490 covid cases, 1345 normal cases, and 3632 pneumonia cases in the training set. The results obtained are promising but need to be tested on new data.

[27] The paper discusses the use of Gradient Class Activation Map (Grad-CAM) as a tool to detect characteristic features from X-ray images for the purpose of aiding visually interpretative decision-making. The Grad-CAM technique was employed to identify the regions in the X-ray images where the model paid more attention during classification, which was found to correlate with clinical findings.

To carry out the study, data was acquired from four open-source databases, namely, the Italian Society of Medical Radiology and Interventional (25 cases), Radiopaedia.org (20 cases), J. Paul Cohen et al. (180 cases), and a hospital in Spain (80 cases). The chest X-ray images were obtained from the NIH Chest X-ray Dataset6, consisting of normal

samples and samples related to 14 lung, heart, and chest-related diseases. Two classes, normal and disease, were generated from the dataset, with four subclasses corresponding to normal, other diseases, pneumonia, and COVID-19.

For training the model, transfer learning was used as a promising alternative to training a Convolutional Neural Network (CNN) from scratch. Transfer learning involves fine-tuning a pre-trained CNN on a large set of available labelled images from another category, which can speed up convergence while lowering computational complexity during training. The shallow fine-tuning of the last few layers was sufficient for transfer learning.

Due to the limited availability of the COVID-19 dataset, the researchers considered it inadequate to train a CNN from scratch, which involves a vast number of trainable parameters. Therefore, transfer learning was employed to leverage the knowledge, features, weights, etc. learned from the source domain (DS) and source task (TS) for training newer models for the target domain (DT) and target task (TT).

The research holds significance globally, especially in the current COVID-19 pandemic, and the initial results of the study show promise. However, the authors suggest that more extensive and diverse data sets are needed for replication to further validate the results.

[28] The paper introduces a DL-based automated method for detecting COVID-19 infection cases from normal cases using chest X-ray images. The study evaluates eight pre-trained CNN models to determine the best-suited model. The models were compared based on various factors, including batch size, learning rate, number of epochs, misclassification rate, and type of optimization techniques. The proposed method is easy to implement, does not require manual feature engineering, and can help radiologists with accurate and stable diagnoses. To address the data imbalance problem, multi-operation data processing was performed, and data from publicly available datasets were used. The ResNet-34 model outperformed other models, achieving an accuracy of 98.33%. The proposed method has significant performance for binary classification, and future studies will explore the use of optimization algorithms to design more reliable models. This study contributes to the development of efficient methods for COVID-19 screening and diagnosis using chest X-ray images.

[29] The developed model can accurately detect non-COVID-19 cases with an evaluation on 1531 X-Ray images and two dataset splits, achieving a detection rate of 96.00% and 70.65%. The dataset consists of 100 chest X-Ray images from 70 subjects and 1431 chest X-ray images diagnosed as pneumonia (not COVID-19) from 1008 subjects. The model can be considered as an effective computer-aided diagnosis tool for low-cost and fast COVID-19 screening, but it still has limitations such as missing 4% COVID-19 cases and a false positive rate of almost 30%. The authors aim to improve the model's effectiveness by reducing the false negative rate and decreasing the false positive rate and validate it with more clinical data.

[30] This paper proposes using deep learning models to automatically analyze chest X-ray images for COVID-19 screening. The models are tailored to detect pneumonia cases, including viral cases, and easy-to-apply health indicators are proposed for predicting patient status. The Chest X-Ray Images (Pneumonia) dataset is used to train the models, and the DenseNet169 architecture achieved the best performance with an average classification accuracy of 95.72%. The study also evaluates the models' performance on a blind test set of 145 COVID-19-infected patient images and suggests future work to improve the models' reliability.

[31] The paper evaluates different CNN-based X-ray classification approaches and finds that the X-ray-specific ResNet-38, integrating nonimage data, yields the best overall results. They used a 5 times re-sampling scheme to assess generalisation performance and estimated the average validation loss overall re-samples to determine the best models. The study performs a ROC analysis, compares classifier scores by Spearman's pairwise rank correlation and uses Grad-CAM to assess CNN model predictions. The optimised ResNet-38-large-meta architecture achieves state-of-the-art results in five out of fourteen classes compared to Guendel et al.14. The study concludes that training deep neural networks in the medical domain becomes viable as more public datasets become available.

[32] This article explores the potential of machine learning methods for the automatic diagnosis of COVID-19 from X-ray images. Logistic regression and CNN classifiers were used, along with dimensionality reduction using PCA to speed up the learning process and data augmentation using GAN. The final dataset contained 250 samples for each class, and the models achieved an overall accuracy of 95.2-97.6% with 97.6-100% accuracy for positive case identification. The study concludes that PCA can be used as a feature extraction technique with CNN, while LR can extract features when feeding LR. The researchers plan to train the network on a larger dataset and improve the system's capability of learning highly abstract features.

[33] Detecting and diagnosing lung cancer at an early stage is critical to improving the chances of patient survival. This study proposes a two-step approach to diagnosing lung cancer from CT images using feature extraction and classification phases. The feature extraction phase uses "image processing techniques" and a "connected pre-trained convolutional neural network model" in order to highlight the lung region and extract features from the images (Aliet al.

2022). The classification phase employs recurrent neural networks with multilayer perceptrons and a support vector machine classifier to classify CT scans and determine if they contain abnormalities indicative of lung cancer. Deep learning, and specifically convolutional neural networks, provide significant advantages over conventional machine learning algorithms in feature engineering, enabling the analysis of data to discover and include related features in the learning process. Tomographical image data is used in this approach and it can handle complicated 3D image datasets and automatically extract features without the need for manual intervention. Overall results directly indicate that the proposed techniques can improve the survival rate of lung cancer patients by detecting and diagnosing lung cancer more accurately and efficiently (Ahmedet al. 2020). The study presents an accuracy graph indicating that the used method can achieve improved accuracy in lung cancer diagnosis, and images are provided to demonstrate the abnormal and normal classifications of CT scans with SVM reaching 82% and classical algorithms reaching 65%.

[34] This paper presents a method for diagnosing "lung cancer" from CT scans using "deep learning" based "support vector network". The proposed method aims to improve the accuracy of lung cancer diagnosis and reduce the false positive rate. The study used a dataset of 179 CT scans, which were divided into 89 cases of lung cancer and 90 cases of non-cancerous lung nodules. The dataset was pre-processed to normalize the intensity of the images and remove any noise. The proposed method consists of two principal steps, "feature extraction" as well as "classification". In the feature extraction step, a deep "convolutional neural network" (CNN) was used to pull out characteristics from the CT scans. The VGG16 architecture was also used in this paper, which was pre-trained on the ImageNet dataset and fine-tuned on the lung CT dataset. In the classification step, an SVM was used to classify the CT scans as either cancerous or non-cancerous (Shafiet al. 2022). The SVM was trained using the extracted features from the CNN. The proposed method was evaluated using a 10-fold cross-validation technique. The results exhibited that the proposed method accomplished an overall accuracy of 93.63%, a sensitivity of 94.39%, and a specificity of 92.86%. The false positive rate was reduced to 7.14%, which is a significant improvement compared to previous studies. The method used in this paper was also studied with other progressive methods for "lung cancer diagnosis", including a traditional SVM-based method and a "deep learning" based method (Hosseiniet al. 2022). The results depicted that the planned method surpassed both methods in terms of quality and false positive rate. The study also conducted an ablation analysis to investigate the contribution of different components of the planned method. The results showed that deep CNN was the most essential component for achieving high accuracy, while the feature selection process improved the false positive rate.

[35] This paper will execute an analysis of Lung Cancer Nodules Detection and Classification in CT Scans. The main purpose of this paper is to understand how deep learning can help the detection of lung cancer Nodules, also how CT scan techniques can be effective if Deep learning will include. Results from this study demonstrate that Deep learning and CT scan can be used to reliably classification in CT Scans and Lung Cancer Nodules Detection. When compared to other cancers, lung cancer consistently ranks as the worst. This is why a lot of nations are working on programs to catch lung disease early. Due to the fact that tumors are difficult to spot even for experienced medical professionals, radiologists are under increasing pressure as the volume of CT images they must evaluate grows. Malignant tumors can only be found if doctors know to look for and assess a very precise set of characteristics. Since deep learning CAD systems can acquire the most important characteristics during training, they can conduct end-to-end identification. Due to the network's ability to record the characteristics of tumors in a variety of CT images with different parameters, it is resilient to changes (mdpi.com, 2023). Training on a diverse dataset allows the algorithm to pick up on unchanging characteristics of cancerous tumors, leading to improved accuracy (ncbi.nlm.nih.gov, 2023). There are a total of 244,527 pictures in the collection, which was compiled from 1018 CT scans taken on 1010 individuals. This data collection enables a multi-tiered analysis and evaluation. Some of the proposed techniques split the task into two phases (candidate identification and false positive reduction), while others use only a single network to solve the issue. Disparate methodologies explain why the outcomes aren't consistently shared. In an aspect of this project, the developers introduced several deep CAD methods and models with the goal of making doctors' lives easier when it comes to detecting pulmonary nodules. It is demonstrated that deep learning has progressed to the point where it can be used not just as a second opinion in diagnosing but as a potent instrument that doctors should take into account. The reviewed literature demonstrates that high performances for lung cancer diagnosis using CT images have been achieved through the application of deep learning methods. The suggested methods utilize sophisticated Convolution Neural Networks (CNN) methods that combined deep learning with more traditional machine learning methods.

[36] This paper presents a novel deep-learning approach for the classification of lung tissue images from the LC25000 dataset. The proposed method consists of a "hybrid model", consisting of a "feature extractor" and "classifier". The "feature extractor" is composed of "three sub-extractors", namely, "inception_v3 network", "hog", and "daisy", which are used to extract visual aspect features from multiple positions (Chenet al. 2021). Each of the sub-extractors is fed the input image, and their output tensors are combined to obtain the final output of the extractor. The classifier then receives the "output tensor" of the extractor, which is then passed through a "3-layer Fully Connected Network". The "softmax layer" is utilized for terminal classification. To train this model, "KFold" is used to select 80% of the dataset as the

training set, and "Adam" is utilized as the optimizer. This paper evaluates the performance of the proposed model against "ResNet" and various accumulations of the covered extractors using metrics such as "weighted average precision", "recall", "F1-score", and "accuracy". The "CNN" models outperform traditional descriptors, and accuracy results from all models using CNN are comparable. The combination of "traditional descriptors" and "CNN" based models leads to slight improvements in every metric, suggesting that they can learn feature representations from different perspectives and complement each other (Pandianet al. 2022). The "Inception-based hybrid extractor" outperforms the "hybrid extractor" using "ResNet", achieving a great 99.97% accuracy. The study concludes that "hybrid deep learning models" are effective in accurately diagnosing cancer, with the proposed model achieving an impressive accuracy of up to 99.6%. The overall sum of each "sub-extractor" was also assessed by withdrawing them separately from acquired extractor.

[37] This presents a deep-learning approach to classify malignant and benign cancer nodules in lung CT scans. The proposed system utilizes a "25-layer deep Alexnet architecture" of a "Convolutional Neural Network" (CNN) to improve the accuracy of the classification process. This paper also highlights detecting lung cancer at an early stage can save lives and emphasize the need for automated and accurate diagnosis of lung cancer using CT scans. The study used a dataset consisting of 624 lung CT images, including 252 benign cases, 218 malignant cases, and 174 normal cases (Garudand Dhage, 2021). A confusion matrix to evaluate the performance of the proposed system is used in this paper, and the results showed an overall accuracy of 85.4This study reports that the system performed better in classifying malignant nodules than benign nodules, indicating that more work is needed to improve the accuracy of the classification for benign cases. This paper concludes that the proposed system can accurately classify malignant and benign cancer nodules in lung CT scans using a deep learning approach. However, it has been suggested that further research is necessary to improve the accuracy of the classification for benign cases (Pandianet al. 2022). One of the limitations of the study is the relatively small size of the dataset used, and information from this paper recommends that increasing the size of the training set can provide more accuracy in future work.

[38] This paper advises a deep learning model based on convolution neural networks (CNN) for detecting and extracting the tumour portion in lung cancer images. The CNN model is designed to classify the stage of lung cancer and predict the possibility of cancer using Tx stages 2, 3, and 4. The model is trained using a back-propagation method with CT images, which is divided into two phases: training and testing. The training set consists of 200 images and uses Dicom data to minimize feature loss. The "CNN model" uses a "scheduled learning strategy" with 560 training and 325 validation samples. The model's metrics are computed for classification resulting in an accuracy of 81% after 20 epochs. The number of epochs indicates the machine learning process's completion after passing the entire training dataset. The model performs several operations to validate the samples, and a dropout layer is used to prevent packet dropping (Deepaand Fathimal, 2022). The accuracy metric is determined based on the overall number of "epochs" and "iterations". The channels in the image are used to compute the layers in the CNN model. Overall, the proposed CNN model achieved an accuracy of 85%, making it a promising tool for automatically detecting and classifying lung cancer stages. Accuracy measures the successful categorization of pixels in an image, while loss function predicts errors in the neural network and serves as a performance measure parameter. Computation time, on the other hand, refers to the duration taken to complete the computation process (Ahmedet al. 2020). The proposed CNN model achieves an accuracy of 85%, a significant improvement in detecting and extracting tumor portions from lung cancer images. The CNN model is tested using images from the dataset, demonstrating its novelty. Overall, the CNN model presents a promising approach to predicting and classifying lung cancer stages, providing assistance to doctors and clinicians in diagnosing and treating lung cancer patients.

[39] This paper utilizes a novel approach for the classification of lung cancer histology using deep learning techniques applied to "computed tomography" (CT) images. In this paper, it has been recognized that the challenges associated with traditional histological analysis methods and suggest that their proposed approach can improve the accuracy and efficiency of lung cancer diagnosis and treatment. This proposed method uses deep "convolutional neural network" (CNN) model trained on a large dataset of CT images to classify lung cancer histology into four categories. Such as, "adenocarcinoma", "squamous cell carcinoma", "small cell carcinoma", and "non-small cell carcinoma" not otherwise specified (NSCLC-NOS). The dataset used to train and test the model consists of over 2,000 CT scans from patients diagnosed with lung cancer, which were annotated by expert pathologists.Approaches of this paper are also compared to traditional histological analysis methods, such as "manual pathological examination" and "immunohistochemistry" (Tafadzwaet al. 2021). The model achieved an overall accuracy of 93.2%, with area under the "receiver operating characteristic curve" (AUC) of 0.961, which outperforms previous studies on the same dataset. In addition, sensitivity analysis is also conducted in this paper to evaluate impact of different CT image acquisition parameters on the performance of their deep learning model. The results show that the model's performance is robust to variations in CT image acquisition parameters, which suggests that the proposed method can be used in diverse clinical settings (Chaunzwaet al. 2021). This paper also provides a visualization tool that can help clinicians interpret the model's classification results by highlighting the regions of the CT images that are most indicative of each histology type. This

tool can aid clinicians in identifying the most relevant features for each histology type and improve the accuracy of lung cancer diagnosis.

[40] This paper will execute an analysis of the Efficient Lung Cancer Image Segmentation and Classification Algorithm Based on an Improved Swin Transformer. It is the goal of the categorization assignment to assign labels to the provided example pictures (e.g., lung cancer image classification). To help physicians improve their clinical diagnostic times and precision, the categorization mission can offer useful advice (arxiv.org, 2023). Researchers used Swin Transformer in their studies of lung cancer in an effort to better detect the disease at its earliest stages. While the transformer is commonly used in NLP for handling natural language patterns, the CNN for image processing direct handles the picture as a matrix for convolution operation. Using it as a convolutional neural network (CNN) for extracting features from images would be a significant challenge. Included in LIDC/IDRI is the LUNA16 collection, which consists of lesions identified by four expert lung doctors. There are a total of 888 pictures in the collection, all of which are low-dose CT scans of the lungs and feature sagittal portions of the chest. The Tumors lower than 3 millimeters in size were also excluded from the LUNA16 dataset during curation. Due to the difficulty even seasoned physicians have in spotting lesions of this size, this streamlines the training process. Imaging equipment, layer thicknesses, and individual patients all play a role in determining how many segments are used to create a single picture (mdpi.com, 2023). Three-dimensional in nature, the initial pictures are actually cross-sectional views of the ribcage in longitudinal planes. These three-dimensional pictures are constructed from a collection of two-dimensional pictures. Traditional picture processing technology has come a long way thanks to transformers. In this article, the Authors suggest a novel approach to segmentation based on an effective converter, and they use it to analyze medical images. It is an innovative visual translator that produces hierarchy feature models with linear processing difficulty with respect to the size of the incoming picture.

[41] The paper proposes appropriate methods in order to detect lung cancer using “low-dose CT scans” and “machine learning” techniques. Lung cancer can be seen as a horrible form of cancer with a high mortality rate, making early detection crucial. “Low-dose CT scans” have been shown to be efficient in order to understand and identify lung nodules. On the other hand, the image quality is poor compared to normal CT scans (Thakral et al. 2022). The proposed methodology uses “deep learning” approaches and “convolutional neural networks” (CNN)in order for upcoming selection and classification. The study aims to provide a new direction for researchers in premature detection of lung cancer using appropriate“LDCT images”. The paper is divided into sections discussing the various modalities used for lung cancer detection, related work, basic terms, available datasets, suggested methodology, discussion and challenges, and conclusion. As per, Joy Mathew et al. (2020), existing “computer-aided detection” (CAD) systems have flaws such as “low automation”, “low sensitivity”, “slow processing”, “high false-positive rate”, and “low computation efficiency”. The proposed methodology focuses on improving image quality and also maintain curtail information in “LDCT” scans by reducing noise and abnormalities. The paper discusses the difference between benign and malignant lung nodules and the importance of detecting cancer on the premature stage to reduce mortality counts. The study aims to provide a reliable system combining “image processing methods” with “machine learning” algorithms for premature lung cancer detection.

[42] This paper will execute an analysis of Lung Cancer Detection and Classification from Chest CT scans. The main aim of this paper is to represent an analysis of these two techniques about how these techniques can be effective if machine learning techniques will include. Results from this study demonstrate that machine learning and picture processing can be used to reliably classify and forecast lung cancer (Tlcr.amegroups.com, 2022). The first step is to collect images. For the purpose of the exploratory study, 70 individuals’ CT images were pooled together to create a collection of 83 scans. On the other hand, about five million people are diagnosed with lung cancer every year, making it one of the leading sources of death everywhere. Comparatively, fatality rate is even greater than that of breast and prostate cancers taken together. The collection of chest CT scan pictures was analyzed using a variety of machine learning algorithms for texture feature categorization, including support vector machine (SVM) technique and K-nearest neighbors (KNN) (Ncbi.nlm.nih.gov, 2023). The suggested method outperforms state-of-the-art methods, with an accuracy of 93% for support vector machines and 91% for K-nearest neighbors. [43] Objective of this article is to solve issue for taking perfect x-ray image using AI and analyze disease including noise as much as possible like scapula overlapping lungs, lungs field lose etc. to solve this problem author used ImageNet which was not enough for classification. Author used Unet also with shortcuts connect shallow layers to deep layers. semantic segmentation is used for diagnostic quality assessment. Accuracy of model was not good hence distance optimization was done. Author proves that pixel wise supervision is required image wise is not enough. Based on multi-label semantic segmentation, author suggest the Semantic Segmentation-based Classification Network for this task, and distance map regression is introduced to group the issue-related regions together. In box-supervised methods coarse segmentation annotations having great potential. AUC of ROC curve to over 0.93

[44] Author is using Bidirectional Convolutional-LSTM in this article in order to achieve required result. At the lowest level of UNet, multiple kernel pooling (MKP) blocks are used to encrypt more spatial information using variously

sized pooling operations along with Spatial Pyramid Pooling. In this project author also tried to use Unet which is state of art technology but it is having mostly two limitation first from the encoding and decoding pathways, exclude connections that blend feature maps with equal sizes. Second repeated downsampling processes lead to the loss of spatial information. This is used for multilevel classification of lungs disease using chest x-ray. [45] This article is using CNN to predict lungs disease like Atelectasis, Consolidation, Effusion, Mass, Nodule, Pleural Thickening, and Pneumothorax. Author is working on data preprocessing to reduce computation reduces size of image, DenseNet121 model is used which was providing .65 accuracy, area covered under AUC of ROC curve to over 0.80 which is expected result from author perspective but it is not that great, fine tunning of model can be done. [46] Author has tried multiple models Hybrid Dense U-Net , Unet, SegNets , but mostly worked on Unet with their own modification. With improvised architecture they worked on Gaussian dropout, model is used for CXR images and used nested convolution blocks and dense skip connections. A modified UNet model with Gaussian dropout that adds more stringent regularisation to the architecture and hence reduces generalisation test error is what we suggest and assess. They demonstrate that the usage of exponential linear units (ELUs) and data augmentation result in more meaningful segmentation outcomes. Data argumentation is also used to get precise image segmentation results. This model's inner layer concatenates dense and protracted skip connections made at several convolutional layers. The convolution blocks in the skip paths between nodes are dense. These blocks combine the output from the previous convolutional layer of the same dense block with the upsampled output from the lower dense blocks. The last stage's output nodes combine all of the earlier feature maps. Accuracy is 0.9680 [47] In this paper author's main objective was to predict the accuracy score using depth wise. To achieve this author used different model which is existing like DenseNet, VGG16, InceptionNet, ResNet, Depth wise convolution, experiment was performed as on same data set using different model with the help of transfer learning. Data argumentation was done rotation, zoom, horizontally moved, vertically moved and flipped. Depth wise convolution is also done. Layers that only add a few parameters are picked. Conv layer, for instance, is changed to Depthwise Conv. The number of parameters introduced by this method will be lower than that of conventional convolution. As each channel has a unique set of filters, more data is gathered. With these layers, a network known as Xception is created. Convolution is utilised along with batch norms. As the network gets more complex, batch norms become more significant. A dense layer with the necessary number of neurons is also added. Thereafter training is carried out at a rapid learning pace. For the network's depth, experiments are also conducted with the number of neurons in the dense layers. Following the discovery of a suitable depth, network training with a slower learning rate and decay is initiated. Accuracy of this model was .85" [48] Authors had worked on segmentation in this paper. They have user a method Mask R-CNN, which is based on two branches, target object classification is one, bounding box coordinate regression is another and the Mask R-CNN adds a third branch that produces the object mask. Author designed a Mask R-CNN-based automatic segmentation and annotating system. This essay uses resnet101 as its primary structure. The learning efficiency and network prediction accuracy are significantly increased by the residual network's deeper network and fewer parameters. Epoch was 600, learning rate is decreased from 0.001 to 0.0001, In order to address the issue of automatic segmentation annotation in medical photos, this research initially suggests an instance segmentation technique. Automatic labelling is significantly more important than conventional manual labelling for the auxiliary diagnosis and care of computers. [49] This paper is focused on segmentation and XLSor Author is using data from multiple sources but only chest x-ray to detects multiclass lungs disease. Main working principal is that it will first do segmentation and will identify lungs out of the chest x-ray, then it will separate lungs image from chest x-ray and then identification of disease will take place along with entire chest x-ray, both x-ray result will be concentrated then classification will be done for diseases which looks most promising and efficient as of now. For classification DenseNet121 is used. Residual-Inception Unet is used for lungs segmentation. Accuracy of this model is quite high ranged in .9818 and .9914 . [50] Author created a model that simultaneously diagnoses diseases using photos with disease labels. DenseNet121 is used as a basic neural network architecture due to its well-known performance. The performance of the model diagnostic is then enhanced by applying the entropy weighting loss function. Given that matching labels cannot fully describe an image, classification tasks based on the kind and quality of labels are performed using the deep CNNs model. Deep CNNs are used to categories chest X-ray pictures based on the types and quantities of existing matching labels under the assumption that labels cannot be totally accurate and labels cannot completely determine samples. Deep CNNs are typically used to maximize the accuracy of diagnostic data with ambiguous preceding data. User clams that result on 14 classes using VGG16, ResNet50, and DenseNet121, three distinct pre-trained basic models. We can see that the four diseases of cardiomyopathy (AUC = 0.8790), pneumothorax (AUC = 0.8379), fibrosis (AUC = 0.8040), and pleurisy likewise show good performance with DenseNet121. [51] Using deep learning techniques, automatic chest X-ray analysis has advanced significantly. Deep models' data-driven feature necessitates training data with a wide distribution in order to function. Consequently, it is important to combine information from several sources, particularly for medical imaging. It is still difficult to learn a disease classification model using more Chest X-ray (CXR) data. Recent studies have shown that joint training on several CXR datasets suffers from a performance bottleneck, but few have attempted to overcome the problem. We contend in this study that using an external CXR dataset produces inaccurate training data, which increases the difficulties. In particular, there are two types of incomplete data: domain discrepancy, which arises from the fact that picture appearances vary

among datasets, and label discrepancy, which results from the partial labelling of various datasets. To achieve this, we define the multi-label thoracic illness classification issue as a set of weighted independent binary tasks based on the categories. To reduce feature disparities for common categories that are shared across domains, we use task-specific adversarial training. To further mine the data from the missing labels for categories contained in a single dataset, we provide uncertainty-aware temporal ensembling of model predictions. By modelling and addressing the domain and label conflicts simultaneously, our approach enhances knowledge mining capabilities. Using three datasets containing more than 360,000 Chest X-ray pictures, we undertake in-depth studies. On the official NIH test set, our strategy surpasses other competing models and achieves state-of-the-art performance with 0.8349 AUC, proving its efficacy in using the external dataset to enhance the internal classification. DenseIBN-121 is used as the backbone network. Like with DensNet-121, DenseIBN-121 uses densely skip connections; the only difference is that Instance Normalization and Batch Normalization are used in the Dense Blocks. In cross-domain situations, it exhibits superior generalisation, as shown in the original paper. As stated in section III-B, we change the classifier layer to C branches, each with two fully connected layers, and initialise other layers using the pre-trained model on ImageNet. Three FC-layers and two LeakyReLU layers make up the discriminators. [52] Superpixels are employed throughout the data preparation phase. A super pixel is a tiny area made up of several nearby pixels that are comparable in terms of brightness, colour, texture, and other attributes. For subsequent picture segmentation, the majority of these tiny areas still contain useful information. The SLIC (simple linear iterative clustering) algorithm developed by Achanta et al. is used to generate superpixels during the preprocessing step of the dataset. 1. Pre-segment superpixels with a certain number of superpixels and a predetermined size and distance from each other. 2. In the 3*3 field of the seed point, choose the seed point once more. The selection approach involves calculating the gradient values of each pixel point in the field before moving the seed point to the location in the field with the least gradient. 3. To select the category of the pixel at first, search the area surrounding each seed point and give each pixel point a labelled category. 4. Determine the separation between each detected pixel point and the original seed point. 5. Choose the closest seed point to serve as the clustering centre for each pixel. 6. Continue iterating until the pixel category is no longer altered or the number of iterations is reached. Accuracy of this model is nearly .95

2 Foundations

2.1 Convolutional Neural Network

A Convolutional neural network (CNN) is a type of deep neural network mainly used for image recognition, image classification, and object recognition tasks. It is inspired by how the brain's visual cortex processes visual information.

A CNN has multiple layers that perform specific operations on the input data. The first layer is usually a convolutional layer, where the network applies a series of filters or kernels to the input image to extract features. Each filter convolves over the input image and produces a feature map that emphasizes specific patterns or features. The next layer is usually a pooling layer that reduces the spatial dimension of the feature map by downsampling the image. This helps reduce the computational complexity of the network and prevent overfitting.

After the pooling layer, there may be additional convolutional and pooling layers that progressively extract more complex features from the input image. Finally, the output of the convolutional layers is flattened and passed through one or more fully connected layers that classify the image into one or more categories.

CNNs can be trained using supervised learning. In this case, the network learns to classify images based on labeled training data. During training, the network adjusts the filter parameters to minimize the error between the predicted output and the ground truth labels.

Overall, CNNs have become an indispensable tool for computer vision tasks, showing state-of-the-art performance on many benchmark data sets.

2.1.1 Convolutional Layer

A convolutional layer is a fundamental building block of Convolutional Neural Networks (CNNs) used for image and video processing tasks. It performs a convolution operation on the input data and applies filters, also known as kernels, to extract features from the input image.

The convolution operation involves sliding a filter over the input image, computing the dot product between the filter and the local patch of the image that is covered by the filter at each position, and producing a feature map. By applying multiple filters, the convolutional layer can learn to detect various features, such as edges, corners, and textures, at different scales and orientations.

The output of a convolutional layer is a stack of feature maps, each representing the response of a specific filter to the input image. These feature maps are then passed to the next layer in the CNN, such as a pooling layer or a fully connected layer, for further processing.

Convolutional layers have several key advantages over fully connected layers in traditional neural networks. First, they can capture local spatial dependencies in the input data, which is important for image and video processing tasks. Second, they reduce the number of parameters in the network by sharing weights across the different locations of the input, making the model more efficient and easier to train. Third, they can learn hierarchical representations of the input data by stacking multiple convolutional layers, enabling the network to capture increasingly abstract features as the depth of the network increases.

Overall, convolutional layers are an essential component of CNNs and have enabled significant advances in various computer vision tasks, including image classification, object detection, and segmentation.

2.1.2 Convolution Filters

Convolution filters are an important technique used in chest X-ray image analysis for tasks such as feature extraction, object detection, and image enhancement. They function by sliding a kernel or matrix over the image, computing the dot product of the kernel with the corresponding pixels in the image, and generating a new output image based on the computed values. These filters can extract different features, such as edges and textures to improve the quality of the image through noise reduction and contrast enhancement. Additionally, convolution filters can detect objects such as nodules, tumors and other abnormalities in the image by highlighting specific regions of the image. The working of

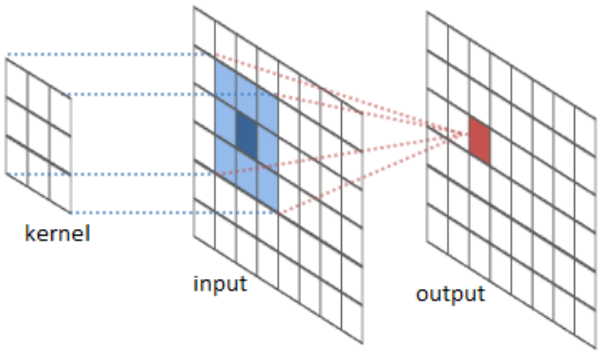


Figure 1: Convolution filtering

convolution filters in chest X-ray can be explained as follows:

- Convolution operation: Convolution filters work by performing a convolution operation on the chest X-ray image. This operation involves sliding a small matrix or kernel over the image, computing the dot product of the kernel with the corresponding pixels in the image, and generating a new output image based on the computed values.
- Feature extraction: Convolution filters can be used to extract different features from the chest X-ray image. For example, edge detection filters can highlight the edges or boundaries of structures in the image, while texture analysis filters can extract the texture features of the lung tissue.
- Image enhancement: Convolution filters can also be used to enhance the quality of the chest X-ray image. For example, noise reduction filters can remove the noise from the image and improve its clarity, while contrast enhancement filters can increase the contrast between different structures in the image.
- Object detection: Convolution filters can be used for object detection in chest X-ray images. For example, a filter can be designed to detect nodules or tumors in the lung tissue by highlighting the regions of the image that exhibit certain characteristics or features.

Traditional methods such as filtered back projection or iterative reconstruction algorithms can be computationally expensive and may produce images with noise and artifacts. By contrast, convolution neural networks with filters can directly learn the mapping in X-ray projections and CT images, resulting in high-quality images with reduced noise and artifacts. Ongoing research in this field is expected to further improve the applicability and effectiveness of convolution filters and convolution neural networks in CT imaging.

Commonly used filtering techniques for X-ray are:

- Edge detection filters: These filters can be used to highlight the edges or boundaries of structures in the chest X-ray image. Examples of edge detection filters include the Sobel filter, the Prewitt filter, and the Laplacian filter.
- Gaussian filter: This filter is commonly used for noise reduction in chest X-ray images. It can also be used for smoothing the image and enhancing the contrast.
- Median filter: This filter is another type of noise reduction filter that can be used for removing salt-and-pepper noise in chest X-ray images.
- High-pass filter: This filter can be used to enhance the details in the chest X-ray image by sharpening the edges and removing the low-frequency components.
- Low-pass filter: This filter can be used to smooth the chest X-ray image and reduce the noise.
- Gradient filter: This filter can be used to extract the gradient information of the chest X-ray image, which can be useful for feature extraction and object detection.

Some of the commonly used filters are:

- Sobel filter: A Sobel filter is a gradient-based filter that is used to detect edges in an image. It calculates the gradient of the image intensity at each pixel, and then applies a smoothing function to reduce noise. The resulting image highlights edges and boundaries, making it useful for identifying features in medical images, such as tumors.
- Laplacian of Gaussian (LoG) filter: A LoG filter is a convolution filter that applies a Gaussian filter to an image to smooth it, and then calculates the Laplacian of the resulting image. The filter is used for detecting fine structures and small features in an image, such as tiny tumors or calcifications. It is particularly effective in detecting structures that have a low signal-to-noise ratio.
- Gabor filter: A Gabor filter is a type of linear filter that applies a set of Gaussian functions of different scales and orientations to an input image. It is used for analyzing and detecting texture patterns in an image. The filter is particularly useful for detecting small, repeating patterns, and has been used in medical imaging for tumor detection and segmentation.
- Wavelet filter: A wavelet filter is a type of filter that decomposes an image into a set of frequency bands, each with different scales and orientations. It is used for feature extraction and analysis in medical images and can detect structures at different scales and resolutions. The filter is useful for detecting tumors in medical images, as well as for analyzing texture and structure in different regions of an image.
- Non-local means filter: A non-local means filter is a type of filter that analyzes the similarities between different regions of an image, rather than just the local neighborhood of each pixel. It is useful for detecting subtle changes in texture and structure and can be used to identify small and low-contrast tumors in medical images.

Overall, a combination of different convolution filters and multi-modular approaches can be used to improve the accuracy of cancer detection in medical imaging. The selection of the most appropriate filtering technique and combination depends on the specific imaging modality, the type of cancer being detected, and the desired level of accuracy.

2.1.3 Pooling layer

A pooling layer is a type of layer in Convolutional Neural Networks (CNNs) used for image and video processing tasks. It is typically placed after a convolutional layer and serves to downsample the feature maps produced by the convolutional layer while preserving their essential information.

The pooling operation involves dividing the input feature map into non-overlapping sub-regions, such as 2x2 or 3x3 patches, and computing a summary statistic, such as the maximum, average, or L2 norm, for each sub-region. The output of the pooling layer is a downsampled feature map with a reduced spatial resolution but preserved key features.

Pooling layers have several benefits in CNNs. First, they reduce the dimensionality of the feature maps, which can make the network more efficient and reduce overfitting. Second, they introduce some degree of translation invariance by summarizing the information within each sub-region regardless of its location. Third, they help to capture local patterns and structures in the input data by summarizing the information within each sub-region.

There are several pooling techniques that are commonly used in deep learning:

- Max Pooling: In this technique, the maximum value within a small rectangular neighborhood of pixels is taken as the output for each corresponding rectangular region in the feature map. Max pooling has been shown to work well in practice and is often used in convolutional neural networks (CNNs).
- Average Pooling: This technique takes the average of the values within a small rectangular neighborhood of pixels as the output for each corresponding rectangular region in the feature map.
- Global Pooling: In this technique, the output is computed by taking the maximum or average value over the entire feature map, rather than within local rectangular regions. This reduces the spatial resolution of the feature map to a single value, which can be useful for classification tasks.
- L2 Pooling: This technique computes the L2 norm of the values within a small rectangular neighborhood of pixels as the output for each corresponding rectangular region in the feature map. This can help in capturing spatial correlations between feature maps.
- Fractional Max Pooling: This technique randomly selects a subset of the pixels within a rectangular neighborhood and takes the maximum value of the selected pixels as the output. This can help in reducing overfitting by adding noise to the feature map.

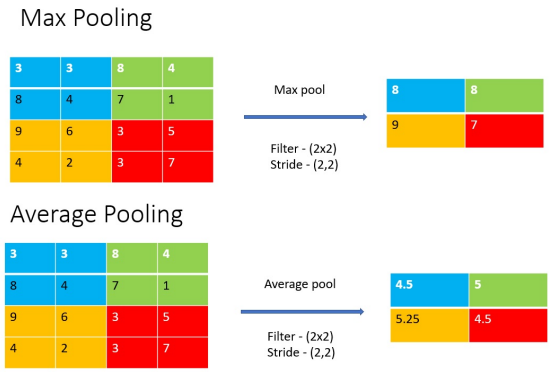


Figure 2: Max and Average Pooling

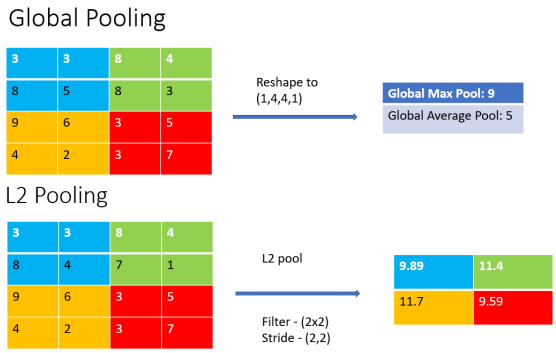


Figure 3: Global and L2 Pooling

Overall, pooling layers are important layers in CNNs for reducing the size of feature maps and improving generalization performance. The choice of pooling technique depends on the specific task and the structure of the network.

2.1.4 Fully Connected Layer

A fully connected layer is a type of layer in Convolutional Neural Networks (CNNs) that connects every neuron in the previous layer to every neuron in the current layer, similar to a traditional neural network. It is typically placed after one or more convolutional layers and one or more pooling layers, and it is responsible for producing the final output of the CNN.

The input to a fully connected layer is a flattened version of the output feature maps produced by the previous layer. The number of neurons in the fully connected layer is determined by the desired output size, which depends on the

specific task of the CNN. For example, in an image classification task with 10 classes, the fully connected layer might have 10 neurons, each corresponding to a different class.

During training, the weights of the fully connected layer are learned using backpropagation and gradient descent, just like in a traditional neural network. The loss function used for training depends on the specific task, but it typically involves a measure of the difference between the predicted output of the CNN and the true label.

Fully connected layers have several advantages in CNNs. First, they can learn complex non-linear functions that map the high-dimensional input data to the desired output. Second, they can integrate information from all the previous layers of the CNN, allowing the network to make a more informed decision based on the extracted features. Third, they can handle inputs of variable sizes by using a fixed-size output.

Overall, fully connected layers are an essential component of CNNs and have enabled significant advances in various computer vision tasks, including image classification, object detection, and segmentation. However, the use of fully connected layers in CNNs has been challenged in recent years due to their large number of parameters, which can lead to overfitting and slow training times. As a result, alternative architectures, such as fully convolutional networks and residual networks, have been proposed that use fewer fully connected layers and more convolutional layers to improve performance and efficiency.

2.2 Activation Functions

Activation functions are mathematical functions that are applied to the output of neural network nodes or a group of nodes. Their primary purpose is to introduce non-linearity into the node output, which enables the neural network to learn more complicated and abstract representations of data.

In a neural network, each node takes input from one or more preceding nodes, performs a computation, and produces an output. This output is then passed to the activation function, which applies a non-linear transformation to it. The transformed output is then forwarded to the next layer of nodes.

There are various activation functions that are typically employed in neural networks, such as the sigmoid, hyperbolic tangent, rectified linear unit (ReLU), and softmax functions. Each activation function has its own unique characteristics and is appropriate for different types of problems and network architectures.

Choosing the correct activation function is a crucial aspect of designing an effective neural network because it can significantly impact network performance.

- **Sigmoid Function:** The sigmoid function is a mathematical function that takes any input and maps it to a value ranging from 0 to 1, resulting in an S-shaped curve. It is frequently used in problems where the neural network output must be a probability, such as binary classification problems.

$$Sigmoid(x) = exp(x)/(1 + exp(x)) \tag{1}$$

- **Hyperbolic Tangent Function:** The hyperbolic tangent function is analogous to the sigmoid function, but instead of mapping the input to a value between 0 and 1, it maps it to a value ranging from -1 to 1. It is frequently employed in neural networks that have multiple hidden layers.

$$tanh(x) = (exp(x) - exp(-x))/(exp(x) + exp(-x)) \tag{2}$$

- **Rectified Linear Unit (ReLU) Function:** The Rectified Linear Unit (ReLU) function is an activation function that sets any negative input to 0 and retains positive inputs. It is a widely utilized activation function, particularly in deep neural networks, due to its effectiveness.

$$f(x) = max(0, x) \tag{3}$$

- **Leaky ReLU Function:** The Leaky ReLU function is an activation function that is comparable to the ReLU function, but it permits a small negative output for negative inputs. This can aid in preventing "dead" neurons in the network and improving the training process.

$$f(x) = max(0, x) \tag{4}$$

- **Softmax Function:** The Softmax function is frequently utilized in multi-class classification tasks where the output of the neural network is expected to represent a probability distribution over multiple classes.

$$softmax(Z)_i = e_i^z / \sum_{j=1}^{N} e_j^z$$

(5)

- **Exponential Linear Unit (ELU) Function:** The Exponential Linear Unit (ELU) function is an activation function that is comparable to the ReLU function, but it allows negative values to have a small, non-zero output. This can aid in preventing vanishing gradients and enhancing the training process.

The selection of an activation function is reliant on the specific problem being addressed and the structure of the neural network. Varying activation functions can notably affect the performance and training duration of a neural network.

Activation functions have several advantages in neural networks, including:

- **Non-linearity:** The role of activation functions is to bring non-linear transformations to the output of the neural network. This non-linearity facilitates the network to grasp more complicated and abstract representations of the input data.
- **Improved performance:** Selecting the appropriate activation function for a neural network is crucial as it can greatly impact the network's performance. Choosing the right activation function can improve the speed and accuracy of the network, leading to better results.
- **Gradient propagation:** Activation functions aid in the backpropagation process, where the neural network weights are updated during training, by facilitating the propagation of gradients through the network.
- **Versatility:** Neural networks offer a diverse set of activation functions, each with their own benefits and distinctive characteristics. This enables the customization of neural networks to a broad range of problem domains and applications.
- **Compatibility with deep learning:** Activation functions play a crucial role in deep neural networks, which have the ability to learn intricate data representations and achieve cutting-edge performance on diverse tasks.

Overall, activation functions play a critical role in the performance and functionality of neural networks, and are essential for achieving state-of-the-art results in many applications of deep learning.

While activation functions have many advantages in neural networks, there are also some potential disadvantages, including:

- **Vanishing and exploding gradients:** Some activation functions, such as the sigmoid function, can lead to vanishing or exploding gradients, which can make training difficult or impossible.
- **Limited range of output:** Some activation functions, such as the ReLU function, have a limited range of output and can lead to "dead" neurons, where the output is always zero, during training.
- **Computational complexity:** Some activation functions, such as the softmax function, can be computationally expensive to compute, especially in large neural networks.
- **Limited interpretability:** The outputs of some activation functions, such as the ReLU function, can be difficult to interpret or visualize, which can make it harder to understand the behavior of the neural network.
- **Not suitable for all types of data:** Some activation functions may not be suitable for certain types of data, such as negative values or highly skewed distributions, which can limit their usefulness in some applications.

It is important to carefully consider the advantages and disadvantages of different activation functions when designing and training neural networks, and to choose the appropriate activation function for the specific problem domain and architecture of the network.

2.3 Optimizers

Optimizers are algorithms used in deep learning to adjust the parameters of a model during training. These algorithms aim to minimize the loss function, which measures how well the model is performing on the training data. There are several optimizers used in deep learning, and each has its own strengths and weaknesses. Here are some of the most commonly used optimizers:

1. **Adam** is an optimizer that is commonly used in deep learning for computer vision tasks. It stands for Adaptive Moment Estimation, and it combines two important concepts in optimization: adaptive learning rates and momentum.

   The basic idea behind Adam is to calculate an adaptive learning rate for each parameter of the neural network. This means that each parameter has its own learning rate that is updated over time based on the history of the gradients. If a parameter has a large gradient, the learning rate for that parameter will be reduced, and if it has a small gradient, the learning rate will be increased. This helps the optimizer to converge faster and more efficiently than traditional gradient descent algorithms.

   Adam also includes a momentum term, which allows the optimizer to move in the direction of the previous gradient updates. This helps the optimizer to escape from local minima and saddle points, which can prevent the network from converging to the global minimum of the loss function.

   The algorithm for Adam can be broken down into several steps:

   (a) Initialize the parameters and hyperparameters: This includes setting the learning rate, momentum, and other hyperparameters.
   (b) Calculate the gradients: Calculate the gradients of the loss function with respect to each parameter.
   (c) Update the moving averages: Calculate the moving averages of the gradient and the squared gradient. This helps to estimate the first and second moments of the gradients, which are used to calculate the adaptive learning rates.
   (d) Calculate the adaptive learning rates: Calculate the adaptive learning rates for each parameter based on the moving averages of the gradients.
   (e) Update the parameters: Update the parameters using a combination of the adaptive learning rates and the momentum.
   (f) Repeat steps 2-5 until convergence: Keep updating the parameters until the loss function converges to a minimum.

   Overall, Adam is a powerful optimizer that is able to handle a wide range of problems in deep learning. It is particularly effective for problems with large amounts of data and many parameters, where traditional gradient descent algorithms can struggle to converge. By using adaptive learning rates and momentum, Adam is able to converge faster and more efficiently than traditional gradient descent algorithms, while also being robust to noisy gradients and other optimization challenges.

2. **SGD** is a commonly used optimizer in deep learning for computer vision tasks. It is a simple algorithm that updates the weights of the neural network in the opposite direction of the gradient of the loss function with respect to the weights. The basic idea behind SGD is to iteratively adjust the weights of the network in order to minimize the loss function.

   The algorithm for SGD can be broken down into several steps:

   (a) Initialize the weights: Set the initial weights of the neural network to small random values.
   (b) Calculate the gradients: Calculate the gradients of the loss function with respect to each weight in the network.
   (c) Update the weights: Update each weight in the network using the following formula:

   $$weight = weight - learning_rate * gradient \tag{6}$$

   where the learning rate is a hyperparameter that determines the step size of the updates. The gradient is the derivative of the loss function with respect to the weight.
   (d) Repeat steps 2-3 for multiple iterations: Keep calculating the gradients and updating the weights until the loss function converges to a minimum.

   One of the main advantages of SGD is its simplicity. It is easy to understand and implement, and it can be used for a wide range of problems in deep learning. Another advantage of SGD is that it can work well for small datasets or for problems where the data is already preprocessed and normalized.

   However, there are also some disadvantages of SGD. One of the main challenges of SGD is that it can get stuck in local minima, which can prevent the network from reaching the global minimum of the

loss function. Another challenge is that the learning rate can be difficult to tune, and if it is set too high, the optimizer can overshoot the minimum of the loss function and diverge.

Overall, SGD is a simple and effective optimizer that is a good choice for many problems in deep learning. It can be a good starting point for optimization and can often provide good results with minimal tuning. However, it may not be the best choice for all problems, and it is important to experiment with other optimizers and hyperparameters to achieve optimal results.

3. **Adagrad** is an optimizer that is commonly used in deep learning for computer vision tasks. It stands for Adaptive Gradient, and it is designed to automatically adjust the learning rate for each parameter of the neural network. The basic idea behind Adagrad is to reduce the learning rate for parameters that have high gradients, and increase the learning rate for parameters that have low gradients.

   The algorithm for Adagrad can be broken down into several steps:

   (a) Initialize the parameters and hyperparameters: This includes setting the learning rate and other hyperparameters.
   (b) Calculate the gradients: Calculate the gradients of the loss function with respect to each parameter.
   (c) Calculate the accumulated gradients: Calculate the sum of the squared gradients for each parameter over all iterations up to the current iteration. This is used to adaptively adjust the learning rate for each parameter.
   (d) Calculate the adaptive learning rates: Calculate the adaptive learning rates for each parameter using the accumulated gradients. The adaptive learning rate for a parameter is the learning rate divided by the square root of the accumulated gradient for that parameter.
   (e) Update the parameters: Update the parameters using the adaptive learning rates.
   (f) Repeat steps 2-5 until convergence: Keep updating the parameters until the loss function converges to a minimum.

   Adagrad is a powerful optimizer that is able to handle a wide range of problems in deep learning. It is particularly effective for problems with sparse data or parameters, where other optimization algorithms can struggle. By using adaptive learning rates that are specific to each parameter, Adagrad is able to converge faster and more efficiently than traditional gradient descent algorithms, while also being robust to noisy gradients and other optimization challenges.

   Overall, Adagrad is a good choice for many problems in deep learning. It can be a good starting point for optimization and can often provide good results with minimal tuning. However, it may not be the best choice for all problems, and it is important to experiment with other optimizers and hyperparameters to achieve optimal results.

4. **RMSprop** is an optimizer that is commonly used in deep learning for computer vision tasks. It stands for Root Mean Square Propagation, and it is designed to adaptively adjust the learning rate for each parameter of the neural network. The basic idea behind RMSprop is to reduce the learning rate for parameters that have high gradients and increase the learning rate for parameters that have low gradients.

   The algorithm for RMSprop can be broken down into several steps:

   (a) Initialize the parameters and hyperparameters: This includes setting the learning rate, decay rate, and epsilon.
   (b) Calculate the gradients: Calculate the gradients of the loss function with respect to each parameter.
   (c) Calculate the moving average of the squared gradients: Calculate the moving average of the squared gradients for each parameter:

   $$cache = decay_rate * cache + (1 - decay_rate) * gradient^2 \tag{7}$$

   where the decay rate is a hyperparameter that determines the weight of the old gradients in the moving average, and the epsilon is a small value added to the denominator to avoid division by zero.
   (d) Calculate the adaptive learning rates: Calculate the adaptive learning rates for each parameter using the moving average of the squared gradients. The adaptive learning rate for a parameter is the learning rate divided by the square root of the moving average of the squared gradients for that parameter.
   (e) Update the parameters: Update the parameters using the adaptive learning rates.
   (f) Repeat steps 2-5 until convergence: Keep updating the parameters until the loss function converges to a minimum.

RMSprop is a powerful optimizer that is able to handle a wide range of problems in deep learning. By using adaptive learning rates that are specific to each parameter, RMSprop is able to converge faster and more efficiently than traditional gradient descent algorithms, while also being robust to noisy gradients and other optimization challenges.

Overall, RMSprop is a good choice for many problems in deep learning. It can be a good starting point for optimization and can often provide good results with minimal tuning. However, it may not be the best choice for all problems, and it is important to experiment with other optimizers and hyperparameters to achieve optimal results.

5. **AdamW** is a variant of the Adam optimizer that is commonly used in deep learning for computer vision tasks. It stands for Adaptive Moment Estimation Weight Decay, and it is designed to adaptively adjust the learning rate for each parameter of the neural network while also adding weight decay regularization to prevent overfitting.

   The algorithm for AdamW can be broken down into several steps:

   (a) Initialize the parameters and hyperparameters: This includes setting the learning rate, weight decay factor, beta1, beta2, and epsilon.
   (b) Calculate the gradients: Calculate the gradients of the loss function with respect to each parameter.
   (c) Update the moving averages of the first and second moments: Calculate the moving averages of the first and second moments of the gradients for each parameter:

   $$m = beta1 * m + (1 - beta1) * gradient \tag{8}$$

   $$v = beta2 * v + (1 - beta2) * gradient^2 \tag{9}$$

   where m and v are the moving averages, and beta1 and beta2 are hyperparameters that determine the weight of the old gradients in the moving averages.
   (d) Calculate the adaptive learning rates: Calculate the adaptive learning rates for each parameter using the moving averages of the first and second moments. The adaptive learning rate for a parameter is the learning rate divided by the square root of the second moment of the gradient for that parameter, plus a small value (epsilon) to avoid division by zero.
   (e) Add weight decay: Add weight decay regularization to the gradients. This is done by subtracting a small fraction of the weight from the gradient. The weight decay factor determines the amount of weight decay.
   (f) Update the parameters: Update the parameters using the adaptive learning rates and the regularized gradients.
   (g) Repeat steps 2-6 until convergence: Keep updating the parameters until the loss function converges to a minimum.

   AdamW is a powerful optimizer that is able to handle a wide range of problems in deep learning. By using adaptive learning rates that are specific to each parameter, AdamW is able to converge faster and more efficiently than traditional gradient descent algorithms. Additionally, the weight decay regularization helps prevent overfitting, which is a common problem in deep learning.

   Overall, AdamW is a good choice for many problems in deep learning. It can be a good starting point for optimization and can often provide good results with minimal tuning. However, it may not be the best choice for all problems, and it is important to experiment with other optimizers and hyperparameters to achieve optimal results.

6. **NAdam (Nesterov-accelerated Adaptive Moment Estimation)** is a variant of the Adam optimizer that is commonly used in deep learning for computer vision tasks. It combines the adaptive learning rates of Adam with the Nesterov momentum, which is a technique for accelerating gradient descent by using a "look-ahead" gradient update.

   The algorithm for NAdam can be broken down into several steps:

   (a) Initialize the parameters and hyperparameters: This includes setting the learning rate, beta1, beta2, and epsilon.
   (b) Calculate the gradients: Calculate the gradients of the loss function with respect to each parameter.

(c) Update the moving averages of the first and second moments: Calculate the moving averages of the first and second moments of the gradients for each parameter:

$$m = beta1 * m + (1 - beta1) * gradient \tag{10}$$

$$v = beta2 * v + (1 - beta2) * gradient^2 \tag{11}$$

where m and v are the moving averages, and beta1 and beta2 are hyperparameters that determine the weight of the old gradients in the moving averages.

(d) Calculate the Nesterov momentum: Calculate the Nesterov momentum, which is a "look-ahead" gradient update that improves the convergence rate of the optimizer:

$$m\_hat = beta1 * m + (1 - beta1) * gradient \tag{12}$$

$$v\_hat = beta2 * v + (1 - beta2) * gradient^2 \tag{13}$$

$$m\_bar = beta1 * m\_hat + (1 - beta1) * gradient \tag{14}$$

$$v\_bar = beta2 * v\_hat + (1 - beta2) * gradient^2 \tag{15}$$

$$nesterov_momentum = -((1 - beta1) * gradient + beta1 * m\_bar)/(sqrt(v\_bar) + epsilon) \tag{16}$$

where m_hat and v_hat are the look-ahead versions of the moving averages, and m_bar and v_bar are the updated moving averages that include the Nesterov momentum.

(e) Calculate the adaptive learning rates: Calculate the adaptive learning rates for each parameter using the moving averages of the first and second moments. The adaptive learning rate for a parameter is the learning rate divided by the square root of the second moment of the gradient for that parameter, plus a small value (epsilon) to avoid division by zero.

(f) Update the parameters: Update the parameters using the adaptive learning rates, the regular gradients, and the Nesterov momentum.

(g) Repeat steps 2-6 until convergence: Keep updating the parameters until the loss function converges to a minimum.

NAdam is a powerful optimizer that is able to handle a wide range of problems in deep learning. By using adaptive learning rates that are specific to each parameter and incorporating the Nesterov momentum, NAdam is able to converge faster and more efficiently than traditional gradient descent algorithms.

Overall, NAdam is a good choice for many problems in deep learning. It can be a good starting point for optimization and can often provide good results with minimal tuning. However, it may not be the best choice for all problems, and it is important to experiment with other optimizers and hyperparameters to achieve optimal results.

2.4 Batch Normalization

Batch normalization is a method that is utilized in deep neural networks to boost the efficiency and stability of the training process. It normalizes the input to every layer of the network by utilizing the mean and variance of the inputs for the current batch of training examples.

Batch normalization is a technique that aims to alleviate the problem of internal covariate shifts in deep neural networks. This occurs when the distribution of activations within the network changes as the parameters of the preceding layers are updated during training. The normalization of inputs performed by batch normalization helps to mitigate this problem, resulting in a more stable and efficient optimization process.

During training, batch normalization computes the mean and variance of the inputs to each layer over the current mini-batch of training examples. It then normalizes the inputs using the mean and variance and applies a scaling and shifting operation to the normalized inputs to allow the network to learn the optimal values for these parameters.

Batch normalization has several benefits, including:

- **Improved training speed:** Batch normalization can decrease the number of iterations required to train a network by decreasing the internal covariate shift.
- **Improved stability:** Batch normalization helps to prevent the vanishing and exploding gradient problems that can occur in deep neural networks.

- **Improved generalization:** Batch normalization can help to reduce overfitting by adding noise to the network inputs during training.

Batch normalization is a widely used technique in deep learning, and is often included in the architecture of state-of-the-art neural networks.

Batch normalization is a technique used in deep learning neural networks to improve their performance by normalizing the input values to each layer. There are several types of batch normalization, including:

- **Batch normalization (BN):** This is the standard form of batch normalization that normalizes the activations of a layer using the mean and variance of the current batch.
- **Layer normalization (LN):** In layer normalization, the activations are normalized across the feature dimension of each sample, rather than across the batch dimension. This is particularly useful in recurrent neural networks, where the batch size can be small.
- **Instance normalization (IN):** In instance normalization, the activations are normalized across the spatial dimensions of each sample, rather than across the batch or feature dimensions. This is often used in image generation and style transfer tasks.
- **Group normalization (GN):** Group normalization divides the activations of a layer into groups and normalizes each group independently. This is useful when the batch size is small or the spatial dimensions are large.
- **Switchable normalization (SN):** Switchable normalization allows the network to choose between different types of normalization depending on the task. It can switch between batch normalization, layer normalization, and instance normalization.
- **Weight normalization (WN):** In weight normalization, the weights of the layer are normalized rather than the activations. This helps to stabilize the training process and improve generalization.

Each type of batch normalization has its own advantages and disadvantages and can be applied in different scenarios depending on the specific requirements of the task.

Batch normalization (BN) is a technique used in deep-learning neural networks to improve their performance. Some of the advantages of batch normalization include:

- **Improved training speed:** Batch normalization works by normalizing the inputs to each layer of a deep neural network based on the mean and variance of the inputs over the current mini-batch of training examples. This helps to reduce the internal covariate shift, which is the change in the distribution of network activations caused by updating the parameters of the preceding layers during training. This normalization process makes the optimization process more stable and efficient, leading to a reduction in the number of iterations required to train the network and achieve convergence.
- **Improved model generalization:** Batch normalization regularizes the model by reducing overfitting and improving its ability to generalize to new data.
- **Better gradient flow:** By reducing the internal covariate shift, batch normalization improves the flow of gradients during backpropagation, which helps to stabilize the training process and avoid vanishing or exploding gradients.
- **Increased learning rates:** Batch normalization allows for higher learning rates during training, which can further speed up the convergence process.
- **Robustness to different types of initialization:** Batch normalization reduces the sensitivity of the model to the choice of initialization values, making it more robust to different initialization methods.
- **Reducing dependence on batch size:** Batch normalization reduces the dependence of the model's performance on the batch size, making it easier to train on small datasets.

Overall, batch normalization is a powerful technique that can improve the performance, stability, and generalization of deep learning neural networks in a variety of settings.

While batch normalization (BN) has many advantages, there are also some limitations that should be considered when using this technique in deep-learning neural networks. Some of the limitations of batch normalization include:

- **Increased memory usage:** Batch normalization requires additional memory to store the mean and variance values of each layer, which can increase the overall memory usage of the model.

- **Dependence on batch size:** Although batch normalization reduces the dependence of the model's performance on the batch size, it can still have an impact on the accuracy of the model, especially for small batch sizes.
- **Impact on model interpretability:** Batch normalization can make it more difficult to interpret the activations of the model, as the input values to each layer are no longer directly related to the original inputs.
- **Limitations for certain tasks:** Batch normalization may not be effective for certain types of tasks, such as reinforcement learning or generative models, where the distributions of inputs can change over time.
- **Limitations for certain architectures:** Batch normalization may not work well with some architectures, such as recurrent neural networks, due to the changing nature of their input distributions.
- **Complexity and computational cost:** Batch normalization increase the computational complexity of the model, as it requires additional calculations to normalize the input values to each layer.

Overall, while batch normalization has many advantages, it is important to consider the limitations and potential drawbacks of this technique when using it in deep-learning neural networks.

2.5 Loss functions

Loss functions are critical components of any statistical model as they establish an objective against which the model's performance is measured, and the parameters learned by the model are determined by minimizing a selected loss function. In essence, the choice of loss function determines the accuracy of the model's estimator. Consequently, the right loss function must be chosen to define a good prediction and eliminate poor ones. This article will delve into loss functions, their significance in validating predictions, and the several loss functions employed. Furthermore, there are various types of loss functions that depend on the problem statement that we are attempting to optimize. This article will specifically explore different loss functions employed in deep learning.

2.5.1 Loss function in Regression-based problem

1. Mean Square Error Loss

   Mean Squared Error (MSE) is a statistical metric used to measure the average squared difference between the estimated values and the actual values of a set of data points. It is commonly used to evaluate the performance of regression models, where the goal is to predict a continuous outcome variable based on one or more input variables.

   To calculate the MSE, you take the sum of the squared differences between the predicted values and the actual values and divide it by the number of data points. The formula for calculating MSE is:

   $$MSE = (1/n) * (i = 1ton)(yi - \hat{y}i)^2 \tag{17}$$

   where: n is the number of data points yi is the actual value of the i-th data point ŷi is the predicted value of the i-th data point The MSE is always non-negative, and a lower value indicates a better fit between the predicted values and the actual values. However, it is sensitive to outliers and can be influenced by extreme values. Therefore, it should be used in conjunction with other performance metrics when evaluating the performance of a model.

2. Mean Absolute Error loss

   Mean Absolute Error (MAE) loss is another commonly used loss function in machine learning, particularly for regression tasks. It measures the average absolute difference between the predicted and actual values of the target variable.

   The formula for MAE is:

   $$MAE = (1/n) * |yi - \hat{y}i| \tag{18}$$

   where n is the number of samples in the dataset, yi is the actual value of the target variable for the i-th sample, and ŷi is the predicted value of the target variable for the i-th sample.

   MAE is also a non-negative value, with a value of 0 indicating perfect predictions. Like MSE, a higher value of MAE indicates that the model's predictions are further away from the actual values.

   One advantage of MAE over MSE is that it is less sensitive to outliers in the data. This means that even if there are extreme values in the target variable, the model will still try to minimize the average absolute error across all samples.

   However, MAE is not differentiable at 0, which makes it less suitable for optimization algorithms that require the computation of gradients such as stochastic gradient descent. In such cases, other loss functions such as MSE may be more appropriate.

2.5.2 Loss function in Binary classification-based problem

- Binary Cross Entropy Loss

Binary Cross Entropy (BCE) loss is a commonly used loss function in machine learning, particularly for binary classification tasks. It measures the difference between the predicted and actual binary labels of the target variable.

The formula for BCE is:

$$BCE = -(1/n) * [yi * log(\hat{y}i) + (1 - yi) * log(1 - \hat{y}i)] \tag{19}$$

where n is the number of samples in the dataset, yi is the actual binary label of the target variable for the i-th sample (0 or 1), and ŷi is the predicted probability of the positive class (usually denoted as 1) for the i-th sample.

BCE is a non-negative value, with a value of 0 indicating perfect predictions. A higher value of BCE indicates that the model's predictions are further away from the actual binary labels.

BCE is differentiable, making it suitable for optimization algorithms that require the computation of gradients such as stochastic gradient descent.

BCE can also be generalized to multi-class classification tasks using the categorical cross-entropy loss function.

2.5.3 Loss function in Multiclass classification-based problem

1. Multiclass Cross Entropy loss

   Multiclass Cross Entropy (CE) loss is a commonly used loss function in machine learning, particularly for multiclass classification tasks. It measures the difference between the predicted and actual class labels of the target variable.

   The formula for CE is:

   $$CE = -(1/n) * [yij * log(\hat{y}ij)] \tag{20}$$

   where n is the number of samples in the dataset, yij is the actual class label of the target variable for the i-th sample and j-th class (0 or 1), and ŷij is the predicted probability of the j-th class for the i-th sample.

   CE is a non-negative value, with a value of 0 indicating perfect predictions. A higher value of CE indicates that the model's predictions are further away from the actual class labels.

   CE is differentiable, making it suitable for optimization algorithms that require the computation of gradients such as stochastic gradient descent.

   There are variants of CE such as weighted cross entropy, where the contribution of each class to the loss function can be weighted differently, and focal loss, which is designed to down-weight the contribution of easy examples and focus more on hard examples.

2. Kullback-Leibler divergence loss

   Kullback-Leibler (KL) divergence is a measure of how different two probability distributions are. In the context of machine learning, it is often used as a loss function to train models to approximate a target distribution.

   The formula for KL divergence between two probability distributions P and Q is:

   $$KL(P||Q) = P(x) * log(P(x)/Q(x)) \tag{21}$$

   where x is a value from the domain of the distributions. Intuitively, KL divergence measures the average difference in the amount of information required to encode events from P using a code based on Q, as compared to encoding the same events using a code based on P.

   KL divergence is always non-negative, with a value of 0 indicating that the two distributions are identical. A higher value of KL divergence indicates that the two distributions are more dissimilar.

   In the context of machine learning, KL divergence is often used as a regularization term in the loss function to encourage the model to produce outputs that are close to a desired distribution. It can also be used to compare the predicted probability distribution of a model with the true distribution of the target variable.

   One important note is that KL divergence is not symmetric, i.e., KL(P ‖ Q) is not the same as KL(Q ‖ P).

3. Perceptual loss

   Perceptual loss is a loss function used in deep learning that measures the difference between two images in terms of their perceptual similarity, rather than pixel-wise differences. It is often used in image-to-image translation tasks, where the goal is to convert an input image into an output image that belongs to a different domain, while preserving the high-level content and style of the input image.

Perceptual loss is based on the idea that high-level features extracted from deep neural networks can capture the perceptual similarity between images. In particular, the loss function is defined as the difference between the feature representations of the two images, obtained by passing them through a pre-trained deep neural network.

One commonly used form of perceptual loss is the mean squared error (MSE) between the feature representations of the two images at a specific layer of the neural network. For example, in the popular style transfer task, the perceptual loss is defined as the MSE between the feature representations of the input image and the output image at a certain layer of a pre-trained convolutional neural network (e.g., VGG19), which captures the style and texture information of the images.

Perceptual loss is effective in capturing the high-level content and style information of the images while being robust to small variations and distortions in the pixel values of the images. It has been successfully applied in a variety of image-to-image translation tasks, such as style transfer, super-resolution, and image colorization.

2.6 Types of CNNs

There are several types of Convolutional Neural Networks (CNNs), each with its architecture and characteristics. Some of the most common types of CNNs are:

1. Standard CNNs: These are the most basic type of CNNs that have one or more convolutional layers followed by one or more fully connected layers. They are commonly used for image classification tasks.
2. Convolutional Autoencoders: These are CNNs that are trained to reconstruct the input image from a compressed representation. They are used for image compression and denoising tasks.
3. Recurrent CNNs: These are CNNs that have a recurrent layer, such as a Long Short-Term Memory (LSTM) layer, to handle sequential data. They are used for tasks such as speech recognition and natural language processing.
4. Fully Convolutional Networks (FCNs): These are CNNs that have only convolutional layers and no fully connected layers. They are used for tasks such as image segmentation, where the goal is to assign a label to each pixel in the image.
5. Residual Networks (ResNets): These are CNNs that use residual connections to address the vanishing gradient problem that occurs in deep networks. They are commonly used for image classification tasks.
6. Inception Networks: These are CNNs that use Inception modules, which are multiple parallel convolutional layers with different filter sizes, to capture features at different scales. They are commonly used for image classification tasks.
7. Generative Adversarial Networks (GANs): These are CNNs that consist of a generator network and a discriminator network that are trained together to generate new data that is similar to the training data. They are used for tasks such as image and video synthesis.

Each type of CNN has its own strengths and weaknesses, and the choice of which type to use depends on the specific task and dataset.

2.7 Transfer Learning

Transfer learning is a machine learning technique that involves taking a pre-trained model and using it as a starting point for a new task. Rather than training a model from scratch on a large dataset, transfer learning allows us to leverage the knowledge learned from an existing model that has been trained on a similar task or dataset.

The pre-trained model is typically a deep neural network that has been trained on a large dataset, such as ImageNet, to perform a particular task, such as image classification. The model has learned to recognize a wide range of features in images, which can be transferred to a new task. The idea is that the lower-level features that are learned by the model in the early layers are general enough to be applicable to many different tasks, whereas the higher-level features in the later layers are more specific to the original task.

To use transfer learning, we typically remove the output layer of the pre-trained model and replace it with a new output layer that is specific to our task. We then freeze the weights of the pre-trained model so that they are not updated during training and only train the weights of the new output layer. This allows us to train a model on a smaller dataset while still achieving high accuracy, as the pre-trained model has already learned many useful features.

Transfer learning has been used successfully in many applications, including computer vision, natural language processing, and speech recognition. ItSeveral popular transfer learning models havehe accuracy of models on new tasks, especially when the new task has a smaller dataset than the original task.

2.7.1 Transfer learning models in Computer Vision

There are several popular transfer learning models that have been pre-trained on large datasets and can be used as a starting point for various computer vision tasks. Some of the most commonly used transfer learning models in computer vision include:

- VGG: The Visual Geometry Group (VGG) model is a deep CNN that has been pre-trained on the ImageNet dataset for image classification. It has a simple architecture and is easy to fine-tune for various tasks.
- ResNet: The Residual Network (ResNet) model is a deep CNN that uses residual connections to address the vanishing gradient problem in deep networks. It has achieved state-of-the-art results on several computer vision tasks, including image classification, object detection, and segmentation.
- Inception: The Inception model is a deep CNN that uses Inception modules, which are multiple parallel convolutional layers with different filter sizes, to capture features at different scales. It has achieved state-of-the-art results on several computer vision tasks, including image classification and object detection.
- MobileNet: The MobileNet model is a lightweight CNN that has been designed for mobile and embedded devices. It has achieved state-of-the-art results on several computer vision tasks while requiring fewer computations and memory than other models.
- DenseNet: The Dense Convolutional Network (DenseNet) model is a deep CNN that uses dense connections, where each layer is connected to every other layer in a feed-forward fashion, to improve the flow of information through the network. It has achieved state-of-the-art results on several computer vision tasks, including image classification and object detection.

These pre-trained models are widely available and can be fine-tuned or used for feature extraction for various computer vision tasks. The choice of which model to use depends on the specific task and dataset, as well as the computational resources available.

2.7.2 Transfer learning techniques

In computer vision, some of the most popular transfer learning methods include:

- Fine-tuning: This involves taking a pre-trained model and fine-tuning it on a new dataset. The weights of the pre-trained model are typically frozen for the early layers, and only the later layers are fine-tuned on the new dataset. Fine-tuning is commonly used for tasks such as image classification and object detection.
- Feature extraction: This involves using the pre-trained model to extract features from the images in the new dataset and then training a new model, such as a linear classifier or support vector machine, on these features. Feature extraction is commonly used for tasks such as image classification and image retrieval.
- Domain adaptation: This involves adapting the pre-trained model to a new domain, such as a different lighting condition or a different camera angle. Domain adaptation is commonly used for tasks such as image classification and object detection.
- Multi-task learning: This involves training a single model on multiple tasks simultaneously, such as object detection and segmentation. The pre-trained model is used as a starting point, and the model is fine-tuned on both tasks. Multi-task learning is commonly used for tasks such as image and video analysis.

These transfer learning methods have been widely used in computer vision and have shown to be effective in improving the accuracy and reducing the training time of deep learning models. The choice of which method to use depends on the specific task and dataset.

3 Methods and Models

3.1 Dataset

Our team is currently working with the COVID-19 Radiography Dataset, which we obtained from Kaggle. We have chosen to focus solely on the image folder within this dataset, as we plan to train our model using these images. Our approach involves dividing the image classes into four categories:

- COVID number of image is 3616
- Lung Opacity number of image is 6012
- Normal number of image is 10192
- Viral Pneumonia number of image is 1345

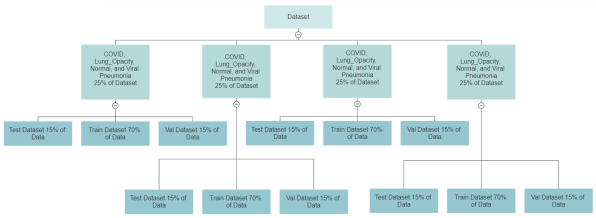


Figure 4: Data split

We will create four subsets for each of these categories by dividing the images in each class by four. This will result in four distinct datasets (i.e., dataset 1, dataset 2, dataset 3, and dataset 4), and each team member will be responsible for working on their own sub-dataset. Within each sub-dataset, we will further divide the images into training, testing, and validation sets to facilitate the training process.

3.2 Data Augmentation

Data augmentation and data generation are important techniques in machine learning and data science because they can help to increase the size and diversity of a dataset, which in turn can improve the performance of machine learning models. Data augmentation involves applying various transformations to existing data samples in order to create new, but similar, samples. This can include techniques like flipping, rotating, cropping, zooming, or adding noise to images or audio signals. By generating these new samples, the model can learn to recognize the same patterns in different contexts, and this can help to reduce overfitting and improve the model's generalization ability. Data generation, on the other hand, involves creating entirely new data samples that are similar to the original data, but not necessarily exact copies. This can be particularly useful in situations where there is limited or no real-world data available, such as in the development of synthetic datasets or simulations. By generating new data samples, it is possible to train models that are robust to a wider range of scenarios, and this can help to improve their accuracy and reliability. Overall, data augmentation and data generation are important tools for improving the quality and diversity of datasets, which can ultimately lead to better performance and more accurate predictions from machine learning models.

3.3 EfficientNet

3.3.1 Introduction

EfficientNet is a family of convolutional neural network models designed for efficient and effective deep learning. It was introduced in a paper by Mingxing Tan and Quoc Le from Google Brain in 2019.

EfficientNet achieves state-of-the-art performance on image classification benchmarks such as ImageNet while using fewer parameters and less computational resources than other models of comparable accuracy. EfficientNetV1 uses a compound scaling method that optimizes the depth, width, and resolution of the network architecture to achieve state-of-the-art performance on image classification benchmarks such as ImageNet. The scaling method involves increasing the depth and width of the network while reducing the image resolution. This balances the network architecture to achieve better model performance with fewer parameters and less computational cost. The EfficientNet family consists of several models, including EfficientNet-B0, EfficientNet-B1, EfficientNet-B2, EfficientNet-B3, EfficientNet-B4, EfficientNet-B5, and EfficientNet-B6, each with different depth, width, and resolution parameters. The larger models (B4, B5, and B6) have achieved state-of-the-art performance on ImageNet with significantly fewer parameters and computational resources than previous state-of-the-art models.

EfficientNet models are trained using a combination of supervised and unsupervised learning techniques. Supervised learning involves training the model on a large dataset of labeled images, while unsupervised learning involves pre-training the model on a larger dataset of unlabeled images, using a technique called self-supervised learning.

EfficientNet has been widely adopted in computer vision research and applications due to its impressive performance and efficiency, especially in scenarios where computational resources are limited, such as mobile and embedded devices.

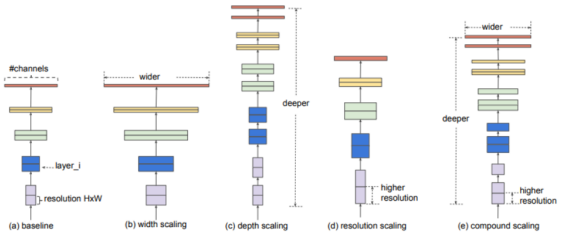


Figure 5: Different Scaling methods

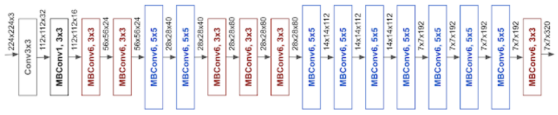


Figure 6: EfficientNetV1 architecture

3.3.2 EfficientNetV2

EfficientNet Version 2 is an improved version of the EfficientNet family of convolutional neural network models designed for efficient and effective deep learning. It was introduced in a paper by Mingxing Tan, Ruoming Pang, and Quoc Le from Google Brain in 2020.

EfficientNetV2 builds upon the success of EfficientNetV1 and introduces several new features and improvements. One of the main innovations is the use of a new compound scaling method that optimizes the depth, width, and resolution of the network architecture in a more effective way. This results in better model performance with fewer parameters and less computational cost than previous versions.

EfficientNetV2 also introduces a new mobile-friendly architecture called MobileNetV3, which is optimized for mobile and embedded devices. This architecture has significantly fewer parameters and computational costs than previous versions, making it ideal for resource-constrained environments.

EfficientNetV2 consists of several models, including EfficientNetV2-S, EfficientNetV2-M, EfficientNetV2-L, and EfficientNetV2-X, each with different depth, width, and resolution parameters. The larger models (V2-L and V2-X) have achieved state-of-the-art performance on image classification benchmarks such as ImageNet with significantly fewer parameters and computational resources than previous state-of-the-art models.

EfficientNetV2 models use a combination of supervised and unsupervised learning techniques, similar to EfficientNetV1.

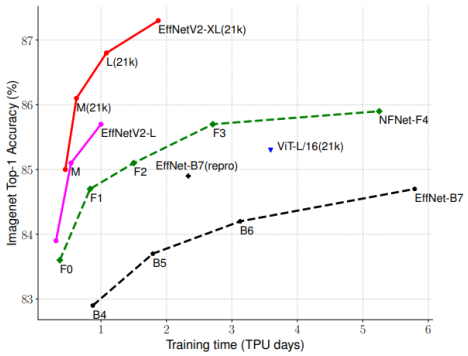
3.3.3 Improvements by EfficientNetV2

EfficientNetV2 introduced several changes to the architecture of the EfficientNet family of models, compared to EfficientNetV1. Some of the key differences between the layers of the two models are:

- Stem: The stem of EfficientNetV2 is wider and deeper than that of EfficientNetV1, which helps to extract more useful features from the input image.
- Convolutional Blocks: EfficientNetV2 uses a new type of convolutional block called the Squeeze-Excitation (SE) block, which selectively amplifies important features and suppresses unimportant ones. This helps to improve the efficiency and effectiveness of the network.
- Depthwise Separable Convolutions: EfficientNetV2 uses a combination of depthwise separable convolutions and regular convolutions, which allows for more efficient computation and parameter reduction.
- Head: EfficientNetV2 uses a new type of head architecture that is more flexible and can adapt to a wider range of tasks, compared to EfficientNetV1.

- Mobile-Friendly Architecture: EfficientNetV2 introduces a new mobile-friendly architecture called MobileNetV3, which is optimized for resource-constrained environments such as mobile and embedded devices.

Overall, EfficientNetV2 makes several improvements to the architecture of EfficientNetV1, including a more effective compound scaling method, a wider and deeper stem, the use of SE blocks, and a more flexible head architecture. These improvements result in better performance and efficiency, especially in scenarios where computational resources are limited.



(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

Figure 7: EfficientNetV2 training and parameter efficiency

3.3.4 EfficientNetV2M

EfficientNetV2M is a medium-sized model in the EfficientNetV2 family and has the following characteristics:

- Input Shape: The input shape of EfficientNetV2M is 224x224x3, which means it takes an RGB image with a resolution of 224x224 pixels.
- Number of Layers: EfficientNetV2M has a total of 19 layers, which include a stem, several convolutional blocks, and a head.
- Number of Trainable Parameters: EfficientNetV2M has approximately 17 million trainable parameters. These are the parameters that are updated during the training process to improve the performance of the model.
- Number of Non-Trainable Parameters: EfficientNetV2M has approximately 1.5 million non-trainable parameters. These are the parameters that are not updated during the training process and are usually used for normalization or regularization.

EfficientNetV2M is designed to balance the trade-off between model performance and computational efficiency. It is smaller than some of the larger models in the EfficientNetV2 family, such as EfficientNetV2-L and EfficientNetV2-X, but still achieves good performance on image classification benchmarks such as ImageNet. It is also more computationally efficient, making it ideal for resource-constrained environments such as mobile and embedded devices.

Overall, EfficientNetV2M is a medium-sized model in the EfficientNetV2 family with a relatively small number of trainable parameters and good performance on image classification benchmarks. It is a good choice for applications where computational resources are limited, but good performance is still required.

EfficientNetV2M architecture consists of several components, which are as follows:

- Stem: The stem of EfficientNetV2M is a sequence of layers that processes the input image. It includes a 3x3 convolutional layer with a stride of 2, followed by a batch normalization layer, a swish activation function, and a 3x3 depthwise separable convolutional layer.

- Convolutional Blocks: EfficientNetV2M uses several convolutional blocks, each consisting of a sequence of layers. Each block includes a combination of regular and depthwise separable convolutions, followed by a swish activation function and a squeeze-and-excitation (SE) block. The SE block selectively amplifies important features and suppresses unimportant ones, which helps to improve the efficiency and effectiveness of the network.
- Head: The head of EfficientNetV2M is a sequence of layers that produces the final output of the network. It includes a global average pooling layer, followed by a dropout layer, a fully connected layer, and a softmax activation function.
- The EfficientNetV2 architecture extensively utilizes both MBConv and the newly added Fused-MBConv in the early layers.
- EfficientNetV2 prefers small 3x3 kernel sizes as opposed to 5x5 in EfficientNetV1.
- EfficientNetV2 completely removes the last stride-1 stage as in EfficientNetV1

EfficientNetV1 introduces a new type of block called the MBConv (Mobile Inverted Residual Bottleneck Convolution) block, which is designed to improve the efficiency and effectiveness of the network by reducing the number of parameters and the computational cost. The MBConv block consists of a depthwise separable convolution, a squeeze-and-excitation block, and a linear projection. Depthwise convolutional layers (MBConv) are slow. They generally have fewer parameters than regular convolutional layers, but the problem is that they cannot fully make use of modern accelerators. To overcome this problem EfficientNetV2 uses a combination of MBConv and Fused MBConv to make the training faster without increasing the parameters

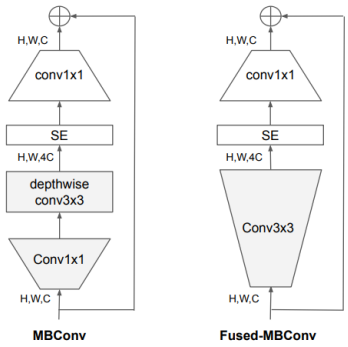


Figure 8: Structure of MBConv and Fuse-MBConv blocks

EfficientNetV2 introduces a new variant of the MBConv block called the Fused MBConv block, which combines the depthwise separable convolution and the linear projection into a single operation, reducing the computational cost and the memory footprint of the block. In Fused MBConv, the depthwise separable convolution and the linear projection are fused into a single operation using a factorized convolutional kernel. The output of the factorized convolutional kernel is then processed by a batch normalization layer, a swish activation function, and a squeeze-and-excitation block.

The main difference between MBConv and Fused MBConv is the number of operations and the memory footprint. Fused MBConv has fewer operations and requires less memory than MBConv because it fuses the depthwise separable convolution and the linear projection into a single operation. This makes Fused MBConv more computationally efficient and memory-efficient than MBConv. However, the performance of Fused MBConv may be slightly lower than that of MBConv because it uses a factorized convolutional kernel, which may not be as expressive as the separate depthwise separable convolution and linear projection.

Overall, Fused MBConv is a more efficient and memory-efficient variant of the MBConv block, which reduces the computational cost and the memory footprint of the network. It is especially useful in resource-constrained environments, such as mobile and embedded devices, where memory and computational resources are limited.

3.4 Ensemble Stacking models - DenseNet169 and MobileNetV2

3.4.1 DesnseNet Introduction

New research has indicated that convolutional neural networks can achieve improved accuracy, efficiency, and depth by incorporating short connections between nearby layers near the input and output layers.

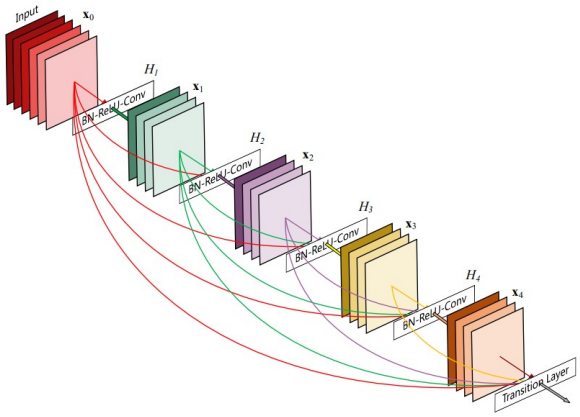


Figure 9: DenseNet Structure

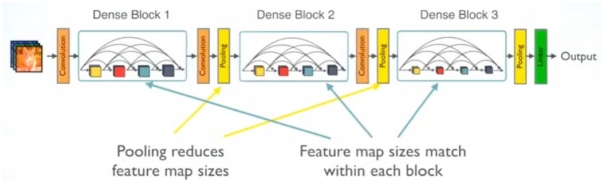


Figure 10: Dense Blocks and Layers

This study builds upon the previous observation and presents a novel dense convolutional network (DenseNet) that connects every layer to all other layers via feedforward, in contrast to traditional convolutional networks that only connect each layer to the next. Therefore, while traditional networks have L connections for L layers, DenseNet has L(L+1)/2 direct connections.

The feature maps of all preceding layers are utilized as inputs for each level in DenseNets, while its own feature maps are used as inputs for all subsequent levels. This method of connection provides several advantages, such as solving the problem of vanishing gradients, improving feature propagation, encouraging feature reuse, and significantly decreasing the number of parameters.

3.4.2 DenseNet169 Introduction

DenseNet-169 is a type of DenseNet architecture that has 169 layers and is commonly used for deep-learning classification tasks. Despite having more layers, it has fewer parameters to train than other DenseNet architectures with fewer layers. Like other DenseNet architectures, DenseNet-169 can overcome the problem of vanishing gradients, has a robust feature propagation approach, minimizes the number of training parameters, and promotes feature reuse. These properties make DenseNet models highly reliable for deep learning tasks. DenseNet models are available in popular deep learning frameworks such as TensorFlow (Keras) and PyTorch. The layered architecture of DenseNet-169 used in this study is provided.

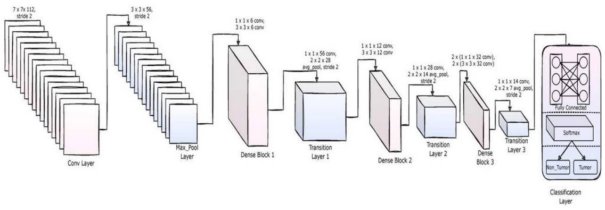
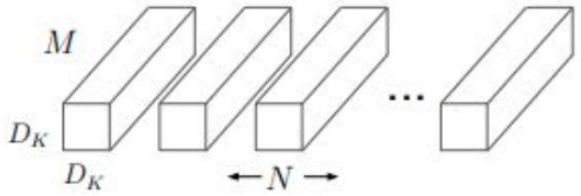


Figure 11: DenseNet 169 Structure

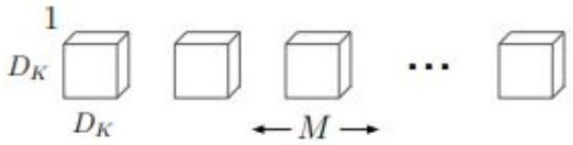
It is observed that the fine-tuned DenseNet-169 model has a better performance compared to other models and it is found to be more efficient than the normal DenseNet-169. Optimization of the feature weight made the accurate detection of lymph node metastases much better

3.4.3 MobileNet Introduction

The MobileNet is a CNN architecture that is capable of delivering powerful and portable models that can be used in real-world applications. Unlike previous architectures, MobileNets mostly employ depthwise separable convolutions to build lighter models. Additionally, MobileNets introduce two new global hyperparameters - width factor and resolution factor - which provide model developers with the flexibility to vary the speed and size of the model while also considering the latency or accuracy requirements of the application.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

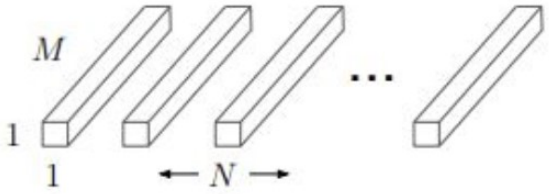


Figure 12: MobileNet Structure

MobileNets use a design based on deeply separated convolutional layers, where each layer includes both depth and point convolutions. In total, there are 28 layers if we count depth and point convolutions as separate layers. These networks typically have 4.2 million parameters, but this number can be reduced further by adjusting the bandwidth hyperparameter. Additionally, MobileNets introduce two new global hyperparameters - width factor and resolution factor - that provide developers with the ability to vary the model's speed, size, and accuracy based on specific requirements. The input image size for MobileNets is typically 224 x 224 x 3.

3.4.4 MobileNetV2 Introduction

The architecture of MobileNetV2 follows an inverse residual structure, in which the residual block's input and output comprise thin bottleneck layers, in contrast to the conventional residual models that utilize expanded representations at the input.

MobileNetV2 architecture utilizes lightweight depth-based convolutions to filter intermediate features in the extension layer. Notably, it is crucial to remove nonlinearities in the narrow layers to maintain the representational power. This design choice leads to better performance and provides insight into the intuition behind it. Moreover, it allows the decoupling of input/output domains from the transformation manifestation, providing a convenient framework for further analysis. The performance of MobileNetV2 is evaluated in several tasks such as Imagenet classification, COCO object detection, and VOC image segmentation. The evaluation considers the trade-offs between the number of operations and parameters, measured by precision and multiplication (MAdd).

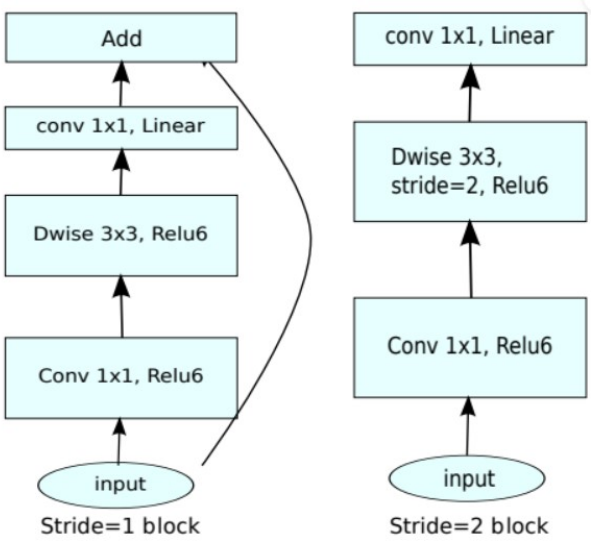


Figure 13: MobileNet V2 Structure

3.4.5 Ensemble Stacking

Ensemble techniques refer to the use of multiple models or learning algorithms to create a more optimal prediction model. This approach often outperforms the individual basic learners used on their own. Ensemble techniques can be utilized in a variety of applications, such as feature selection and information combination. They are typically categorized into bagging, boosting, and stacking.

Stacking, in particular, employs heterogeneous weak learners, which are trained alongside each other and combined by training a meta-learner to predict based on the various weak learner predictions. The meta-learner receives the predictions as functions and learns how to combine them to improve the overall prediction performance based on the ground truth values of the data.

Random Forest is an example of an intermediate setting where the model combines predictions from several trained models. However, this approach has a limitation in that each model contributes equally to the ensemble forecast, regardless of its performance. An alternative is an ensemble-weighted average, where the contribution of each member is calculated based on its effectiveness in generating the best predictions.

A further generalization of this approach is to use a learning algorithm such as linear regression or logistic regression to combine the predictions of the submodels. This technique is called stacking, where the algorithm takes the outputs of the submodels as input and learns how to optimally combine them to improve the overall prediction performance.

3.4.6 Ensemble Stacking - Densenet169 and Mobilenet V2

To preserve predefined information, freeze all middle layers and practice only on the last layers after stacking. These pre-trained stacked models are trained on the image network dataset. tf.keras.layers.Concatenate() concatenates the outputs of the models.

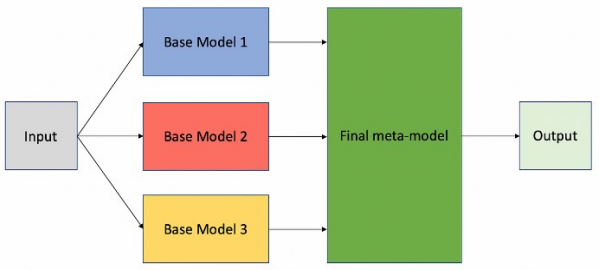


Figure 14: stacking Structure

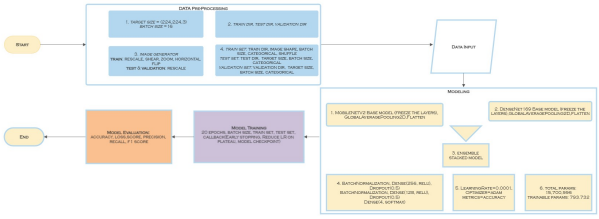


Figure 15: ensemble stacking model topology

We stack DenseNet169 and MobilenetV2 for better performance, we stack 2 models: freeze the original layers and stack the output.

3.4.7 Machine setup

System details below:

Processor	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
RAM	8.00 GB (7.84 GB usable)
System type	64-bit operating system, x64-based processor
OS	Windows 11 Home Single Language (version:21H2, build: 22000.1335, experience: Windows Feature Experience Pack 1000.22000.1335.0)

Figure 16: Machine set up

3.5 VGG16

A Convolutional Neural Network, also referred to as ConvNet, is a type of artificial neural network that typically includes an input layer, an output layer, and multiple hidden layers. VGG16 is an example of a CNN and is widely recognized as one of the top-performing computer vision models available.

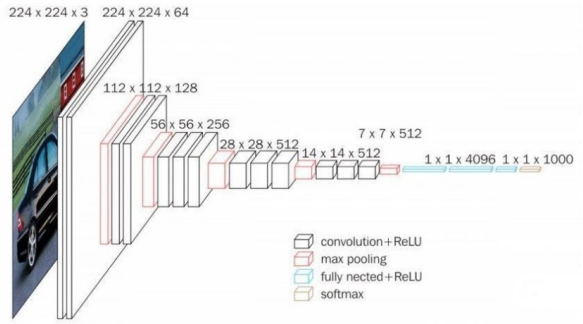


Figure 17: VGG16 architecture

The architects of this model increased the depth of their networks by using a design with small 3x3 convolutional filters, which outperformed previous designs. They expanded the depth to 16-19 weight layers, resulting in around 138 trainable parameters.

VGG16 is a classification algorithm that can identify 1000 images from 1000 different categories with an accuracy of 92.7

The number 16 in VGG16 represents a layer with 16 weights. VGG16 has a total of 21 layers, including thirteen convolution layers, five max pooling layers, and three dense layers. However, it only has sixteen weight layers, which are the learning parameter layers.

VGG16 uses an input tensor with dimensions of 224, 244, and 3 RGB channels. What sets VGG16 apart is that it emphasizes convolutional layers with a 3x3 filter in the first stage, and always uses the same padding and max pooling

layer on the 2x2 filter. The inverse and maximum pool layers are placed throughout the architecture. The Conv-1 layer has 64 filters, Conv-2 has 128 filters, Conv-3 has 256 filters, and Conv-4 and Conv-5 have 512 filters. Following a stack of convolutional layers, three fully connected (FC) layers are present: the first two have 4096 channels each, the third has 1000 channels for ILSVRC classification. The last layer is the soft-max layer.

3.5.1 VGG16 - last 4 layer trainable

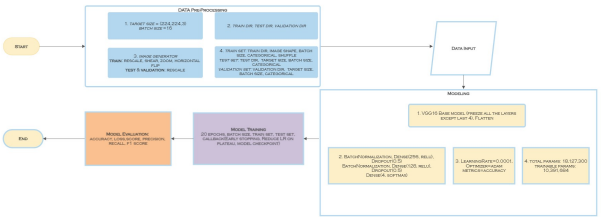


Figure 18: VGG16 model topology

To preserve the preset information, freeze all levels except the last 4 levels for practice, and only practice the upper levels after freezing the lower levels.

This pre-trained model is pre-trained on the image network dataset.

3.5.2 Machine setup

System details below:

Processor	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
RAM	8.00 GB (7.84 GB usable)
System type	64-bit operating system, x64-based processor
OS	Windows 11 Home Single Language (version:21H2, build: 22000.1335, experience: Windows Feature Experience Pack 1000.22000.1335.0)

Figure 19: Machine set up

3.6 AlexNet

AlexNet is a deep convolutional neural network architecture designed for image classification tasks. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, and it was the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in the same year. AlexNet consists of eight layers, including five convolutional layers and three fully connected layers. The first two convolutional layers have a filter size of 11x11 with a stride of 4, followed by a max-pooling layer with a filter size of 3x3 and a stride of 2. The next two convolutional layers have a filter size of 5x5 with a stride of 1, followed by another max-pooling layer with the same filter size and stride. Finally, the last convolutional layer has a filter size of 3x3 with a stride of 1. The fully connected layers have 4,096 neurons each. One of the key innovations in AlexNet is the use of ReLU (Rectified Linear Unit) activation functions, which

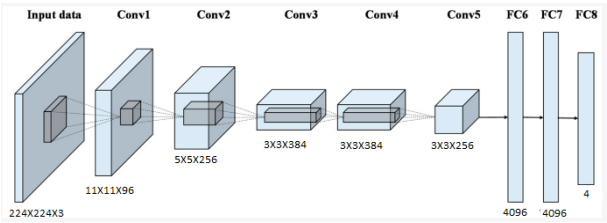


Figure 20: AlexNet architecture

improve the convergence of the model during training. AlexNet also uses dropout regularization to prevent overfitting. AlexNet achieved state-of-the-art performance on the ImageNet dataset, significantly outperforming previous methods.

Its success paved the way for the development of deeper and more powerful neural network architectures, such as VGG, ResNet, and Inception.

The AlexNet architecture consists of eight layers, including five convolutional layers and three fully connected layers. Here's a breakdown of each layer:

- Input layer: The input layer accepts images with a size of 227x227x3 (RGB images).
- Convolutional layer 1: This layer has 96 filters with a size of 11x11, a stride of 4, and a padding of size 0. The output size of this layer is 55x55x96.
- Max-pooling layer 1: This layer performs max-pooling over a 3x3 window with a stride of 2. The output size of this layer is 27x27x96
- Convolutional layer 2: This layer has 256 filters with a size of 5x5, a stride of 1, and a padding of size 2. The output size of this layer is 27x27x256
- Max-pooling layer 2: This layer performs max-pooling over a 3x3 window with a stride of 2. The output size of this layer is 13x13x256
- Convolutional layer 3: This layer has 384 filters with a size of 3x3, a stride of 1, and a padding of size 1. The output size of this layer is 13x13x384.
- Convolutional layer 4: This layer has 384 filters with a size of 3x3, a stride of 1, and a padding of size 1. The output size of this layer is 13x13x384.
- Convolutional layer 5: This layer has 256 filters with a size of 3x3, a stride of 1, and a padding of size 1. The output size of this layer is 13x13x256.
- Max-pooling layer 3: This layer performs max-pooling over a 3x3 window with a stride of 2. The output size of this layer is 6x6x256.
- Flatten layer: This layer flattens the output of the previous layer into a 1D array with size 9216.
- Fully connected layer 1: This layer has 4096 neurons with ReLU activation.
- Dropout layer 1: This layer randomly drops out 50
- Fully connected layer 2: This layer has 4096 neurons with ReLU activation.
- Dropout layer 2: This layer randomly drops out 50
- Fully connected layer 3: This layer has 4 neurons (corresponding to the 4 classes in the ImageNet dataset) with softmax activation.

3.7 InceptionV3

InceptionV3 is a deep convolutional neural network architecture for image classification tasks. It was developed by Google researchers in 2015 as an improvement over its predecessor, InceptionV1, and it achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015. The InceptionV3 architecture consists of 48 layers, including 17 inception modules, which are blocks of layers designed to extract features at different scales. The key innovation in InceptionV3 is the use of "Inception modules," which combine filters of different sizes in parallel to extract features at different scales. InceptionV3 also incorporates a number of other techniques to improve performance, including the use of batch normalization, factorized convolutions, and aggressive data augmentation. The architecture of InceptionV3 can be broken down into the following key layers:

- Input layer: The input layer accepts images with a size of 224x224x3 (RGB images).
- Convolutional layers: The input is first passed through a series of convolutional layers with different filter sizes and strides, followed by max-pooling layers to reduce the spatial size of the feature maps.
- Inception modules: Inception modules are designed to extract features at different scales. Each inception module consists of several parallel convolutional layers with different filter sizes and pooling operations. The outputs of these parallel layers are concatenated along the depth dimension.
- Average pooling: The output of the final inception module is passed through a global average pooling layer to reduce the dimensionality of the feature maps.
- Fully connected layer: The output of the global average pooling layer is flattened and passed through a fully connected layer with softmax activation to produce the final class probabilities. InceptionV3 achieved state-of-the-art performance on the ImageNet dataset and has been widely adopted as a baseline architecture for a variety of computer vision tasks, including object detection, semantic segmentation, and image captioning.

3.8 DenseNet121

DenseNet121 is a convolutional neural network architecture designed for image classification tasks. It was introduced in 2017 by researchers at Facebook AI Research and has achieved state-of-the-art performance on a number of benchmark datasets.

The key innovation of the DenseNet121 architecture is the use of dense blocks, which are blocks of layers that are densely connected to each other. In a dense block, the output of each layer is concatenated with the output of all previous layers, which creates a "highway" of information flow between the layers. This dense connectivity has been shown to improve the flow of information through the network and to reduce the number of parameters needed for training.

The architecture of DenseNet121 can be broken down into the following key layers:

- Input layer: The input layer accepts images with a size of 224x224x3 (RGB images).
- Convolutional layers: The input is passed through a series of convolutional layers with different filter sizes and strides, followed by max-pooling layers to reduce the spatial size of the feature maps.
- Dense blocks: Dense blocks are designed to extract features at different scales. Each dense block consists of several convolutional layers that are densely connected to each other.
- Transition layers: Transition layers are used to reduce the dimensionality of the feature maps between dense blocks. They consist of a batch normalization layer, a 1x1 convolutional layer, and a max-pooling layer.
- Average pooling: The output of the final dense block is passed through a global average pooling layer to reduce the dimensionality of the feature maps.
- Fully connected layer: The output of the global average pooling layer is flattened and passed through a fully connected layer with softmax activation to produce the final class probabilities.

DenseNet121 has achieved state-of-the-art performance on a number of benchmark datasets, including ImageNet, and has been widely adopted as a baseline architecture for a variety of computer vision tasks, including object detection, semantic segmentation, and image captioning.

3.8.1 Machine setup

System details below:

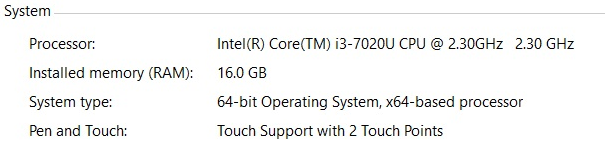


Figure 21: Machine set up

3.9 ResNet-50

3.9.1 Introduction

ResNet-50 is a deep convolutional neural network architecture that was introduced by Microsoft researchers in 2015 as part of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The ResNet-50 architecture is a variant of the original ResNet architecture, which stands for "Residual Network". The key innovation of ResNet is the use of residual connections, which allow information to be passed forward through a shortcut connection that bypasses one or more layers.

The ResNet-50 architecture consists of 50 layers and is comprised of a series of convolutional layers, followed by several residual blocks. Each residual block consists of multiple convolutional layers and a shortcut connection that bypasses the layers in the block. This shortcut connection allows the model to learn more easily by reducing the vanishing gradient problem, which is a common issue in deep neural networks.

ResNet-50 has been shown to be highly effective for image classification tasks, achieving state-of-the-art performance on a range of benchmark datasets. It has also been used as a feature extractor in transfer learning, where the model is

pretrained on a large dataset and then fine-tuned on a smaller, specific dataset. This approach has been shown to be effective for a range of computer vision tasks, including object detection, image segmentation, and facial recognition.

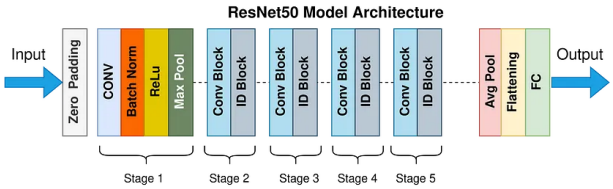


Figure 22: ResNet50 Architecture

The ResNet-50 architecture is a deep convolutional neural network that consists of 50 layers. The architecture is designed to address the problem of vanishing gradients that can occur in deep neural networks.

The ResNet-50 architecture can be divided into several blocks, each containing multiple layers. The blocks are designed to learn increasingly complex features of the input image. The architecture also includes skip connections, which allow information to bypass one or more layers and flow directly to later layers in the network. This approach helps to alleviate the problem of vanishing gradients and enables the training of deeper models.

The ResNet-50 architecture includes the following layers and blocks:

Input layer: This layer receives the input image.

Convolutional layer: This layer applies convolutional filters to the input image to extract features.

Pooling layer: This layer reduces the size of the feature maps by performing pooling operations.

Residual block: This block contains multiple convolutional layers, followed by batch normalization and ReLU activation, and a skip connection that adds the input to the output of the block. The residual block enables the network to learn residual functions, which are easier to optimize than the original mapping functions.

Fully connected layer: This layer aggregates the features learned by the previous layers and produces the final output of the network.

The ResNet-50 architecture is pre-trained on large datasets such as ImageNet, which enables it to be used as a starting point for transfer learning in various computer vision tasks. It's high accuracy and relatively low computational cost make it a popular choice for many applications.

3.9.2 Machine setup

System details are below:

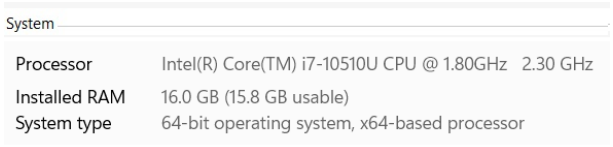


Figure 23: Machine set up

3.10 Xceptionet

3.10.1 Introduction

The Xceptionet model is a deep-learning neural network architecture for image recognition and classification. This was introduced by François Chollet in a 2016 research paper titled "Xception: Deep Learning with Depthwise Separable Convolutions". The Xceptionet model is an extension of the Inception architecture, a popular neural network model for image recognition. The Xceptionet model uses depth-separated convolutions, a type of convolutional operation that separates spatial and channel-specific filtering of convolutional layers. The Xceptionet model consists of a series of convolutional layers, followed by depth-separated convolutions and point convolutions. This architecture allows a more efficient use of computing resources while maintaining high accuracy for image classification tasks. The Xceptionet

model has achieved excellent performance in several benchmark image classification datasets, including ImageNet and COCO. It has been widely used in various applications such as self-driving cars, medical image analysis and natural language processing.

The Xceptionnet model architecture is a deep neural network architecture for image classification tasks. It is based on the original architecture but uses another type of convolutional layer called a deep separable circuit to reduce the number of parameters and improve efficiency.

Here is a high-level overview of the architecture of the Xceptionnet model:

- Input: The input to the Xceptionnet model is a fixed size color image.
- Original Convolutional Layers: The input image is passed through several convolutional layers to extract features. These convolutional layers use a small kernel size of 3x3 and step 2 to reduce the size of the feature maps.
- Depth-separated convolutions: Instead of traditional convolutional layers, the Xceptionnet model uses depth-separated convolutions, which separate spatial and channel-specific filtering of convolutional layers. This reduces the number of parameters and increases efficiency. Average Flow: The output of the deep-separated convolutional layers is passed through a series of average flow modules that contain residual connections to facilitate training.
- Output stream: The intermediate stream output is routed through an output stream that includes a global intermediate collection layer and several fully connected layers. The final result is a probability distribution between the possible classes.
- Output: The output of the Xceptionnet model is the predicted class name of the input image.

Overall, the Xceptionnet model architecture is designed to be highly efficient and accurate for image classification tasks with fewer parameters and faster computation than traditional convolutional neural network architectures.

4 Results and Models Comparison

4.1 EfficientNetV2M model

We have trained our EfficientNetV2M model on a Chest X-ray Dataset from COVID-19 Radiography Database. This database consists of

- 3616 COVID-19 positive cases
- 6012 Lung Opacity (Non-COVID lung infection)
- 10,192 Normal
- 1345 Viral Pneumonia images

Due to resource constraints and for faster training of the entire dataset our team of researchers decided to split the data set into 4 subsets and use our models to train on the subset dataset. Below are the details of the model trained on the first subset of dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ The Subset dataset consisted of

- 3706 train images belonging to 4 classes
- 794 validation images belonging to 4 classes
- 795 test images belonging to 4 classes

We have included weights of imagenet in the base model of EfficientNetV2M and excluded the top layer of the EfficientNetV2m Base model Average Pooling technique was used while building the base efficientnetv2m pretrained model The following layers were added on top of the EfficientNetv2m model

1. Dropoutlayer with 0.4 dropout rate
2. Flatten layer
3. Dense layer of 64 neurons with Relu activation function
4. Finally the output layer with 4 neurons and softmax activation function

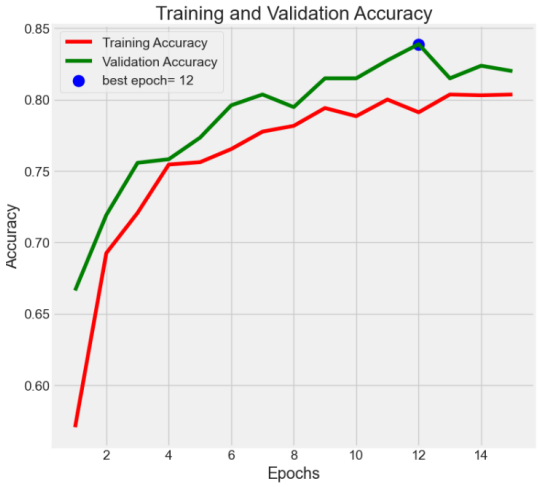
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetv2-m (Functiona  (None, 1280)             53150388
 l)

 dropout (Dropout)           (None, 1280)              0

 flatten (Flatten)           (None, 1280)              0

 dense (Dense)               (None, 64)                81984

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 53,232,632
Trainable params: 82,244
Non-trainable params: 53,150,388
```

Figure 24: EfficientNetV2m based model

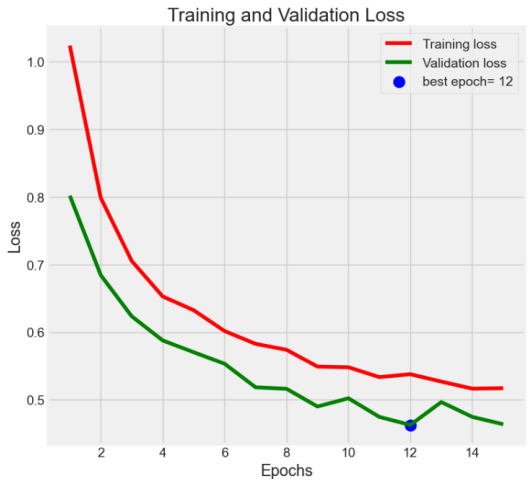
We have compiled the base model with Adam Optimizer whose learning rate as 0.0001, with categorical cross-entropy loss and metric as accuracy The model summary is shown in the figure Efficientnetv2m base model.
The total number of trainable parameters was 82,244. We used a batch size of 16 and the number of epochs we trained was 40. We have trained the model on train images and provided validation images to calculate the validation accuracy and loss. We have also provided steps per epoch which is the ratio of number of training images to the batch size.
We have also used multiple types of callbacks such as

- Early stopping callback to monitor based on minimum validation loss to stop the training in case of no further reduction in the validation loss
- Model Checkpoint to save the best model based on minimum validation loss
- ReduceLRonPlateau callback to monitor based on maximum validation accuracy

Though the number of epochs given was 40, the model finished training at the 15th epoch, as further reduction in the validation loss is not possible. Below are the plots of the Training_validation_loss and Training_validation_accuracy for reference



(a) EfficientNetV2m Train validation Accuracy



(b) EfficientNetV2m Train Validation loss

Figure 25: EfficientNetV2m based model Training and Validation plots

Below are the metrics of the efficientnetv2m based initial model on the first subset of dataset.

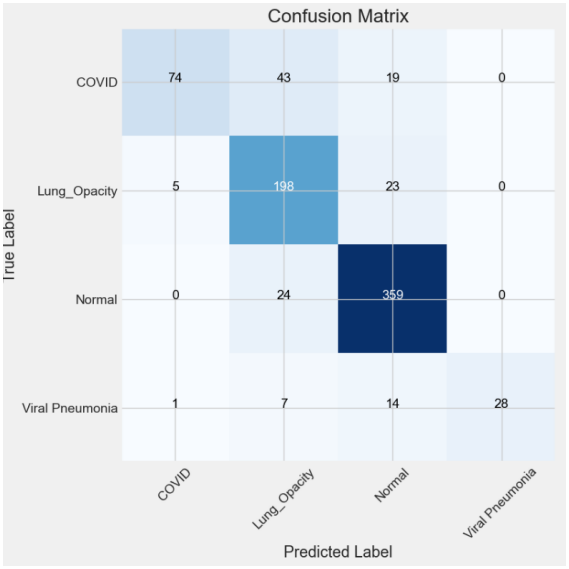
- Train Loss: 0.457, Train Accuracy: 0.832

- Validation Loss: 0.450, Validation Accuracy: 0.835
- Test Loss: 0.443, Test Accuracy: 0.827

We have calculated the confusion matrix and it is shared below Confusion matrix results show an accuracy of 83%.

	precision	recall	f1-score	support
COVID	0.93	0.54	0.69	136
Lung_Opacity	0.73	0.88	0.80	226
Normal	0.87	0.94	0.90	383
Viral Pneumonia	1.00	0.56	0.72	50
accuracy			0.83	795
macro avg	0.88	0.73	0.77	795
weighted avg	0.84	0.83	0.82	795

(a) Confusion Matrix



(b) Confustion matrix plot

Figure 26: EfficientNetv2m Confusion matrix

This model has an F1-score of 90% for Normal class where as lowest F1-score of 69% for Covid class.

4.2 Ensemble Stacking of EfficientNetv2m and ResNet50

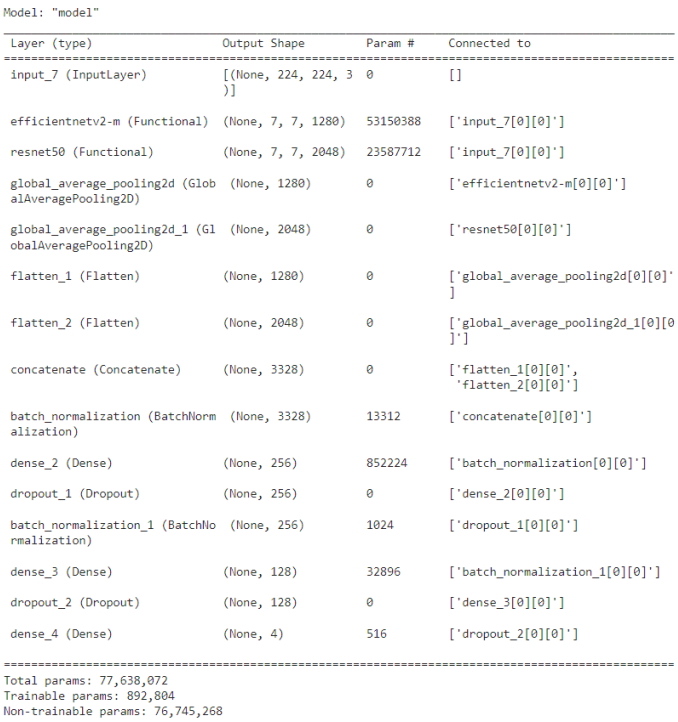
To improve the accuracy of the model further we have come up with the ensemble method of combining EfficientNetV2m based model and ResetNet50 model

Below are the details of the ensemble model trained on first subset of dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ and batch size fixed as 16. The Subset dataset consisted of

- 3706 train images belonging to 4 classes
- 794 validation images belonging to 4 classes
- 795 test images belonging to 4 classes

We have included weights of imagenet in the base models of EfficientNetV2M and Resnet50 and excluded the top layers of the EfficientNetV2m model and ResNet50 model. Global Average Pooling technique was used while building the ensemble model from efficientnetv2m and ResNet50m model Flattened layers of both these pre-trained models are merged into a single model The following layers were added on top of the merged model

- BatchNormalizationlayer
- Dense layer with 256 neurons and relu activation function
- Dropoutlayer with 50% dropout rate
- Batch normalization layer
- Dense layer with 128 neurons and relu activation function
- Droputlayer with 50% dropout rate
- Finally the output layer with 4 neurons and softmax activation function



```
Model: "model"
_____________________________________________________________________________________________________
 Layer (type)                    Output Shape         Param #     Connected to
=====================================================================================================
 input_7 (InputLayer)            [(None, 224, 224, 3  0           []
                                 )]

 efficientnetv2-m (Functional)   (None, 7, 7, 1280)   53150388    ['input_7[0][0]']

 resnet50 (Functional)           (None, 7, 7, 2048)   23587712    ['input_7[0][0]']

 global_average_pooling2d (Glob  (None, 1280)         0           ['efficientnetv2-m[0][0]']
 alAveragePooling2D)

 global_average_pooling2d_1 (Gl  (None, 2048)         0           ['resnet50[0][0]']
 obalAveragePooling2D)

 flatten_1 (Flatten)             (None, 1280)         0           ['global_average_pooling2d[0][0]'
                                                                  ]

 flatten_2 (Flatten)             (None, 2048)         0           ['global_average_pooling2d_1[0][0
                                                                  ]']

 concatenate (Concatenate)       (None, 3328)         0           ['flatten_1[0][0]',
                                                                   'flatten_2[0][0]']

 batch_normalization (BatchNorm  (None, 3328)         13312       ['concatenate[0][0]']
 alization)

 dense_2 (Dense)                 (None, 256)          852224      ['batch_normalization[0][0]']

 dropout_1 (Dropout)             (None, 256)          0           ['dense_2[0][0]']

 batch_normalization_1 (BatchNo  (None, 256)          1024        ['dropout_1[0][0]']
 rmalization)

 dense_3 (Dense)                 (None, 128)          32896       ['batch_normalization_1[0][0]']

 dropout_2 (Dropout)             (None, 128)          0           ['dense_3[0][0]']

 dense_4 (Dense)                 (None, 4)            516         ['dropout_2[0][0]']

=====================================================================================================
Total params: 77,638,072
Trainable params: 892,804
Non-trainable params: 76,745,268
```

Figure 27: EfficientNetV2m and ResNet50 Ensemble model

We have compiled the base model with Adam Optimizer whose learning rate as 0.0001, with categorical cross-entropy loss and metric as accuracy The model summary is shown in the figure EfficientNetV2m and ResNet50 Stacked model. The total number of trainable parameters was 892,804. We used a batch size of 16 and the number of epochs we trained was 20. We have trained the model on train images and provided validation images to calculate the validation accuracy and loss. We have also provided steps per epoch which is the ratio of number of training images to the batch size.
We have also used multiple types of callbacks such as

- Early stopping callback to monitor based on minimum validation loss to stop the training in case of no further reduction in the validation loss
- Model Checkpoint to save the best model based on minimum validation loss
- ReduceLRonPlateau callback to monitor based on maximum validation accuracy

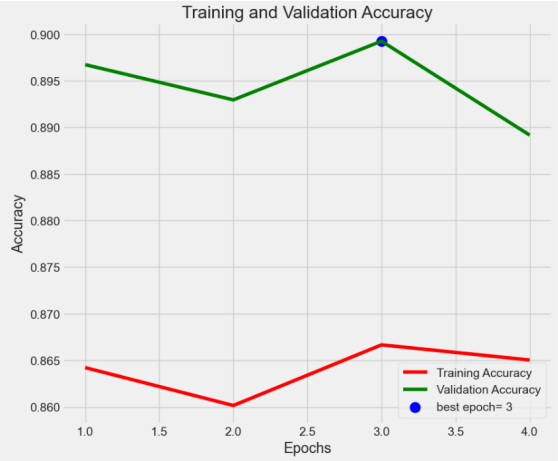
Though the number of epochs given was 20, the model finished training at 17th epoch, as further reduction in the validation loss is not possible. Below are the plots of the Training_validation_loss and Training_validation_accuracy for reference
Below are the metrics of the efficientnetv2m and resnet50 ensemble model on the first subset of dataset.

- Train Loss: 0.22, Train Accuracy: 0.91
- Validation Loss: 0.29, Validation Accuracy: 0.88
- Test Loss: 0.33, Test Accuracy: 0.88

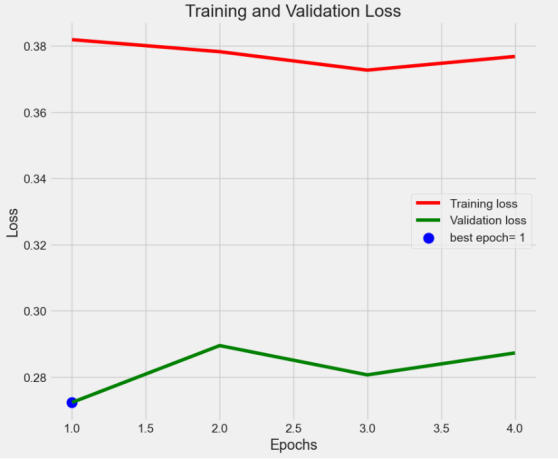
We have calculated the confusion matrix and it is shared below Confusion matrix results show an accuracy of 88%. This model has an F1-score of 93% for Normal class, 91% for Viral Pneumonia, 84% for Lung_Opacity and an F1-score of 79% for Covid class.

4.3 Stacked EfficientNetV2M and ResNet50 on Balanced dataset

Below are the details of the ensemble model trained on a subset-balanced dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ and batch size fixed as 16. The Subset dataset consisted of



(a) Ensemble model Train validation Accuracy

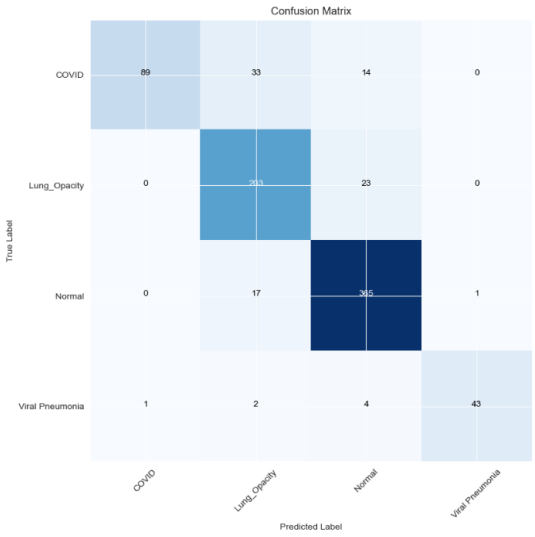


(b) Ensemble model Train validation Loss

Figure 28: Ensemble model Training and Validation plots

	precision	recall	f1-score	support
COVID	0.99	0.65	0.79	136
Lung_Opacity	0.80	0.90	0.84	226
Normal	0.90	0.95	0.93	383
Viral Pneumonia	0.98	0.86	0.91	50
accuracy			0.88	795
macro avg	0.92	0.84	0.87	795
weighted avg	0.89	0.88	0.88	795

(a) Ensemble model Confusion Matrix



(b) Ensemble model Confusion matrix plot

Figure 29: EfficientNetv2m and ResNet50 ensemble model Confusion matrix

- 4091 train images belonging to 4 classes
- 877 validation images belonging to 4 classes
- 877 test images belonging to 4 classes

We have included weights of imagenet in the base models of EfficientNetV2M and Resnet50 and excluded the top layers of the EfficientNetV2m model and ResNet50 model. Global Average Pooling technique was used while building the ensemble model from efficientnetv2m and ResNet50m model Flattened layers of both these pre-trained models are merged into a single model The following layers were added on top of the merged model

- BatchNormalizationlayer
- Dense layer with 256 neurons and relu activation function
- Dropoutlayer with 50% dropout rate
- Batch normalization layer
- Dense layer with 128 neurons and relu activation function
- Droputlayer with 50% dropout rate
- Finally the output layer with 4 neurons and softmax activation function

We have compiled the base model with Adam Optimizer whose learning rate as 0.0001, with categorical cross-entropy loss and metric as accuracy We had used a batch size of 16 and the number of epochs we trained was 18. We have trained the model on train images and provided validation images to calculate the validation accuracy and loss. We have also provided steps per epoch which is the ratio of number of training images to the batch size.
We have also used multiple types of callbacks such as

- Early stopping callback to monitor based on minimum validation loss to stop the training in case of no further reduction in the validation loss
- Model Checkpoint to save the best model based on minimum validation loss
- ReduceLRonPlateau callback to monitor based on maximum validation accuracy

Though the number of epochs given was 20, the model finished training at 18th epoch, as further reduction in the validation loss is not possible. Below are the metrics of the efficientnetv2m and resnet50 ensemble model on the subset balanced dataset.

- Train Loss: 0.17, Train Accuracy: 0.94
- Validation Loss: 0.27, Validation Accuracy: 0.9
- Test Loss: 0.32, Test Accuracy: 0.89

We have calculated the confusion matrix and it is shared below Confusion matrix results show an accuracy of 89%. This model has an F1-score of 85% for Normal class, 93% for Viral Pneumonia, 89% for Lung_Opacity and an F1-score of 90% for Covid class.

4.4 Ensemble Stacking of DenseNet169 and MobileNetV2

Below are the details of the model trained on the second subset of dataset and the results of the model. Input shape for the input layer is fixed as $224, 224, 3$ The Subset dataset consisted of

- 3702 train images belonging to 4 classes
- 796 validation images belonging to 4 classes
- 793 test images belonging to 4 classes

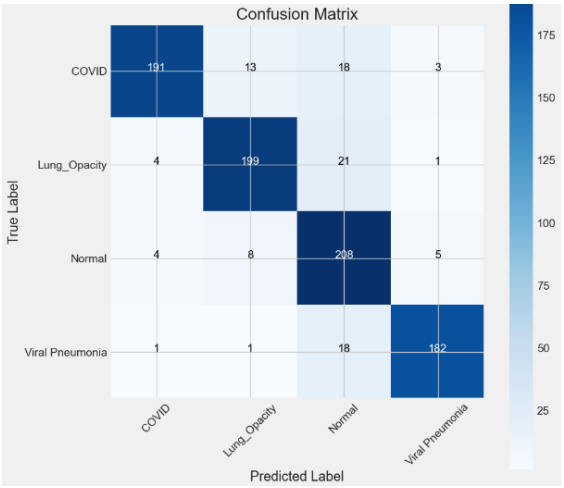
Below represents the Ensemble Stacking layers of DenseNet169 and MobileNetV2 models -

Below represents the evaluation result of the modeling of Ensemble Stacking layers of DenseNet169 and MobileNetV2 models -

The above represents the accuracy and value of the modeling of Ensemble Stacking layers of DenseNet169 and

	precision	recall	f1-score	support
COVID	0.95	0.85	0.90	225
Lung_Opacity	0.90	0.88	0.89	225
Normal	0.78	0.92	0.85	225
Viral Pneumonia	0.95	0.90	0.93	202
accuracy			0.89	877
macro avg	0.90	0.89	0.89	877
weighted avg	0.90	0.89	0.89	877

(a) Ensemble model Confusion Matrix



(b) Ensemble model Confusion matrix plot

Figure 30: EfficientNetv2m and ResNet50 ensemble model Confusion matrix

Model: "model_1"

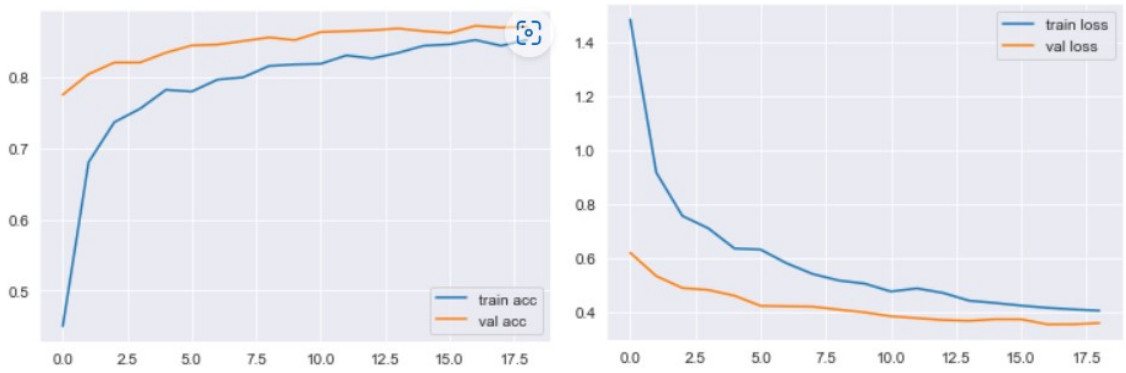
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 224, 224, 3)]	0	[]
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984	['input_1[0][0]']
densenet169 (Functional)	(None, 7, 7, 1664)	12642880	['input_1[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0	['mobilenetv2_1.00_224[0][0]']
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1664)	0	['densenet169[0][0]']
flatten (Flatten)	(None, 1280)	0	['global_average_pooling2d[0][0]']
flatten_1 (Flatten)	(None, 1664)	0	['global_average_pooling2d_1[0][0]']
concatenate (Concatenate)	(None, 2944)	0	['flatten[0][0]', 'flatten_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 2944)	11776	['concatenate[0][0]']
dense_3 (Dense)	(None, 256)	753920	['batch_normalization_2[0][0]']
dropout_2 (Dropout)	(None, 256)	0	['dense_3[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 256)	1024	['dropout_2[0][0]']
dense_4 (Dense)	(None, 128)	32896	['batch_normalization_3[0][0]']
dropout_3 (Dropout)	(None, 128)	0	['dense_4[0][0]']
dense_5 (Dense)	(None, 4)	516	['dropout_3[0][0]']

Total params: 15,700,996
Trainable params: 793,732
Non-trainable params: 14,907,264

Figure 31: Ensemble Stacking layers

Model	Class	F1-Score	Accuracy	Precision	Recall
Ensemble Stacking- Densenet169 & MobileNetV2	COVID	0.73	0.85	0.58	0.98
	Lung_Opacity	0.79		0.85	0.75
	Normal	0.89		0.92	0.86
	Viral_Pneumonia	0.99		0.98	1.00

Figure 32: result ensemble stacking



(a) Ensemble model accuracy (b) Ensemble model loss value

Figure 33: Stacking (DenseNet169,MobileNetV2) - accuracy and loss plots

MobileNetV2 models.
Below represents the confusion matrix of the modeling of Ensemble Stacking layers of DenseNet169 and MobileNetV2 models

	precision	recall	f1-score	support
0	0.58	0.98	0.73	81
1	0.85	0.75	0.79	255
2	0.92	0.86	0.89	400
3	0.98	1.00	0.99	48
micro avg	0.85	0.85	0.85	784
macro avg	0.83	0.90	0.85	784
weighted avg	0.87	0.85	0.85	784
samples avg	0.85	0.85	0.85	784

Figure 34: classification report - stacking

4.5 VGG16 - last 4 layer trainable

Below are the details of the model trained on second subset of dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ Subset dataset consisted of

- 3702 train images belonging to 4 classes
- 796 validation images belonging to 4 classes
- 793 test images belonging to 4 classes

Figure 32 below represents the VGG16 - last 4 layer trainable model

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Total params: 14,714,688
Trainable params: 7,079,424
Non-trainable params: 7,635,264

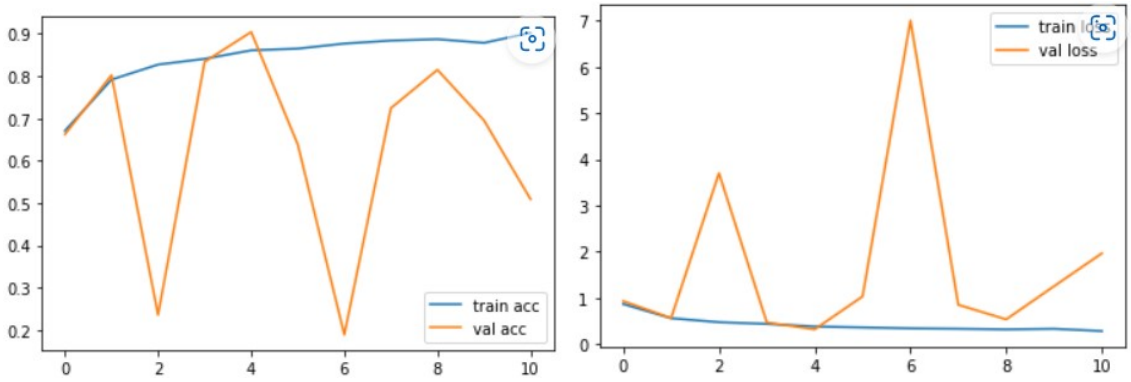
Figure 35: Vgg16 layers

Figure 33 below represents the evaluation result of the modeling of VGG16 - last 4 layer trainable model.

Model	Class	F1-Score	Accuracy	Precision	Recall
VGG 16 last 4 trainable layers	COVID	0.69	0.83	0.58	0.85
	Lung_Opacity	0.79		0.77	0.81
	Normal	0.89		0.93	0.84
	Viral_Pneumonia	0.90		1.00	0.82

Figure 36: result Vgg16

Figure 34 below represents the accuracy and value of the modeling of VGG16 - last 4 layers trainable model.
Figure 35 below represents the confusion matrix of the modeling of VGG16 - last 4-layer trainable model.



(a) Vgg16 model accuracy (b) Vgg16 model loss value

Figure 37: Vgg16 - accuracy and loss plots

	precision	recall	f1-score	support
0	0.58	0.85	0.69	93
1	0.77	0.81	0.79	212
2	0.93	0.84	0.89	417
3	1.00	0.82	0.90	62
micro avg	0.83	0.83	0.83	784
macro avg	0.82	0.83	0.82	784
weighted avg	0.85	0.83	0.84	784
samples avg	0.83	0.83	0.83	784

Figure 38: classification report - Vgg16

4.6 RestNet-50

Below represents the RestNet-50 trainable model results on the subset dataset.

	precision	recall	f1-score	support
0	0.01	0.17	0.01	6
1	0.74	0.78	0.76	385
2	0.94	0.66	0.78	529
3	0.78	0.97	0.87	40
micro avg	0.72	0.72	0.72	960
macro avg	0.62	0.64	0.60	960
weighted avg	0.85	0.72	0.77	960
samples avg	0.72	0.72	0.72	960

Figure 39: RestNet50 Classification report

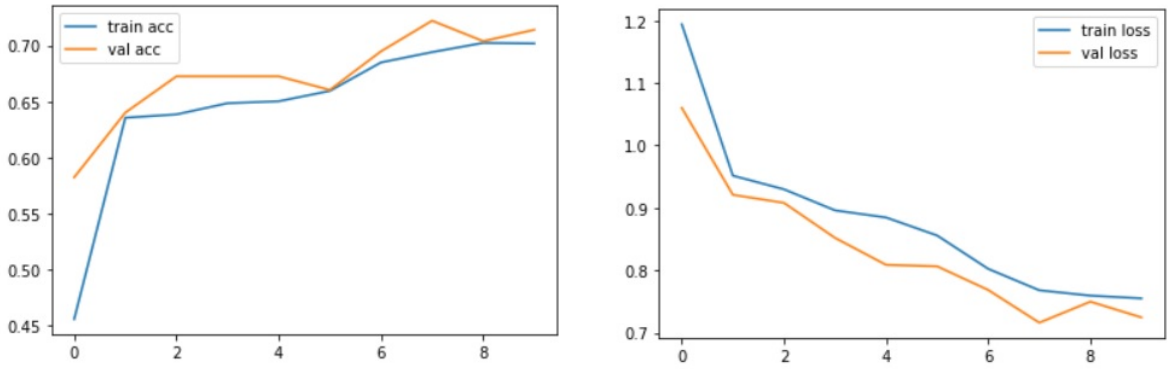


Figure 40: RestNet50 based model Training and Validation plots

4.7 AlexNet

Below are the details of the model trained on fourth subset of dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ The Subset dataset consisted of

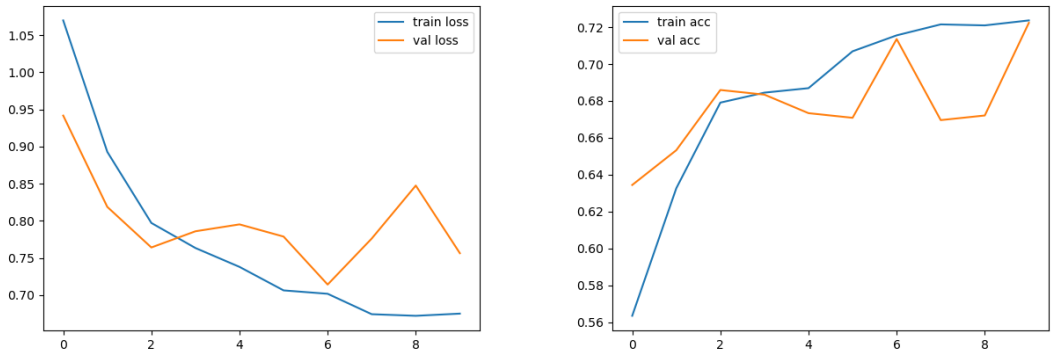
- 3706 train images belonging to 4 classes
- 794 validation images belonging to 4 classes
- 795 test images belonging to 4 classes

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 54, 54, 96)        34944

 max_pooling2d (MaxPooling2D  (None, 26, 26, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 22, 22, 256)       614656

 max_pooling2d_1 (MaxPooling  (None, 10, 10, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 384)         885120

 conv2d_3 (Conv2D)           (None, 6, 6, 384)         1327488

 conv2d_4 (Conv2D)           (None, 4, 4, 256)         884992

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 256)        0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 4096)              1052672

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 4)                 16388

=================================================================
Total params: 21,597,572
Trainable params: 21,597,572
Non-trainable params: 0
_________________________________________________________________
```

Figure 41: AlexNet- Architecture

	precision	recall	f1-score	support
0	0.43	0.59	0.50	96
1	0.58	0.78	0.67	161
2	0.90	0.71	0.79	466
3	0.86	0.98	0.92	45
micro avg	0.72	0.72	0.72	768
macro avg	0.69	0.77	0.72	768
weighted avg	0.77	0.72	0.74	768
samples avg	0.72	0.72	0.72	768

Figure 42: AlexNet- confusion matrix



(a) ALexNet model accuracy (b) AlexNet model loss value

Figure 43: AlexNet - accuracy and loss plots

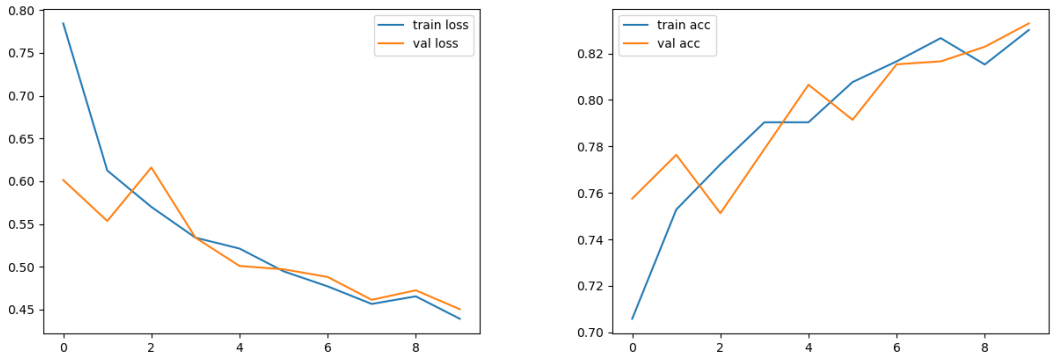
4.8 InceptionV3

Below are the details of the model trained on fourth subset of dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ Subset dataset consisted of

- 3706 train images belonging to 4 classes
- 794 validation images belonging to 4 classes
- 795 test images belonging to 4 classes

	precision	recall	f1-score	support
0	0.66	0.83	0.74	106
1	0.77	0.85	0.81	199
2	0.92	0.81	0.86	416
3	0.96	1.00	0.98	47
micro avg	0.84	0.84	0.84	768
macro avg	0.83	0.87	0.85	768
weighted avg	0.85	0.84	0.84	768
samples avg	0.84	0.84	0.84	768

Figure 44: InceptionV3- confusion matrix



(a) InceptionV3 model accuracy

(b) InceptionV3 model loss value

Figure 45: InceptionV3 - accuracy and loss plots

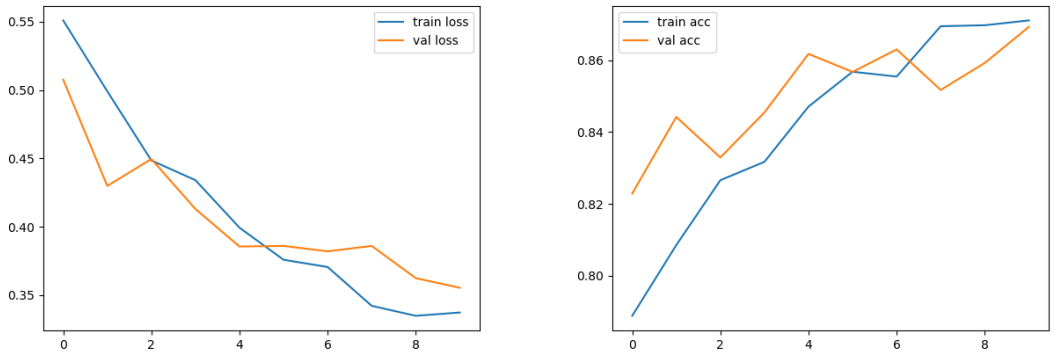
4.9 DenseNet121

Below are the details of the model trained on fourth subset of dataset and results of the model. Input shape for the input layer is fixed as $224, 224, 3$ The Subset dataset consisted of

- 3706 train images belonging to 4 classes
- 794 validation images belonging to 4 classes
- 795 test images belonging to 4 classes

	precision	recall	f1-score	support
0	0.89	0.83	0.86	142
1	0.79	0.86	0.82	202
2	0.89	0.88	0.89	375
3	0.98	1.00	0.99	49
micro avg	0.87	0.87	0.87	768
macro avg	0.89	0.89	0.89	768
weighted avg	0.87	0.87	0.87	768
samples avg	0.87	0.87	0.87	768

Figure 46: DenseNet121- confusion matrix



(a) DenseNet121 model accuracy

(b) DenseNet121 model loss value

Figure 47: DenseNet121 - accuracy and loss plots

4.10 Resnet101

The following results are obtained when trained on full dataset

reportResnet101:

```
resnet = tf.keras.applications.resnet.ResNet101(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
for layer in resnet.layers[:180]:
    layer.trainable = False
```

```
Epoch 25/30
1059/1059 [==============================] - 200s 188ms/step - loss: 0.0107 - accuracy: 0.9966 - val_loss: 0.3710 - val_accurac
y: 0.9395
Epoch 26/30
1059/1059 [==============================] - 200s 189ms/step - loss: 0.0214 - accuracy: 0.9930 - val_loss: 0.3593 - val_accurac
y: 0.9376
Epoch 27/30
1059/1059 [==============================] - 199s 188ms/step - loss: 0.0116 - accuracy: 0.9962 - val_loss: 0.3323 - val_accurac
y: 0.9381
Epoch 28/30
1059/1059 [==============================] - 200s 189ms/step - loss: 0.0166 - accuracy: 0.9942 - val_loss: 0.3505 - val_accurac
y: 0.9376
Epoch 29/30
1059/1059 [==============================] - 209s 197ms/step - loss: 0.0133 - accuracy: 0.9959 - val_loss: 0.4366 - val_accurac
y: 0.9376
Epoch 30/30
1059/1059 [==============================] - 200s 189ms/step - loss: 0.0157 - accuracy: 0.9953 - val_loss: 0.3863 - val_accurac
y: 0.9414
```


```
Train Loss:  0.11492899060249329
Train Accuracy:  0.9913793206214905
--------------------
Validation Loss:  0.34801310300827026
Validation Accuracy:  0.9482758641242981
--------------------
Test Loss:  0.37515127658843994
Test Accuracy:  0.9452054500579834
```

Figure 48: ResNet101 model results

	precision	recall	f1-score	support
COVID	0.99	0.98	0.98	362
Lung_Opacity	0.91	0.92	0.92	602
Normal	0.95	0.95	0.95	1019
Viral Pneumonia	0.98	0.98	0.98	134
accuracy			0.95	2117
macro avg	0.96	0.95	0.96	2117
weighted avg	0.95	0.95	0.95	2117

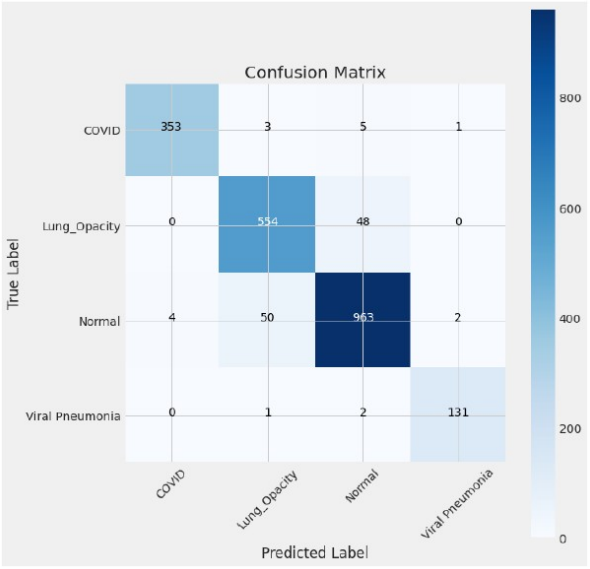


Figure 49: Resnet101 confustion matrix

4.11 Densenet201

The following results are obtained when trained on full dataset

Densenet201

```
densenet = tf.keras.applications.densenet.DenseNet201(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
for layer in densenet.layers[:180]:
    layer.trainable = False
```

```
Epoch 25/30
1059/1059 [==============================] - 157s 148ms/step - loss: 0.0601 - accuracy: 0.9797 - val_loss: 0.2183 - val_accurac
y: 0.9471
Epoch 26/30
1059/1059 [==============================] - 157s 148ms/step - loss: 0.0573 - accuracy: 0.9790 - val_loss: 0.2934 - val_accurac
y: 0.9239
Epoch 27/30
1059/1059 [==============================] - 157s 148ms/step - loss: 0.0559 - accuracy: 0.9792 - val_loss: 0.2552 - val_accurac
y: 0.9414
Epoch 28/30
1059/1059 [==============================] - 157s 148ms/step - loss: 0.0507 - accuracy: 0.9803 - val_loss: 0.3485 - val_accurac
y: 0.9263
Epoch 29/30
1059/1059 [==============================] - 158s 149ms/step - loss: 0.0505 - accuracy: 0.9820 - val_loss: 0.3100 - val_accurac
y: 0.9343
Epoch 30/30
1059/1059 [==============================] - 158s 149ms/step - loss: 0.0505 - accuracy: 0.9832 - val_loss: 0.2858 - val_accurac
y: 0.9244
```


```
Train Loss:  0.042105887085199356
Train Accuracy:  0.9913793206214905
--------------------
Validation Loss:  0.22615979611873627
Validation Accuracy:  0.9331896305084229
--------------------
Test Loss:  0.271098792552948
Test Accuracy:  0.9296174049377441
```

Figure 50: Densenet201 - result

	precision	recall	f1-score	support
COVID	0.99	0.97	0.98	362
Lung_Opacity	0.84	0.95	0.89	602
Normal	0.96	0.90	0.93	1019
Viral Pneumonia	0.98	0.96	0.97	134
accuracy			0.93	2117
macro avg	0.94	0.94	0.94	2117
weighted avg	0.93	0.93	0.93	2117

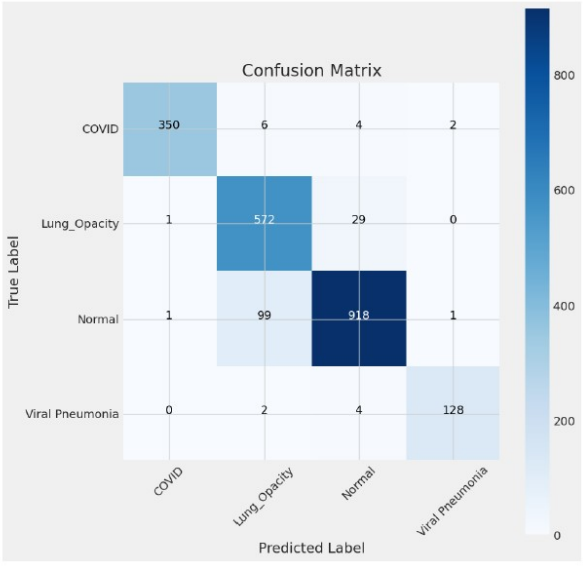


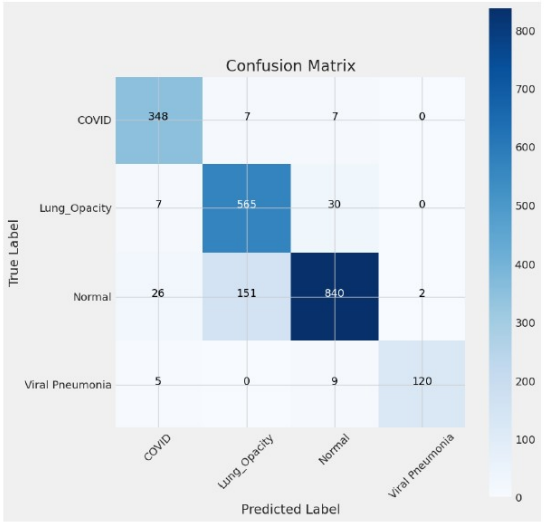
Figure 51: Densenet201 - confusion matrix

4.12 Xceptionnet

The following results are obtained when trained on full dataset

(a) Xceptionnet - result

	precision	recall	f1-score	support
COVID	0.90	0.96	0.93	362
Lung_Opacity	0.78	0.94	0.85	602
Normal	0.95	0.82	0.88	1019
Viral Pneumonia	0.98	0.90	0.94	134
accuracy			0.88	2117
macro avg	0.90	0.90	0.90	2117
weighted avg	0.89	0.88	0.89	2117



(b) Xceptionnet - confusion matrix

Figure 52: Xceptionnet

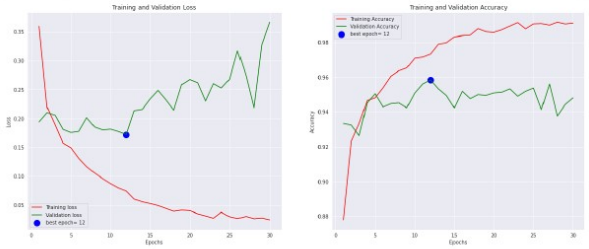
4.13 EfficientnetB5

The following results are obtained when trained on full dataset

Efficientnet

```
base_model = tf.keras.applications.EfficientNetB5(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
for layer in base_model.layers[:400]:
    layer.trainable = False
```

```
1059/1059 [==============================] - ETA: 0s - loss: 0.0308 - accuracy: 0.9898
Epoch 27: val_accuracy did not improve from 0.95841
1059/1059 [==============================] - 203s 192ms/step - loss: 0.0308 - accuracy: 0.9898 - val_loss: 0.2744 - val_accuracy: 0.9560
Epoch 28/30
1059/1059 [==============================] - ETA: 0s - loss: 0.0268 - accuracy: 0.9915
Epoch 28: val_accuracy did not improve from 0.95841
1059/1059 [==============================] - 203s 191ms/step - loss: 0.0268 - accuracy: 0.9915 - val_loss: 0.2187 - val_accuracy: 0.9376
Epoch 29/30
1059/1059 [==============================] - ETA: 0s - loss: 0.0281 - accuracy: 0.9904
Epoch 29: val_accuracy did not improve from 0.95841
1059/1059 [==============================] - 203s 191ms/step - loss: 0.0281 - accuracy: 0.9904 - val_loss: 0.3277 - val_accuracy: 0.8442
Epoch 30/30
1059/1059 [==============================] - ETA: 0s - loss: 0.0250 - accuracy: 0.9910
Epoch 30: val_accuracy did not improve from 0.95841
1059/1059 [==============================] - 203s 191ms/step - loss: 0.0250 - accuracy: 0.9910 - val_loss: 0.3663 - val_accuracy: 0.8480
```




```
Train Loss:  0.005036416929215193
Train Accuracy:  0.9978448152542114
--------------------
Validation Loss:  0.2753634750843048
Validation Accuracy:  0.9504310488700867
--------------------
Test Loss:  0.3450460433959961
Test Accuracy:  0.9461501836776733
```

Figure 53: EfficientNetB5 - result

	precision	recall	f1-score	support
COVID	1.00	0.96	0.98	362
Lung_Opacity	0.96	0.88	0.92	602
Normal	0.92	0.97	0.95	1019
Viral Pneumonia	0.94	0.99	0.97	134
accuracy			0.95	2117
macro avg	0.96	0.95	0.95	2117
weighted avg	0.95	0.95	0.95	2117

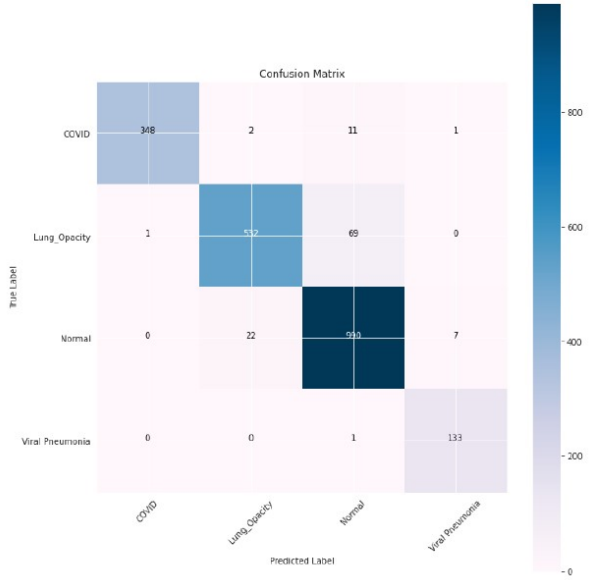


Figure 54: EfficientNetB5 - confusion matrix

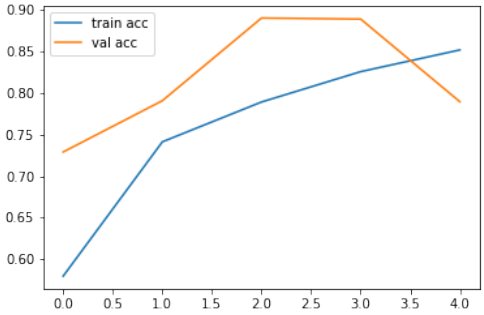
5 Model Optimization

5.1 Model Optimization - VGG16

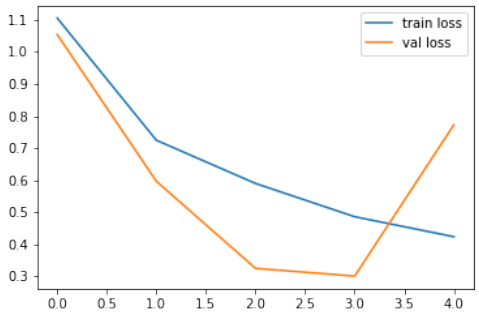
There are various techniques used to optimize the VGG16 base model performance further as below:

- Data augmentation: it is a technique used in deep learning to artificially expand the size of a training dataset by creating new, modified versions of the original data. This is typically done by applying various transformations to the images, such as rotation, scaling, cropping, flipping, and color jittering. One common way to apply data augmentation in deep learning is through an image generator. An image generator is a tool that generates new images by applying random transformations to the original images in real time during training. This can help improve the generalization performance of the model, reduce overfitting, and improve the accuracy of the predictions. In Keras, the ImageDataGenerator class provides a convenient way to apply data augmentation using an image generator. values used are rescale=1./255, rotation range=10, width shift range=0.1, height shift range=0.1, shear range=0.1, zoom range=0.1, horizontal flip=True, fill mode = 'nearest'
- Learning Rate Schedule: This technique involves gradually decreasing the learning rate as training progresses. This helps the optimization algorithm to converge faster and reduce oscillations around the minimum. A learning rate of 1e-6 has been used.
- Momentum: This technique involves adding a momentum term to the SGD update rule. The momentum term helps the optimization algorithm to move faster in the direction of the minimum by accumulating gradients over time. A momentum of 0.9 has been used.
- Batch Normalization: This technique involves normalizing the inputs to each layer to have zero mean and unit variance. This helps to stabilize the training process and improve the performance of the model. This technique has been used after freezing the top 4 layers of VGG16.
- Dropout: This technique involves randomly dropping out some neurons during training to prevent overfitting. This helps the model to learn more robust features that generalize better to new data. This technique has been used after freezing the top 4 layers of VGG16.
- Early Stopping: This technique involves monitoring the validation loss during training and stopping the training process when the validation loss stops improving. This helps to prevent overfitting and improves the generalization performance of the model. This technique has been used with monitor:'val accuracy', min delta:0.01, patience:6, verbose:1, mode:'auto', baseline: none, restore best weights: True These are just a few of the many optimization techniques that can be used with VGG16 to improve its performance on image classification tasks. The choice of technique depends on the specific problem and the data involved.
- Reduce LR on the plateau: Reduce LR on Plateau is a technique used in training deep learning models to adjust the learning rate (LR) of the optimizer when the validation loss stops improving. This technique is used to speed up the training process and to avoid the optimizer getting stuck in a local minimum or plateau. The basic idea behind the Reduce LR on Plateau technique is to monitor the validation loss at the end of each epoch or after a certain number of iterations, and if the validation loss stops improving, to reduce the learning rate by a factor. The reduction factor is typically set to a value between 0.1 and 0.5, depending on the problem and the model. Here are the steps involved in implementing the Reduce LR on Plateau technique in a deep learning model training: Define the optimizer and learning rate scheduler, such as the SGD optimizer and StepLR scheduler, in PyTorch or Keras. Set a monitor for the validation loss to check if it has stopped improving. If the validation loss does not improve for a certain number of epochs or iterations, reduce the learning rate by a factor using the learning rate scheduler.Continue training the model with the reduced learning rate. The Reduce LR on Plateau technique is useful in deep learning models, including VGG16, because it allows the optimizer to adjust the learning rate to escape from a local minimum or plateau and to converge faster to the global minimum. It is a widely used technique in many state-of-the-art deep learning models and is considered an essential component of effective model training.monitor ='val accuracy',factor=0.01, patience=6, verbose=0,mode='max',min delta=0.01
- class weights: In VGG16, class weights optimization can be implemented to address class imbalance during training for image classification tasks. Determine the frequency of each class in the training data. Compute the class weights by taking the inverse of the frequency of each class or by using a different formula based on the problem. Define the loss function for the VGG16 model. For multi-class classification, the most commonly used loss function is categorical cross-entropy. Apply the class weights to the loss function. In Keras, the class weights can be passed as a dictionary to the model.fit() function.
- Stochastic Gradient Descent (SGD): it is a popular optimization algorithm used in deep learning to minimize the loss function during training. It is a variant of the gradient descent algorithm that updates the model

parameters in small batches rather than using the entire dataset. This allows SGD to converge faster and is especially useful when dealing with large datasets. In SGD, the weights of the model are updated using the negative gradient of the loss function with respect to the weights. The update rule for SGD is as follows: w = w - lr * dw, where w is the weight matrix, lr is the learning rate, and dw is the gradient of the loss function with respect to w computed on a mini-batch of training examples. The learning rate determines the size of the step taken during each update and is a hyperparameter that needs to be tuned carefully for optimal performance. This is how it is used : opt = SGD(lr=1e-6, momentum=0.9) & model.compile(optimizer=sgd, loss='categorical crossentropy', metrics=['accuracy'])



(a) VGG16 optimization accuracy-class weights

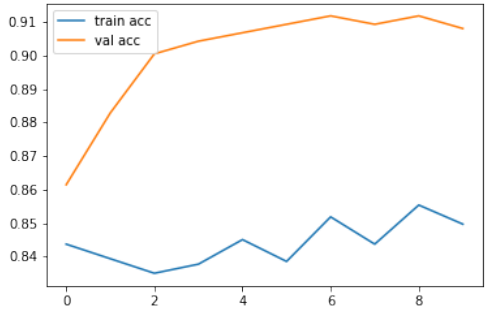


(b) VGG16 optimization loss-class weights

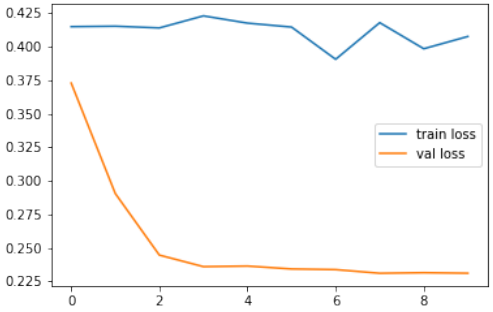
Figure 55: VGG16 accuracy and loss with class weights

	precision	recall	f1-score	support
0	0.91	0.72	0.81	169
1	0.69	0.92	0.79	166
2	0.88	0.83	0.86	402
3	0.96	0.98	0.97	47
micro avg	0.84	0.84	0.84	784
macro avg	0.86	0.86	0.85	784
weighted avg	0.85	0.84	0.84	784
samples avg	0.84	0.84	0.84	784

Figure 56: VGG16 classification report after optimization - class weights



(a) VGG16 optimization accuracy-SGD



(b) VGG16 optimization loss-SGD

Figure 57: VGG16 accuracy and loss with SGD

	precision	recall	f1-score	support
0	0.90	0.95	0.92	128
1	0.86	0.82	0.84	235
2	0.89	0.91	0.90	372
3	1.00	0.98	0.99	49
micro avg	0.89	0.89	0.89	784
macro avg	0.91	0.91	0.91	784
weighted avg	0.89	0.89	0.89	784
samples avg	0.89	0.89	0.89	784

Figure 58: VGG16 classification report after optimization - SGD

5.2 Model Optimization - EfficientNetB5

To improve the accuracy of the EfficientNetB5 model further we decided to apply the RAdam optimizer. RAdam (Rectified Adam) optimizer is a variant of the Adam optimizer that was proposed in 2019 by Liu et al. in their paper "On the Variance of the Adaptive Learning Rate and Beyond". The main motivation behind the development of RAdam was to address the issue of slow convergence that can occur with the Adam optimizer, particularly in the early stages of training.

The key feature of RAdam is that it adapts the learning rate of each weight parameter based on the gradient variance and historical gradient information. Like Adam, RAdam uses moving averages of the gradient and the squared gradient to estimate the first and second moments of the gradient. However, in RAdam, the moving averages are corrected to account for the variance of the adaptive learning rate, which helps to stabilize the training process and improve the convergence speed.

RAdam has been shown to outperform Adam and other popular optimization algorithms on a wide range of deep learning tasks, including image classification, object detection, and natural language processing. In addition, RAdam does not require manual tuning of hyperparameters, making it easier to use in practice.

Below are the details of the optimized model trained on full dataset. The input shape for the input layer is fixed as $224, 224, 3$ and batch size is fixed as 16. Full dataset consisted of

- 14815 train images belonging to 4 classes

- 3175 validation images belonging to 4 classes
- 3175 test images belonging to 4 classes

We have compiled the base model with RAdam Optimizer with categorical cross entropy loss and metric as accuracy. Below are the hyperparameters of the optimized model

- learning rate: 0.0001
- beta_1:0.9
- beta_2:0.999
- epsilon=1e-7
- amsgrad=False

We have frozen the base model layers. So the total number of trainable parameters was the same as base model 23826044. We used a batch size of 16 and the number of epochs we trained was 10. We have trained the model on train images and provided validation images to calculate the validation accuracy and loss. We have also provided steps per epoch which is the ratio of number of training images to the batch size.
We have also used multiple types of callbacks such as

- Early stopping callback to monitor based on minimum validation loss to stop the training in case of no further reduction in the validation loss
- Model Checkpoint to save the best model based on minimum validation loss
- ReduceLRonPlateau callback to monitor based on maximum validation accuracy

Though the number of epochs given was 10, the model finished training at 6th epoch, as further reduction in the validation loss is not possible.

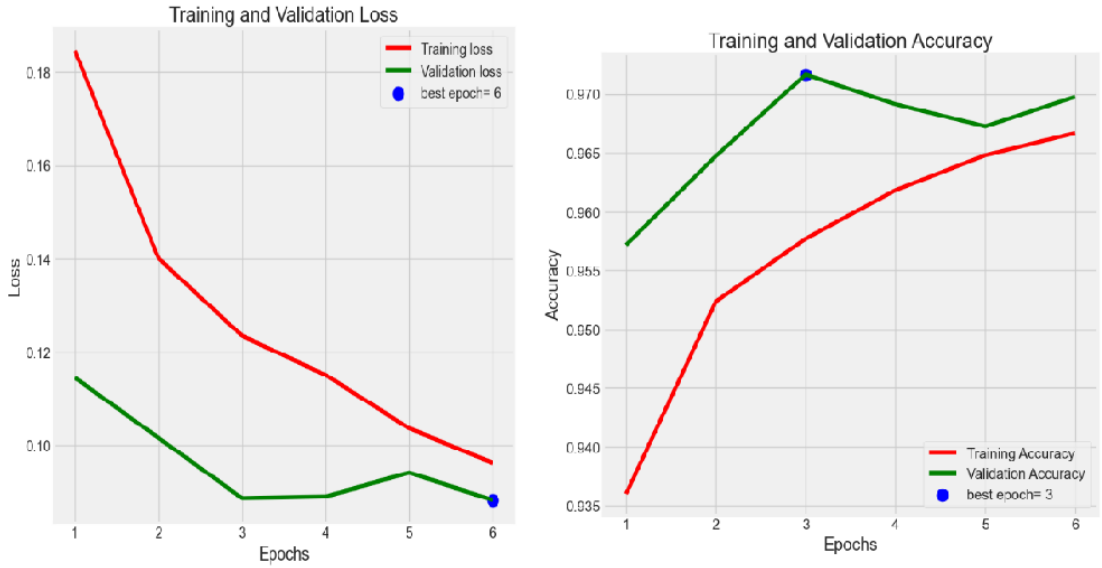


Figure 59: EfficientNetB5 Optimized model Train & validation loss and accuracy

Below are the metrics of the EfficientNetB5 based optimized model on the full of dataset.

- Train Loss: 0.06, Train Accuracy: 0.98
- Validation Loss: 0.09, Validation Accuracy: 0.97

- Test Loss: 0.04, Test Accuracy: 0.98

We have calculated the confusion matrix and it is shared below

	precision	recall	f1-score	support
COVID	1.00	0.99	0.99	542
Lung_Opacity	0.98	0.97	0.98	902
Normal	0.98	0.99	0.99	1529
Viral Pneumonia	1.00	0.97	0.98	202
accuracy			0.99	3175
macro avg	0.99	0.98	0.99	3175
weighted avg	0.99	0.99	0.99	3175

Figure 60: EfficientNetB5 Optimized model Confusion matrix

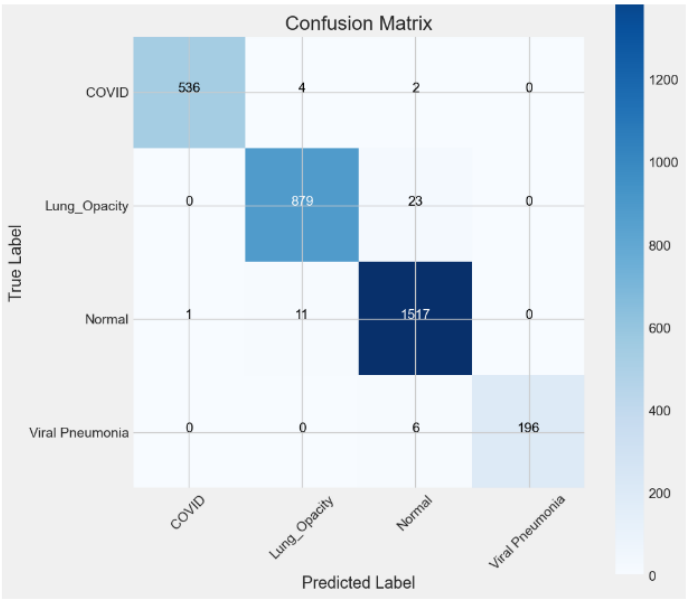


Figure 61: EfficientNetB5 Optimized model Confusion matrix plot

Confusion matrix results show test accuracy of 99%.
EfficientNetB5 Optimized model has F1-score of 99% for Covid Class
EfficientNetB5 Optimized model has F1-score of 98% for Viral Pneumonia,
EfficientNetB5 Optimized model has F1-score of 98% for Lung Opacity
EfficientNetB5 Optimized model has F1-score of 99% for Normal Class

6 Consolidated Results

The below table shows the consolidated performance metrics of all the models that we tried to train on our dataset.

Model performance results using subset dataset

Model, dataset	Class	Epochs	F1-Score	Accuracy	Precision	Recall
EfficientNetV2M Base model, subset1	COVID	15	0.69	0.83	0.93	0.54
	Lung_Opacity		0.80		0.73	0.88
	Normal		0.92		0.87	0.94
	Pneumonia		0.72		1	0.56
Stacked Model : EfficientNetV2M + ResNet50, subset1	COVID	17	0.79	0.88	0.99	0.65
	Lung_Opacity		0.84		0.80	0.90
	Normal		0.93		0.90	0.95
	Pneumonia		0.91		0.98	0.86
Ensemble Stacking- Densenet169 and Mobilenet V2 , subset2	COVID	13	0.73	0.85	0.58	0.98
	Lung_Opacity		0.79		0.85	0.75
	Normal		0.89		0.92	0.86
	Pneumonia		0.99		0.98	1.00
VGG16 (4 layers trainable), subset4	COVID	5	0.69	0.83	0.58	0.85
	Lung_Opacity		0.79		0.77	0.81
	Normal		0.89		0.93	0.84
	Pneumonia		0.99		1.00	0.82
AlexNet, subset3	COVID	10	0.50	0.72	0.43	0.59
	Lung_Opacity		0.67		0.58	0.78
	Normal		0.79		0.90	0.71
	Pneumonia		0.92		0.86	0.98
InceptionV3, subset3	COVID	10	0.74	0.83	0.66	0.83
	Lung_Opacity		0.81		0.77	0.85
	Normal		0.86		0.92	0.81
	Pneumonia		0.98		0.96	1.00
DenseNet121,subset3	COVID	10	0.86	0.87	0.89	0.83
	Lung_Opacity		0.82		0.79	0.86

	Normal		0.89		0.89	0.88
	Pneumonia		0.99		0.98	1.00
Stacked model of EfficientV2m & ResNet50, with balanced dataset	COVID	18	0.9	0.89	0.95	0.85
	Lung_Opacity		0.89		0.9	0.88
	Normal		0.85		0.78	0.92
	Pneumonia		0.93		0.95	0.9

Model performance results using full dataset

Model, dataset	**Class**	**Epochs**	**F1-Score**	**Accuracy**	**Precision**	**Recall**
Resnet101 (full dataset)	COVID	30	0.98	0.95	0.99	0.98
	Lung_Opacity		0.92		0.91	0.92
	Normal		0.95		0.95	0.95
	Pneumonia		0.98		0.98	0.98
Densenet201 (full dataset)	COVID	30	0.98	0.93	0.99	0.97
	Lung_Opacity		0.89		0.84	0.95
	Normal		0.93		0.96	0.9
	Pneumonia		0.97		0.98	0.96
Xceptionnet (full dataset)	COVID	30	0.93	0.88	0.9	0.96
	Lung_Opacity		0.85		0.78	0.94
	Normal		0.88		0.95	0.82
	Pneumonia		0.94		0.98	0.9
EfficientNetB5 (full dataset)	COVID	30	0.98	0.95	1	0.96
	Lung_Opacity		0.92		0.96	0.88
	Normal		0.95		0.92	0.97
	Pneumonia		0.97		0.95	0.99

Class-wise Best models results using subset dataset

Class	Model	F1-score	Accuracy
COVID	Stacked model of EfficientV2m & ResNet50	**0.9**	**0.89**
Lung_Opacity	Stacked model of EfficientV2m & ResNet50	**0.89**	**0.89**
Normal	DenseNet121	**0.89**	**0.87**
Pneumonia	DenseNet121	**0.99**	**0.87**

Class-wise VGG16 model Optimization results - subset-4

Class	Model	F1-score	Accuracy	Precision	Recall
COVID	VGG16 with balanced class weight, SGD (learning rate=1e-6,momentum=0.9)	**0.92**	**0.89**	**0.90**	**0.95**
LungOpacity		**0.84**		**0.86**	**0.82**
Normal		**0.90**		**0.89**	**0.91**
Pneumonia		**0.99**		**1.0**	**0.98**

Class-wise VGG16 model Optimization results - subset-4

Class	Model	F1-score	Accuracy	Precision	Recall
COVID	VGG16 with balanced class weight, ADAM (learning rate=0.0001)	**0.81**	**0.83**	**0.91**	**0.72**
LungOpacity		**0.79**		**0.69**	**0.92**
Normal		**0.86**		**0.88**	**0.83**
Pneumonia		**0.97**		**0.96**	**0.98**

Class-wise Best models results on Full dataset

Class	Model	F1-score	Accuracy
COVID	EfficientNetB5	**0.98**	**0.95**
Lung_Opacity		**0.92**	
Normal		**0.95**	
Pneumonia		**0.97**	

Class-wise EfficientNetB5 model Optimization results - Full dataset

Class	Model	F1-score	Accuracy	Precision	Recall
COVID	EfficientNetB5 with R-ADAM optimizer learning_rate=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-7, amsgrad=False epochs = 6	**0.99**	**0.99**	**1.0**	**0.99**
LungOpacity		**0.98**		**0.98**	**0.97**
Normal		**0.99**		**0.98**	**0.99**
Pneumonia		**0.98**		**1.0**	**0.97**

7 Gradient Class Activation Heatmap

7.1 GRAD CAM for EfficientNetB5 model

Gradient class activation maps (GradCAM) is a technique used to visualize the regions of an image that are important for predicting a particular class by a convolutional neural network (CNN). GradCAM is an extension of the class activation map (CAM) technique that allows for better localization of important regions by using the gradient information of the predicted class with respect to the feature maps. In CAM, the last convolutional layer feature maps are weighted by the corresponding importance weight to generate a class activation map. However, CAM does not take into account the gradient information of the predicted class with respect to the feature maps. GradCAM addresses this limitation by computing the gradients of the predicted class with respect to the feature maps and weighting the feature maps by the gradient information to generate a more accurate class activation map.

Using EfficientNetB5 Model output, we have generated gradient class activation maps for all the 3 disease classes COVID, Pneumonia and Lung Opacity.

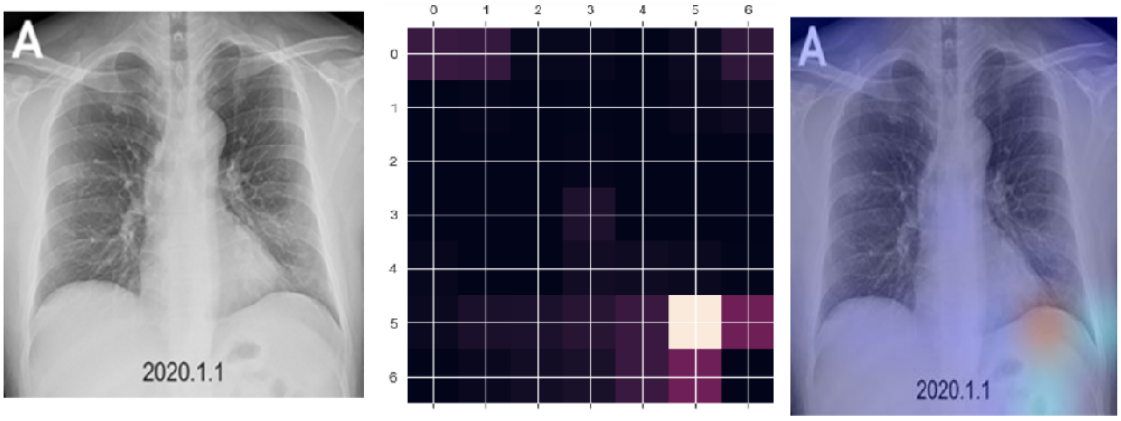


Figure 62: COVID Gradient CAM

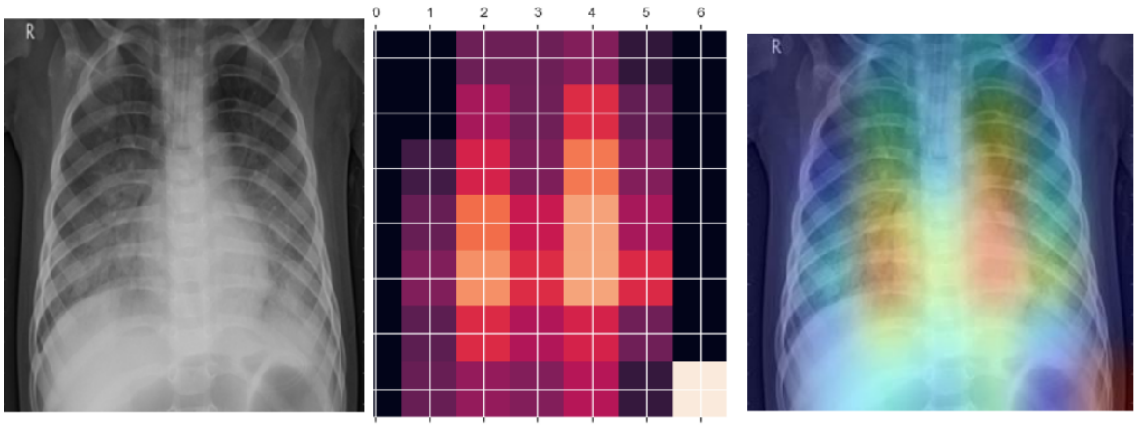


Figure 63: Pneumonia Gradient CAM

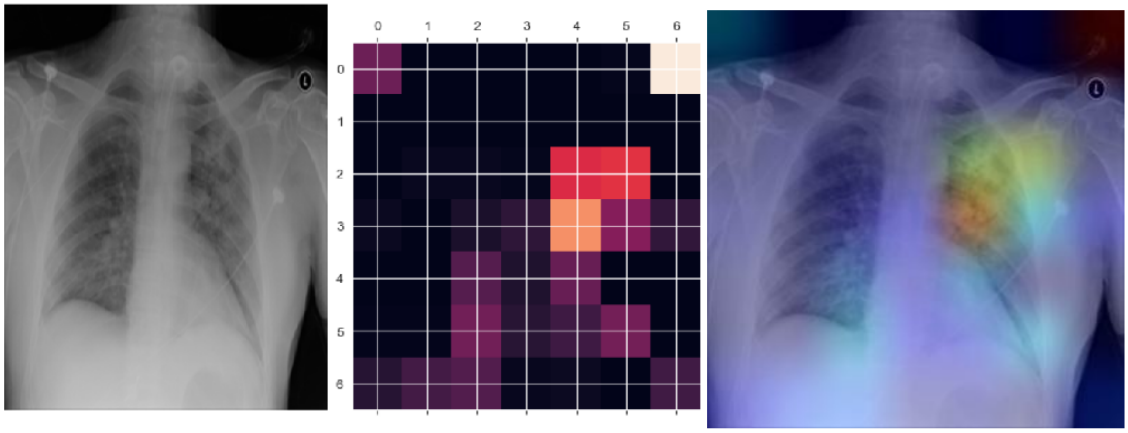


Figure 64: Lung Opacity Gradient CAM

7.2 Gradient Class Activation Map results for VGG16 model

Gradient Class Activation was applied on VGG16 pre-trained model as well to see whether it is able to localize the disease in 4 categories or not. This technique worked perfectly.

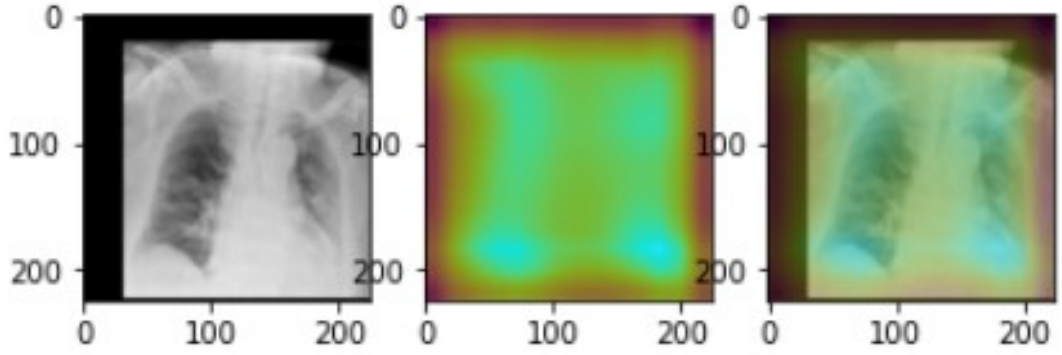


Figure 65: COVID Gradient CAM

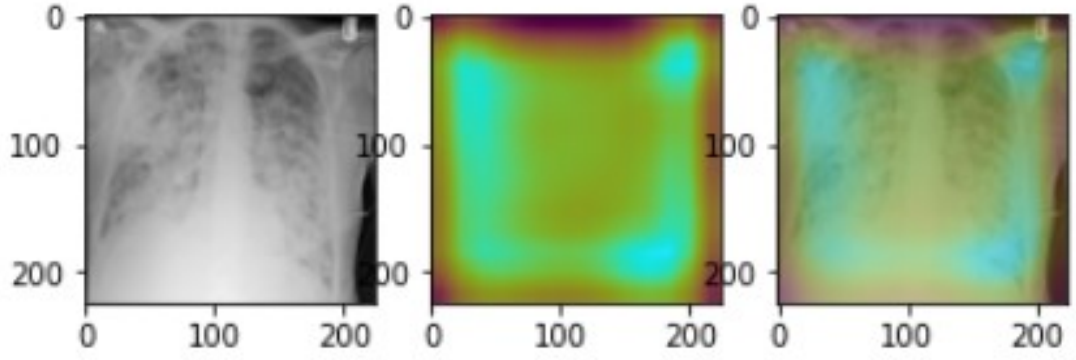


Figure 66: Lung Opacity Gradient CAM

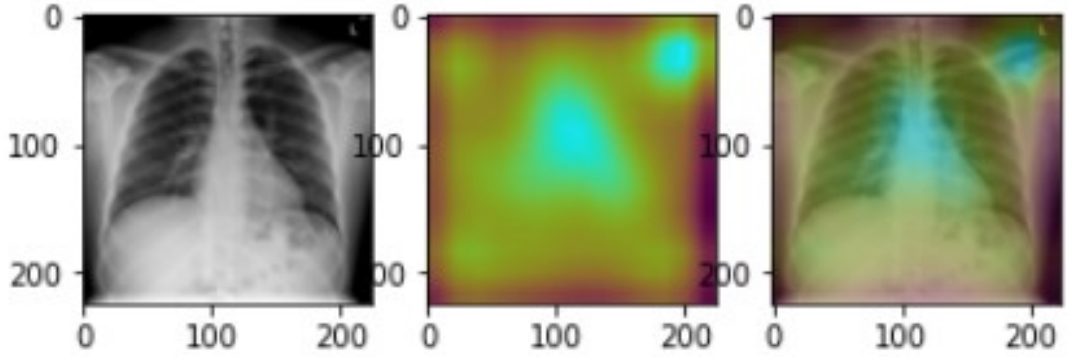


Figure 67: Normal Gradient CAM

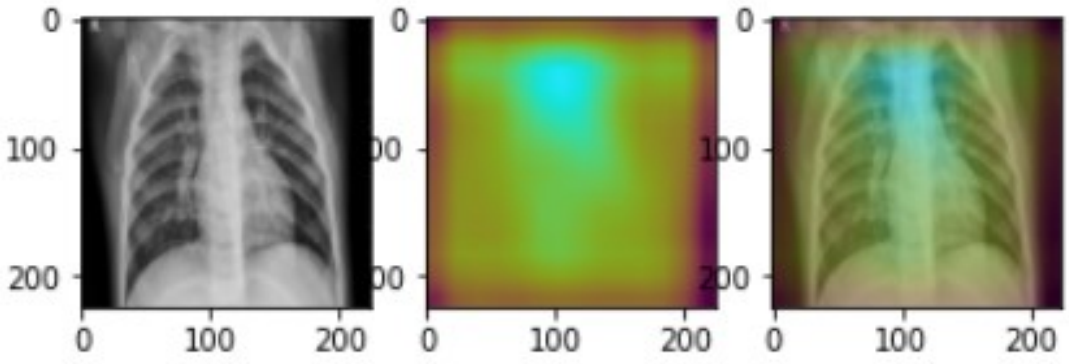


Figure 68: Viral Pneumonia Gradient CAM

8 Conclusion and future work

The objective of this research was to develop a deep convolutional neural network (CNN) model that could accurately distinguish between COVID-19 and non-COVID-19 diseases, such as viral pneumonia, using chest X-ray images. To achieve this, we implemented various state-of-art Computer Vision architectures, pre-trained models and ensemble techniques. Out of the whole lot, EfficientNetB5 with RAdam optimization demonstrated outstanding results with a remarkable overall accuracy of 99% and F1-scores of 99% for the COVID, 98% for the lung opacity, 99% for normal, and 98% for viral pneumonia. Grad CAM technique was implemented to visualize the infected area of the lung. In the future, we will continue the research to classify and localize more such lung diseases. Below are some problem areas that we plan to workon.

Problem 1: Influence of Data Generation, Attention and Ensemble Boosting in the performance of CNN architecture.
Problem 2: Pneumothorax, Fibrosis and Infiltration (Lung diseases) detection in CXR images using deep learning methods.
Problem 3: Cardiomegaly detection and segmentation using CXR images through deep learning methods
Problem 4: Lung disease detection and localization using CT scan images through deep learning methods

Acknowledgement 8.1 *The first author would like to thank Great Learning for the opportunity. The second author would like to thank PES University for the computing facility.*

References

[1] Kaylie Zhu Brandon Yang Hershel Mehta Tony Duan Daisy Ding Aarti Bagul Curtis Langlotz Katie Shpanskaya Matthew P. Lungren Andrew Y. Ng Pranav Rajpurkar, Jeremy Irvin. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv e-prints*, pages arXiv–1711, 2017.

[2] Seongjun Choi Ahyoung Lee Sedong Min Min Hong Sungyeup Kim, Beanbonyka Rim. Deep learning in multi-class lung diseases' classification on chest x-ray images. *mdpi journals*, pages mdpi–2075, 2022.

[3] Hannes Nickisch Michael Grass Tobias Knopp Axel saalbach Ivo M, Baltruschat. Comparison of deep learning approaches for multi-label chest x-ray classification. *Scientific Reports*, pages 1–10, 2019.

[4] Qiang Ni a Richard Jiang a Haris Pervaiz a Nada M Elshennawy b Goram Mufarah M, Alshmrani a. A deep learning architecture for multi-class lung diseases classification using chest x-ray (cxr) images. *Alexandria Engineering Journal*, pages 1–13, 2022.

[5] Plácido Francisco Lizancos Vidal Milena Cruz Laura Abelairas López Eva Castro Lopez Jorge Novo Marcos Ortega Joaquim De Moura, Lucía Ramos García. Deep convolutional approaches for the analysis of covid-19 using chest x-ray images from portable devices. *ieee*, pages ieee–9239272, 2020.

[6] Jun Shen Chengdi Wang Zhihuan Li Linsen Ye Xingwang Wu Ting Chen Kai Wang Xuan Zhang Zhongguo Zhou Jian Yang Ye Sang Ruiyun Deng Wenhua Liang Tao Yu Ming Gao Jin Wang Zehong Yang Huimin Cai Guangming Lu Lingyan Zhang Lei Yang Wenqin Xu Winston Wang Andrea Olvera Ian Ziyar Charlotte Zhang Oulan Li Weihua Liao Jun Liu Wen Chen Wei Chen Jichan Shi Lianghong Zheng Longjiang Zhang Zhihan Yan Xiaoguang Zou Guiping Lin Guiqun Cao Laurance L. Lau Long Mo Yong Liang Michael Roberts Evis Sala Carola-Bibiane

Schönlieb Manson Fok Johnson Yiu-Nam Lau Tao Xu Jianxing He Kang Zhang Weimin Li Guangyu Wang, Xiaohong Liu and Tianxin Lin. A deep-learning pipeline for the diagnosis and discrimination of viral, non-viral and covid-19 pneumonia from chest x-ray images. *nature biomedical engineering*, pages 1–16, 2023.

[7] Dongxiao Zhu Xin Li, Chengyin Li. Covid-mobilexpert: On-device covid-19 patient triage and follow-up using chest x-rays. *arXiv e-prints*, pages arXiv–2004, 2020.

[8] Shahadate Rezvy Tahmina Zebin. Covid-19 detection and disease progression visualization: Deep learning on chest x-rays for classification and coarse localization. *Applied Intelligence*, 2020.

[9] MOHAMED BEN AHMED IMANE ALLAOUZI. A novel approach for multi-label chest x-ray classification of common thorax diseases. *ieee*, pages 1–10, 2019.

[10] Marwa A.Shouman Mohamed Esmail Karar, Ezz El-Din Hemdan. Cascaded deep learning classifiers for computer-aided diagnosis of covid-19 and pneumonia diseases in x-ray scans. *Complex Intelligent Systems*, pages 1–13, 2020.

[11] Yerzhan Tashkenbayev Dinara Tokseit Nurbek Saparkhojayev, Lazzat Zholayeva. Abnormality detection in chest x-ray images using uncertainty prediction algorithms. *ieee*, pages ieee–9663852, 2021.

[12] Chien-Ming Li Ying-Che Kuo Neng-Sheng Pai Chia-Hung Lin Jian-Xing Wu, Pi-Yun Chen. Multilayer fractional-order machine vision classifier for rapid typical lung diseases screening on digital chest x-ray images. *ieee*, pages ieee–9108227, 2020.

[13] Peishu Wu a Kathy Clawson Han Li a, Nianyin Zeng a. Cov-net: A computer-aided diagnosis method for recognizing covid-19 from chest x-ray images via machine vision. *sciencedirect*, pages sciencedirect–S095741742201243X, 2022.

[14] Hrishikesh Deshpande Axel Saalbach Evan Schwab, André Gooßen. Localization of critical findings in chest x-ray without local annotations using multi-instance learning. *ieee*, pages ieee–9098551, 2020.

[15] R.K. Pateriya 1 Rajeev Kumar Gupta 2-Ashutosh Sharma Om Uparkar 1, Jyoti Bharti 1. Vision transformer outperforms deep convolutional neural network-based model in classifying x-ray images. *sciencedirect*, pages sciencedirect–S1877050923002090, 2023.

[16] Tripti Mahara Mayank Bali. Comparison of affine and dcgan-based data augmentation techniques for chest x-ray classification. *sciencedirect*, pages sciencedirect–S1877050923000108, 2023.

[17] Marcus Stoffel Rutwik Gulakala, Bernd Markert. Rapid diagnosis of covid-19 infections by a progressively growing gan and cnn optimisation. *sciencedirect*, pages sciencedirect–S0169260722006435, 2023.

[18] Sina Mojtahedi Tohid Yousefi Rezaii Ali Farzamnia Saeed Meshgini-Ismail Saad Sobhan Sheykhivand, Zohreh Mousavi. Developing an efficient deep neural network for automatic detection of covid-19 using chest x-ray images. *sciencedirect*, pages sciencedirect–S1110016821000144, 2023.

[19] Zong Woo Geem Gi-Tae Han Ram Sarkar Rohit Kundu, Ritacheta Das. Pneumonia detection in chest x-ray images using an ensemble of deep learning models. *journals.plos*, pages journals.plos–journal.pone.0256630, 2021.

[20] Devvi Sarwinda Naufal Hilmizen, Alhadi Bustamam. The multimodal deep learning for diagnosing covid-19 pneumonia from chest ct-scan and x-ray images. *ieee*, pages ieee–9315478, 2021.

[21] Krishna Prakasha Pranshu Bahadur-Ankit Choraria Sushobhitha M Sowjanya J Srikanth Prabhu-Krishnaraj Chadaga Wattana Viriyasitavat Vasundhara Acharya, Gaurav Dhiman and Sandeep Kautish. Ai-assisted tuberculosis detection and classification from chest x-rays using a deep learning normalization-free network model. *hindawi*, pages hindawi–2022/2399428, Oct2022.

[22] Erdaw Tachbele Yabsera Erdaw. Machine learning model applied on chest x-ray images enables automatic detection of covid-19 cases with high accuracy. *ncbi*, pages ncbi–PMC8409602, Aug2021.

[23] Aravind Sasidharan Pillai. Multi-label chest x-ray classification via deep learning. *Journal of Intelligent Learning Systems and Applications*, pages Journal of Intelligent Learning Systems and Applications, 2022, 14, 43–56, 2022.

[24] MUHAMMAD ABDUL KADIR KHANDAKER REJAUL ISLAM KHANDAKAR F. ISLAM RASHID MAZHAR TAHIR HAMID5 MOHAMMAD TARIQUL ISLAM SAAD KASHEM ZAID BIN MAHBUB MOHAMED ARSELENE AYARI MUHAMMAD E. H. CHOWDHURY TAWSIFUR RAHMAN, AMITH KHANDAKAR. Reliable tuberculosis detection using chest x-ray with deep learning, segmentation and visualization. *IEEE*, page Digital Object Identifier 10.1109/ACCESS.2020.3031384, 2020.

[25] Dina M. Ibrahim Nada M. Elshennawy. Deep-pneumonia framework using deep learning models based on chest x-ray images. *mdpi*, pages https://www.mdpi.com/2075–4418/10/9/649, 2020.

[26] Soham Taneja1 D. Jude Hemanth Rachna Jain, Meenu Gupta. Deep learning based detection and analysis of covid-19 on chest x-ray images. *Springer*, pages 51, pages1690–1700, 2020.

[27] Nilanjan Saha Sanhita Basu, Sushmita Mitra. Deep learning for screening covid-19 using chest x-ray images. *ieee*, pages pp. 2521–2527, 2020.

[28] Utkarsh Sinha Vaibhav Arora Ram Bilas Pachori Soumya Ranjan Nayak, Deepak Ranjan Nayak. Application of deep learning techniques for detection of covid-19 cases using chest x-ray images: A comprehensive study. *Science Direct*, pages sciencedirect–S1746809420304717, 2021.

[29] Yi Li Chunhua Shen Yong Xi Jianpeng Zhang, Yutong Xie. Covid-19 screening on chest x-ray images using deep learning based anomaly detection. *Scinapse*, page arXiv:2003.12338v1, 2020.

[30] Mahmoud Melkemi Fadi Dornaika Ignacio Arganda-Carreras Dominique Collard Arnaud Scherperee Karim Hammoudi, Halim Benhabiles. Deep learning on chest x-ray images to detect and evaluate pneumonia cases at the era of covid-19. *Springer*, pages https://doi.org/10.1007/s10916–021–01745–4, 2020.

[31] MichaelGrass Tobias Knopp Axel Saalbach3 Ivo M. Baltruschat, Hannes Nickisch. Comparison of deep learning approaches for multi-label chest x-ray classifcation. *Springer*, pages sciencedirect–s41598–019–42294–8, 2019.

[32] Chawki Djeddi Akhtar Jamil Fadi Al-Turjman Jawad Rasheed, Alaa Ali Hameed. A machine learning-based framework for diagnosis of covid-19 from chest x-ray images. *Springer*, pages sciencedirect–S12539–020–00403–6, 2021.

[33] Syed Amer Ali; Nikhil Vallapureddy; Sridivya Mannem; Yashwanth Gudla; V Malathy. Detection of cancer in lung ct image using 3d cnn. *Journal of Intelligent Learning Systems and Applications*, pages 2022 2nd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2022, pp. 1–4, doi: 10.1109/CONIT55038.2022.9848092, 2022.

[34] S.; Khan A.; Díez I.D.L.T.; Casanova R.d.J.P.; Pifarre K.T.; Ashraf I Shafi, I.; Din. An effective method for lung cancer diagnosis from ct scan using deep learning-based support vector network. *MDPI*, pages Cancers 2022, 14, 5457. https://doi.org/10.3390/cancers14215457, 2022.

[35] M.A Riquelme, D.; Akhloufi. Deep learning for lung cancer nodules detection and classification in ct scans. *MDPI*, pages AI 2020, 1, 28–67. https://doi.org/10.3390/ai1010003, 2020.

[36] Z. Huang M. Chen, S. Huang and Z. Zhang. Detection of lung cancer from pathological images using cnn model. *IEEE*, pages pp. 352–358, doi: 10.1109/CEI52496.2021.9574590, 2021.

[37] S. Garud and S. Dhage. Lung cancer detection using ct images and cnn algorithm. *ieee*, pages pp. 1–6, doi: 10.1109/ICAC353642.2021.9697158., 2021.

[38] V. Deepa and P. M. Fathimal. Lung cancer prediction and stage classification in ct scans using convolution neural networks -a deep learning model. *IEEE*, pages pp. 1–5, doi: 10.1109/ICDSAAI55433.2022.10028880, 2022.

[39] Hosny A. Xu Y. et al Chaunzwa, T.L. Deep learning classification of lung cancer histology using ct images. *nature*, pages https://doi.org/10.1038/s41598–021–84630–x, 2021.

[40] Li W Sun R, Pang Y. Efficient lung cancer image classification and segmentation algorithm based on an improved swin transformer. *mdpi*, page 12(4):1024. https://doi.org/10.3390/electronics12041024, 2023.

[41] S. Gambhir G. Thakral and N. Aneja. Proposed methodology for early detection of lung cancer with low-dose ct scan using machine learning. *IEEE*, pages pp. 662–666, doi: 10.1109/COM–IT–CON54601.2022.9850607, 2022.

[42] I. Abunadi A. Rehman, M. Kashif and N. Ayesha. Lung cancer detection and classification from chest ct scans using machine learning techniques. *IEEE*, pages pp. 101–104, doi: 10.1109/CAIDA51941.2021.9425269, 2021.

[43] Junhao Hu Chenyang Zhang; Kang Zhou; Shenghua Gao. Chest x-ray diagnostic quality assessment: How much is pixel-wise supervision needed? *ieee*, pages ieee–9703342, 2022.

[44] Arcot Sowmya Md. Sharitul Alam, Dadong Wang. Bidirectional convolutional-lstm based network for lung segmentation of chest x-ray images. *ieee*, pages ieee–9643184, 2021.

[45] Tribikram Panthi Milan Maharjan Basanta Joshi Tejasvi Raj Pant, Ravi Kiran Aryal. Disease classification of chest x-ray using cnn. *ieee*, pages ieee–9666246, 2021.

[46] Neetu Sood Ruchika Arora, Indu Saini. Modified unet++ model: A deep model for automatic segmentation of lungs from chest x-ray images. *ieee*, pages ieee–9478101, 2021.

[47] Hritvik Jamwal Shailender Kumar Jayant Rathi, Kalp Talwadia. Depth wise convolution on chest x-ray comparative analysis with transfer learning. *ieee*, pages ieee– 9847901, 2022.

[48] Zakir Ullah Khan Chenglin Liu Ming Zhu Binquan Wang, Zeyuan Wu. Deep convolutional neural network with segmentation techniques for chest x-ray analysis. *ieee*, pages ieee– 8834117, 2019.

[49] Vivek Kanhangad G. Jignesh Chowdary. A dual-branch network for diagnosis of thorax diseases from chest x-rays. *ieee*, pages ieee– 9924588, 2022.

[50] Tribikram Panthi Milan Maharjan; Basanta Joshi Tejasvi Raj Pant, Ravi Kiran Aryal. Deep learning based multi-label chest x-ray classification with entropy weighting loss. *ieee*, pages ieee– 9666246, 2020.

[51] Hao Chen Quande Liu Xi Wang Jiaqi Xu Pheng-Ann Heng Luyang Luo, Lequan Yu. Deep mining external imperfect data for chest x-ray disease screening. *ieee*, pages ieee– 9110911, 2020.

[52] Saichao Lu; Yingmei Qin; Yanqiu Che; Haibing Guo. Classification of chest x-ray images based on superpixels and deep learning models. *ieee*, pages ieee– 9979867, 2022.