VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology & Engineering

Fall Sem – 2021 - 2022
J-Component Report

Programme : B.Tech (IT)
Course Title : Soft Computing **Course Code : ITE1015**
Date of Submission: 02-12-21

Name	Registration Number
Saksham Arora	19BIT0179
Swetank Kaushik	19BIT0164
Akshat Mishra	19BIT0156

Title : Identification of lung tissues affected from COVID-19

Abstract

Medical service providers generate and capture massive amounts of data comprising critical signals and data at a rate much above what traditional investigative approaches can achieve. As you may be aware, the RT-PCR test is one of the time-consuming and expensive methods for locating COVID19 suspects. Along these lines, it's critical to find the best combination of deep learning classifiers with Grad-CAM-based clinical images for detecting COVID-19 infection quickly and precisely during CXR and CT-Scan lung tests. The suggested technique in this research may directly assist radiologists in the speedy finding of COVID-19, which enhances the overall identification time and accuracy rate in CXR and CT-Scan images.

Keywords

CXR : - Chest X-Ray
CT : - Computerized Tomography
CNN : - Convolutional Neural Network
GRAD-CAM :- Gradient-weighted Class Activation Mapping
DL : - Deep Learning
NN : - Neural Network
nCOVID : - Novel Coronavirus Disease
AUC : - Area Under Curve
SVM : - Support Vector Machine
KNN : - K-Nearest Neighbour
VGG : - Visual Geometry Group

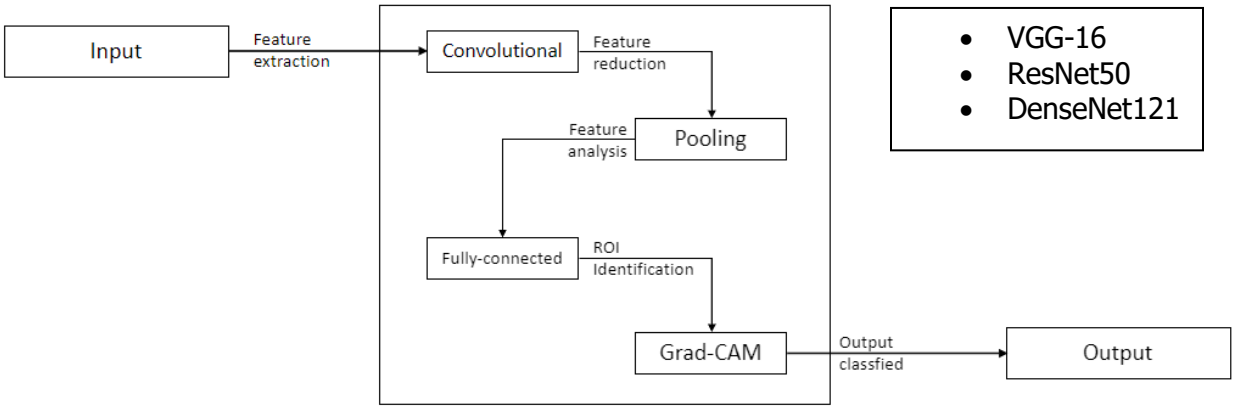
Introduction

In this study, we will look at the use of deep learning-based classifiers to distinguish the highlights from COVID-19 radiological images such as CXR and CT scan, and prepare them from a combination of Pneumonia, other respiratory illnesses, and common cases. Our model will use the Grad-CAM approach to create class enactment maps. So, a radiologist will be able to use our model both freely and closely. Because of the patients with Pneumonia, the model will also focus on False Positives (FP), which might create a stormy scenario. If a patient with Pneumonia is incorrectly identified as COVID-19 positive by our model, the patient will be contacted to be advertised in a COVID-19 area with the emergency clinic.

General Architecture

The working of our model is as follows:

- First, we will take images as the input of our system
- Then we'll perform basic transformations on the image. (converting them to black and white, and resizing them).
- After that we'll split the given dataset into train test split
- Then we are going to extract the features from the image using our convolutional layer
- The extracted features then will be reduced by removing redundant features using pooling layer
- All the features then will be analyzed using fully-connected layer
- The part that we are interested in the image is then emphasized using Grad-CAM which finally gives the output that we are interested in.



Review on various schemes (mention appropriate references)

Image acquisition:
Image acquisition was done using CT-scans and Chest X-Ray images acquired from 3 datasets which are: COVID-chest X-ray-dataset, SARS-COV-2 CT-scan, Chest X-ray Images (Pneumonia).
ref. [41],[14],[3]
Feature Extraction:
Convolutional layers extracts features from the image.
ref. [24],[25],[29]
Image Pre-processing:
Image Pre-processing is done in the pooling layer which further reduces the feature.
ref. [17],[12],[11]
ROI from NON-ROI
Grad-CAM separates ROI from non-ROI
ref. [36],[35],[34]
Classification
The fully connected layer does the feature analysis and it then passes output to GRAD-CAM which further focuses on ROI.
ref. [29],[12],[33]

Comparative study on various subtitles (using table)

Authors	Methodology or Techniques used	Advantages	Issues	Metrics used
Group I:				
Vijay Badrinarayanan et al. [2017]	Deep NN architecture	Performs non-linear up-sampling.	Training time is significantly high	mIoU metric
Giacomo Capizzi et al. [2019]	Fuzzy system combined with a neural network	Internal organs examined by the use of screening methods.	FN ratio was high	Accuracy
Retz Mahima Devarapalli et al. [2019]	Support Vector Machine	Detection and confirmation of early detection of cancer.	Except SVM other models have low accuracy.	Sensitivity
Mohammad Jamshidi et al. [2018]	Deep Learning	Accelerate the process of diagnosis of the	Novel approaches needed for	Accuracy

		COVID-19 disease.	problems of this level of complexity.	
Qiao Ke et al. [2019]	Neuro-heuristic approach	The method is flexible and has low computational burden.	Improvements in the developed methods	Accuracy
Michelle Livne et al. [2019]	U-Net	Proposed alternative for classic rule-based segmentation algos	Improved segmentation and methodologies to deal with specific pathologies.	Dice coefficient
Jingdong Yang et al. [2021]	Dilated MultiResUNet	superior accuracy and achieve better generalization performance	more test required to improve the system	Dice
Michał Wieczorek et al. [2020]	NN powered predictor	Unified architecture that makes it easy to work as predictor	Low network efficiency when amount of data is low	Accuracy
Pranav, Jothi V et al. [2020]	Deep Learning	The model has achieved high sensitivity specificity	Larger dataset required for higher accuracy	Specificity
Kanakaprabha. S et al. [2021]	Deep Learning	Even with abnormal condition prediction can be done	Ensembling of different models to improve the accuracy	Accuracy
Harsh Agrawal et al. [2020]	Deep Learning	Image processing can be used to improve the performance of	classifiers with more than one model can be considered	F1 score

		deep learning models		
Gautam R.G. et al. [2021]	Deep Neural Networks with PyTorch	Proposed reverse-transfer learning	-	mIoU - mean Intersection over Union
Simona Condaragiu et al. [2021]	DNNs	Comparative analysis of different models and data	low quality images harms the system very drastically	F1, AUC
Hayat O. Alasasfeh et al. [2020]	Deep learning	Model is fast and accurate at the same time	Large dataset can't be inserted into the NN at once	F1 score
Md. Foysal et al. [2021]	Ensemble DNNs	Overall false prediction is decreased	High GPU power was required	F1 score
Mohd. H. Naviwala et al. [2021]	DNNs	Performance comparison of different DNNs	Vanishing gradient	AUC
Asif Iqbal Khan et al. [2020]	CoroNet, a Deep Neural Network Model	Tailored COVID-19 detection through custom XceptionNet	Their model was overfitting the test set.	Classification Accuracy
Tulin Ozturk et al. [2020]	DarkCovidNet Model, a neural network based on customization of DarkNet-19	End-to-end architecture which requires only raw X-ray image to determine CoVID status,	Model made incorrect predictions in poor quality X-ray images	Accuracy
Ezz EL-DIn Hemdan et al. [2020]	COVIDX-Net, Framework of various Deep Neural Networks	Use of Stochastic Gradient Descent for faster running time	Batch Gradient Descent would be drastically better for a dataset of this size, so as to	F1-Score

			reach global minimum	
Wei Li et al. [2020]	RNN Hybrid with ResNet	Using ResNet in Hybrid with RNN for faster prediction.	RNN based Hybrid requires large labelled training set, and absence of this is significantly hurting performance	T-Test
Group II:				
Abbas A et al. [2021]	DeTraC	DeTraC can deal with any irregularities in the image dataset	Limited availability of annotated medical images makes it hard to classify	sensitivity
Sharvari Kalgutkar et al. [2019]	Transfer Learning	The work achieves increased accuracy values by 5–7 %.	Patients historical data can be used for more detailed result	Accuracy, F1 score
Varalakshmi P. et al. [2o21]	Transfer learning	Used sonograms for detection	Sensitive to outliers	F1 score
Shuai Wang et al. [2020]	Modified Inception transfer learning model	Identifies Region of Interest and then performs feature extraction and classification in ROI.	High number of False Negatives due to improper acclimation of ROI in cases with Pneumonia	AUC
Y. Pathak et al. [2020]	Deep Transfer Learning	Used 10-fold cross-validation to prevent overfitting	Optimal selection of hyper-parameters is not considered	Precision

Muhammad E.H Chowdhury et al. [2020]	Transfer learning with Softmax Function	Softmax activation function enhances result for multi class classification between pneumonia and covid	Artificial dataset created by just rotation is not helpful to the model at all, and is creating a bias.	F1-Score
Neha Gianchandani et al. [2020]	Deep transfer learning	Improve the generalization capability of the classifier for binary and multi-class problems.	More data can be levered to improve the feature extraction capabilities model.	F1-Score
Varan Singh Rohila et al. [2021]	Ensemble learning	Proposed a new framework to detect traces of COVID-19	Requires 2 or more GPUs to run	AUC
Stefanus Tao Hwa Kieu et al. [2021]	Ensemble classification	Achieved overall accuracy of 97.9o%	The image resolution should be increased	Accuracy
Ojas A. Ramwala et al. [2020]	Deep Residual Network	solved the issue of accuracy reduction with the increase in depth of the network	Models can be trained only on balanced datasets	Accuracy, Positive/Negative Predicted Value
Muhammad A.N, et al. [2021]	Attention Based Residual Network	Diagnostic Learning On Lung Ultrasound	larger dataset could've helped	Accuracy
Group III:				
Michael J. Horry et al. [2020]	Multimodal Imaging	The model is capable of classifying COVID-19 vs	Data was not sufficient	F1 score

		Pneumonia conditions		
Ferhat Bozkurt et al. [2021]	LBP (local binary pattern)	Over 99% accuracy is achieved with the LBP+SVM on CXR images	Model is biased as recall is very low	Accuracy, precision, recall
Rajarshi Bhadra et al. [2020]	Multi-layered pre-processing	Accuracy of 99.1% achieved	Training time increased 1o fold	F1 score, accuracy
Ahmed M.F. et al. [2020]	XGBoost	Model showed promising results for further studies	Training time was reported quite high	AUC
Fian Yulio Santoso et al. [2020]	Batch normalisation	Proposed model avoids overfitting	Testing and tuning still required	Accuracy
R.M. Rawat et al. [2020]	Comparative study	Compared different models and proposed the best among them	Models weren't tuned	F1, accuracy
Buyut Khoirul Umri et al. [2020]	Contrast Limited Adaptive Histogram Equalization	Study to compare CLAHE with existing models	Small dataset of 40 images	Accuracy
Harsh Panwar et al. [2o2o]	NCOVnet, a Neural Network based Model	Model prevented bias against train test split	-	F1 Score
Group IV:				
P.K. Gupta et al [2020]	Deep Learning using Grad-CAM	Gradient Weighted Class Activation Mapping results in really accurate binary classification	Slightly biased model towards COVID and Pneumonia due to GRAD-CAM	F1-Score

		between normal and infected lung scans.		
Luca Brunese et al. [2020]	Pipelining of two Visual Geometry Group Models	1st pipeline classifies between healthy and infected lungs, 2nd pipeline then identifies COVID	Skewed dataset, and heavy processing needed for prediction	F1 Score
Hap Quan et al. [2021]	Detection through DenseCapsNet, a Capsule Neural Network method.	Can perform sound analysis without much data and any kind of pre-processing	FP score was high for Pneumonia, which results in low F1 score.	Sensitivity, Specificity
Ali Narin [2021]	Specific feature selection which is then passed on custom SVM based Model	Used Ant Colony Optimization technique using results from SVM and KNN to customize SVM	Overfitting due to mini batch size of 10 and only 30 iterations of Neural Model	Accuracy
Warda M. Shaban et al. [2020]	Enhanced KNN based Classifier	Used Hybrid selection on extracted features to plot cluster space and hence classifying on the basis of vector being nearest to the specific cluster	Extensive feature extraction is needed, and large dataset is also needed to plot COVID- non COVID cluster space	Accuracy
Ferhat Ucar et al. [2020]	COVIDiagnosis-Net, a Deep Bayes-SqueezeNet based model	Extremely small model size, makes hardware deployment easily feasible.	Generated data tuples by adding shear and brightness to the original x-ray which is just	Accuracy

			adding additional bias.	

Group-I
In group-I, researchers used Deep Neural Network as their major component. They have worked majorly on feature selection and classification. Asif Iqbal Khan et al. [32] have used CNN for feature selection which proved to be quite helpful in comparison to Wei Li et al. [34] and Ali Narin [38] as they have used RNN and SVM based models respectively.
Group-II
In group-II, researchers used transfer learning as their major component. In this group, Sharvari Kalgutkar et al. [42] achieved accuracy values by 5-7% but found dataset scarcity as an issue. Also, Varan Singh Rohila et al. [16] proposed a new framework to detect the virus. Overall, these models proved to be better at dealing with outliers, distorted and low resolution images (except for Varalakshmi P. et al. [29], who used Sonograms for detection).
Group-III
Here in Group-III, Most of the papers were related to Image Pre-Processing, and important papers in this section were Michael J. Horry et al.[14], Ferhat Bozkurt et al.[21] and Buyut Khoirul et al. [30], They had used CLAHE (Contrast Limited Adaptive Histogram Equalization), LBP (Local Binary Pattern) and Multimodal Imaging respectively, to better understand the data, by comparing with adjacent pixels.
Group-IV
Papers in Group-IV were mostly based around Region of Interest (ROI) detection and feature extraction, few Note-worthy papers here were P.K. Gupta et al. [36] and Luca Burnese et al. [33], They elegantly showed application of ROI identification through Grad-CAM and VGG, and their accuracy regarding infected and healthy lungs were among the top, then one another important paper was Warda M. Shaban et al. [38] extracted features through a combination of SVM and KNN based models.

Conclusions **(Among various schemes which is best for which application)**

GROUP-I
Feature Extraction and Reduction + Classifier
Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various objects in the image and be able to differentiate one from the other. CNN is the best model for image processing so we have chosen CNN for our objective.
An input layer, a hidden layer, and an output layer make up a convolutional neural network. Any middle layer in a fed-forward neural network is considered to be hidden since the activation function and the final convolution hide their inputs and outputs. The hidden layers in a convolutional neural network are the layers that perform the convolution. Typically, this entails a layer that performs the dot product of the convolution kernel with the input matrix of the

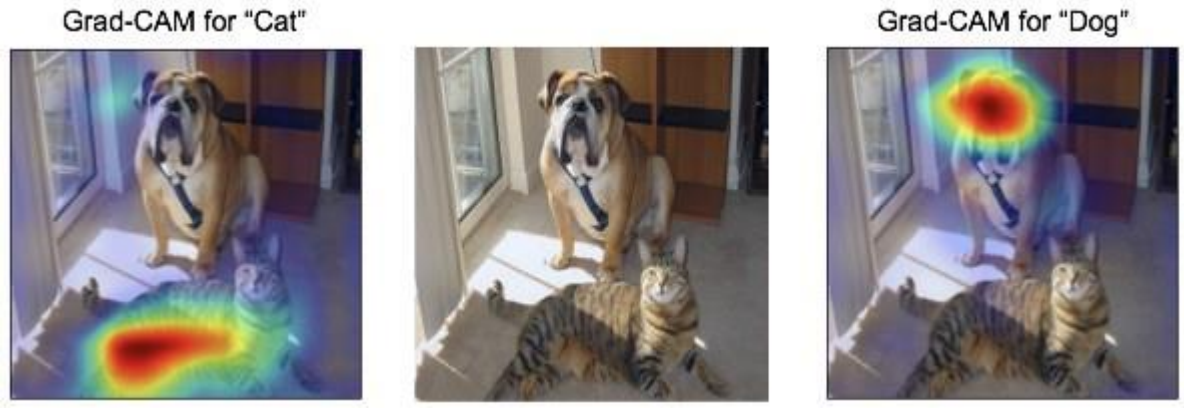
layer. The activation function of this product is normally ReLU, and it is generally a Frobenius inner product. The convolution operation creates a feature map as the convolution kernel slides along the input matrix for the layer. This contributes to the input of the next layer. other layers, such as pooling layers, completely connected layers, and normalisation layers, come next. 17 Except for convolution operations that take place in one or more layers of a CNN, CNNs are quite similar to vanilla neural networks.

Group-IV
ROI Identification and Analysis
Gradient-weighted Class Activation Mapping (Grad-CAM) utilizes the gradients of any target feature flowing into the final convolutional layer to produce a coarse localization map emphasising the essential regions in the image for predicting the concept.

Softmax is a mathematical process that converts a vector of integers into a vector of probabilities, with the probabilities of each value proportional to the vector's relative scale. The softmax function is most commonly used in applied machine learning as an activation function in a neural network model. The network is set up to produce N values, one for each class in the classification task, and the softmax function is used to normalise the outputs, converting them from weighted sum values to probabilities that add up to one.

MODULE DESCRIPTIONS

Grad-CAM:
Gradient-weighted Class Activation Mapping (Grad-CAM) utilizes the gradients of any target feature flowing into the final convolutional layer to produce a coarse localization map emphasizing the essential regions in the image for predicting the concept.
Grad-CAM can be used for weakly-supervised localization, *i.e.* determining the location of particular objects using a model that was trained only on whole-image labels rather than explicit location annotations.



Convolutional Neural Network (CNN):
An input layer, a hidden layer, and an output layer make up a convolutional neural network. Any middle layer in a fed-forward neural network is considered to be hidden since the activation function and the final convolution hide their inputs and outputs. The hidden layers in a convolutional neural network are the layers that perform the convolution. Typically, this entails a layer that performs the dot product of the convolution kernel with the input matrix of the layer. The activation function of this product is normally ReLU, and it is generally a Frobenius inner product. The convolution operation creates a feature map as the convolution kernel slides along the input matrix for the layer. This contributes to the input of the next layer. other layers, such as pooling layers, completely connected layers, and normalization layers, come next. Except for convolution operations that take place in one or more layers of a CNN, CNNs are quite similar to vanilla neural networks.

The layer of a simple neural network is represented by the equation below.

$$z^{[1]} = g(W^{[1]}a^{[0]} + b^{[1]})$$

Where z[1] is the current layer, a[o] is the input layer, W[1] represents the weights for the input layer and b[1] is the bias.

VGG-16
VGG16 is a convolutional neural network model developed by K. Simonyan and A. Zisserman of the University of oxford in their paper "Very deep convolutional networks for large-scale image recognition." In ImageNet, a dataset of over 14 million images belonging to 1ooo classes, the model achieves 92.7 percent top-5 test accuracy. It was one of the well-known models shown at the ILSVRC-2o14. It improves on AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layers, respectively) with several 33 kernel-sized filters one

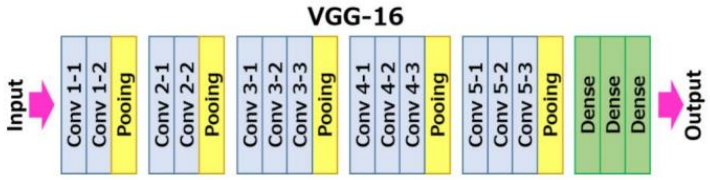
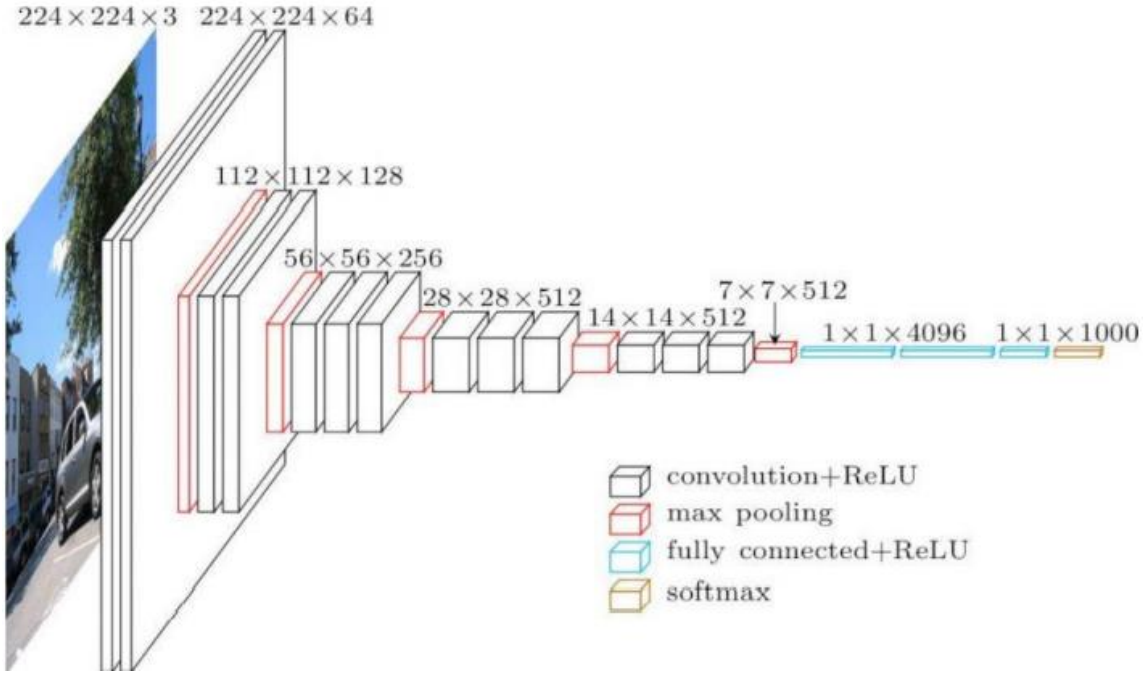


Fig 5 VGG-16 Basic Architecture

The Architecture depicted below is VGG16.



Unfortunately, VGG.net has two significant flaws:
• Training is very slow.
• The network architecture weight (in relation to disk/bandwidth) is rather large.
VGG16 is over 533MB in size because to its depth and the number of completely connected nodes. This makes VGG deployment a difficult process.

We used VGG16 with imagenet weights and all the internal layers with their pre-trained weights in our project. To get the outcome, we used pooling, densification, dropout, and a flattened layer. A softmax function is also used in the final layer. The model was created with Adam as the optimizer and accuracy as the optimization matrix.

Code to train VGG-16:

```
baseModel = VGG16(weights="imagenet", include_top=False, input_tensor=Input(shape=(224,224,3)))
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
vggmodel = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
```

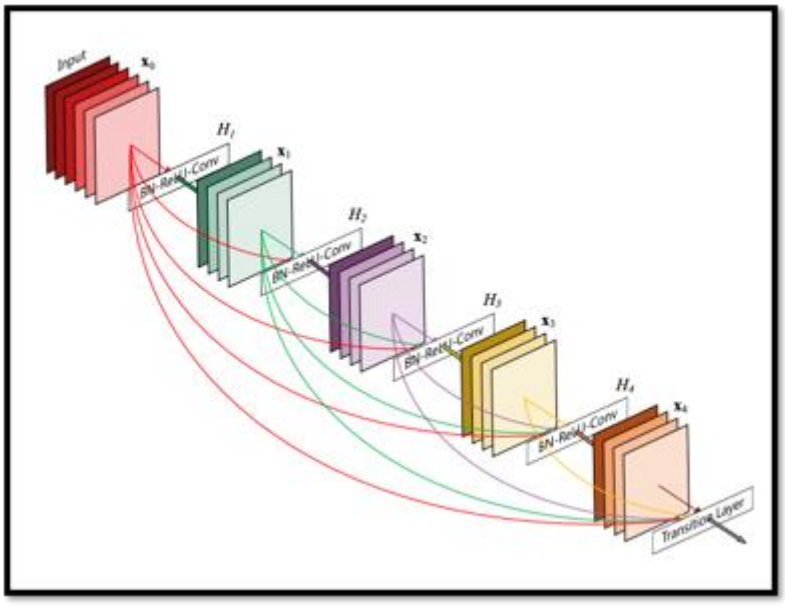
```
#Using Adam optimizer
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
vggmodel.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy","AUC"])
```

```
# train the head of the network
print("[INFO] training head...")
H = vggmodel.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)
```

DENSE NET121

DenseNet (Dense Convolutional Network) is an architecture that employs fewer connections between layers to make deep learning networks increasingly deeper while also making them more economical to train. DenseNet is a convolutional neural network with each layer connected to all other layers deep in the network, for example, the first layer is connected to the second, third, fourth, and so on; the second layer is connected to the third, fourth, fifth, and so on. This is done to guarantee that the greatest amount of data can move between the levels of the network. It does not integrate features by putting them together, like ResNets do, but rather by combining them. The 'ith' layer, for example, has an I input and includes all of the previous convolutional blocks' feature maps.

All succeeding 'eye-eye' layers get their own feature maps. It uses the '(I(I+1))/2' connection to link the network, rather than the 'I' connection used in normal deep learning designs. Because no unnecessary feature mappings must be trained, it uses fewer parameters than typical convolutional neural networks.



Traditional fed-forward neural networks link the outputs of the previous layer to the next layer after conducting a composite of operations.

We can look at feature maps as network data. Each layer has access to its previous feature maps and, as a result, to the collective knowledge. Then, in a feature map of solid k information, each layer adds a new piece of information to the collection.

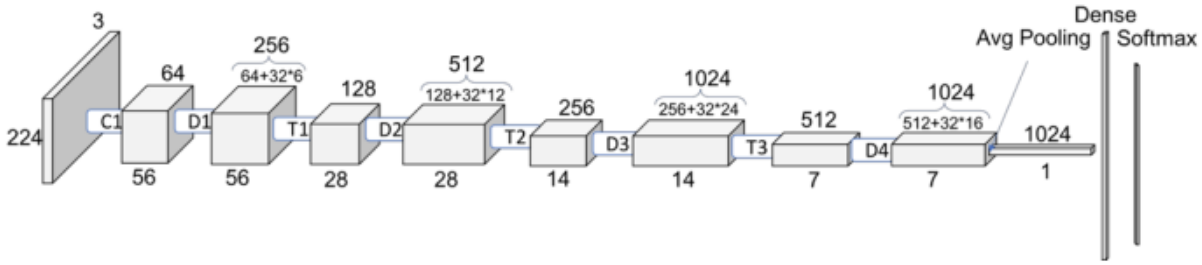


Fig 8 Dense Net 121, Dx-Dense Block, Tx- Transition Block

DenseNets uses a fundamental connection rule to merge the qualities of identity mapping, deep observation, and various depths. They allow network features to be reused, resulting in more compact and, in our experience, more accurate model learning. Because of their compact internal representations and minimal feature redundancy, DenseNets can be useful feature extractors for a variety of computer vision applications based on convolutional features.

Code to train DenseNet121:

```
baseModel = DenseNet121(input_shape = (224, 224, 3),include_top = False, weights = None)
headModel = baseModel.output
headModel=BatchNormalization()(headModel)
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
dense_model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
```

```
#Using Adam optimizer
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
dense_model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
# train the head of the network
print("[INFO] training head...")
H = dense_model.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)
```

RESNET50

ResNet, or Residual Networks, is a traditional neural network that is used as a backbone in many computer vision applications. The winner of the ImageNet challenge in 2015 was this model. The fundamental breakthrough with ResNet was that it allowed us to successfully train extremely deep neural networks with 150+ layers. Due to the problem of vanishing gradients, training very deep neural networks was challenging prior to ResNet.

However, just stacking layers together will not increase network depth. Because of the infamous disappearing gradient problem, which occurs when a gradient is back-propagated to earlier layers, repeated multiplication can cause the gradient to become extremely tiny. As a result, as the network becomes more complex, its performance becomes saturated or even degrades fast.

The concept of skip connection was initially introduced by ResNet. The graphic below shows how to skip a connection. The figure on the left shows stacking convolution layers one on top of the other. on the right, we stack convolution layers as before, but now we additionally add the original input to the convolution block's output. This is referred to as a skip connection

The reason why skipping connections works is because:

- They allow the model to learn an identity function, ensuring that the higher layer will perform at least as well as the lower layer, if not better.
- They mitigate the problem of disappearing gradient by enabling gradient to flow along an alternate shortcut channel.

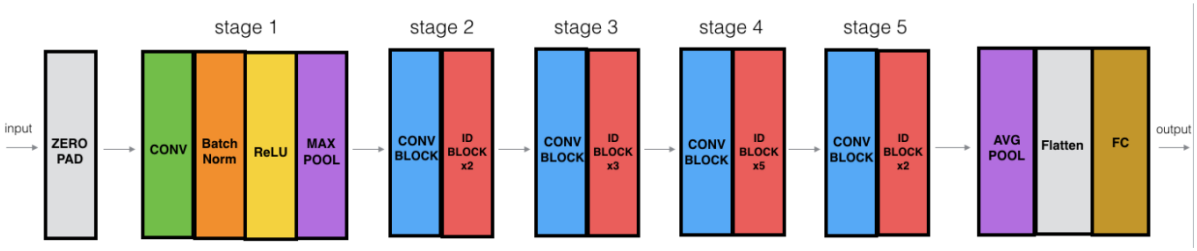


Fig 12 ResNet50 model

The ResNet-50 model is made up of five stages, each having its own convolution and identity block. Each convolution block contains three convolution layers, and each identity block contains three as well. over 23 million trainable parameters make up the ResNet-50.

Code to train ResNet50

```
baseModel = ResNet50(input_shape = (224, 224, 3),
  include_top = False, weights = None)
headModel = baseModel.output
headModel=BatchNormalization()(headModel)
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
resnet = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
```

```
#Using Adam optimizer
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
resnet.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
# train the head of the network
print("[INFO] training head...")
H = resnet.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)
```

DATASET DISCRIPTION

COVID-chest X-ray-dataset – This dataset was compiled by Cohen et al. and is widely utilised by numerous scholars. The database is kept up to date with the most recent updates. This dataset has 3616 COVID-19 positive cases along with 10,192 Normal, 6012 Lung Opacity (Non-COVID lung infection), and 1345 Viral Pneumonia images.
database of COVID-19 x-ray images from the Italian Society of Medical and Interventional Radiology (SIRM) COVID-19 DATABASE, Novel Corona Virus 2019 Dataset

Chest X-ray Images (Pneumonia) – This dataset was developed by Kermany et al. A substantial number of OCT and CXR images are publically available in this dataset. We used the X-Rays section of this dataset in the studies, which included 5863 photos of Pneumonia and Normal individuals.

Description of the metadata:
The columns specified in the metadata and their details are mentioned below.

- Patientid: internal identifier, just for this dataset
- offset: Number of days since the start of symptoms or hospitalization for each image. If a report indicates "after a few
- Sex: Male (M), Female (F), or blank
- Age: Age of the patient in years
- Finding: Type of pneumonia
- Survival: Yes (Y) or no (N) or blank if unknown
- Intubated: Yes (Y) if the patient was intubated (or ventilated) at any point during this illness

- or No (N) or blank if unknown.
- Intubation present: Yes (Y) or no (N) or blank if unknown
- Went_icu: Yes (Y) if the patient was in the ICU (intensive care unit) or CCU (critical care
- unit) at any point during this illness or No (N)
- In_icu: Yes (Y) if the patient is in the ICU (intensive care unit) or CCU (critical care unit) or
- No (N) or blank if unknown.
- Needed_supplement: Yes (Y) if the patient required supplemental oxygen at any point
- during this illness or No (N) or blank if unknown
- Extubated: Yes (Y) if the patient was successfully extubated or No (N) or blank if unknown
- Temperature: Temperature of the patient in Celsius at the time of the image
- Po2_saturation: partial pressure of oxygen saturation in % at the time of the image
- Leukocyte_count: white blood cell count in units of 10^3/uL at the time of the image
- Neutrophil_count: neutrophil cell count in units of 10^3/uL at the time of the image
- Lymphocyte_count: lymphocyte cell count in units of 10^3/uL at the time of the image
- View: Posteroanterior (PA), Anteroposterior (AP), AP Supine (APS), or Lateral (L) for Xrays; Axial or Coronal for CT scans
- Modality: CT, X-ray, or something else
- Date: Date on which the image was acquired
- Hospital: Hospital name, city, state, country
- Folder: Name of folder containing the scan
- Filename: Name with extension
- Doi: Digital object identifier (DoI) of the research article
- url: URL of the paper or website where the image came from
- license: License of the image such as CC BY-NC-SA. Blank if unknown

Normal:

Covid:

Lung Opacity:

Pneumonia:

MODULES ARCHITECTURE
LOSS FUNCTIOS
Binary Crossentropy
The loss function binary crossentropy is used in binary classification applications. These are activities in which you must choose between two options (yes or no, A or B, o or 1, left or right). Several independent such questions, such as multi-label classification or binary image segmentation, can be answered at the same time.
on many two-category tasks, this loss is equivalent to the average of the categorical crossentropy loss.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Fig 22 Binary crossentropy

p(y) is the predicted chance of a point being green for all N points, where y is the label (1 for green points and 0 for red points).
If each classification can be reduced to a binary choice (i.e. yes or no, A or B, 0 or 1), the binary crossentropy is a very convenient way to train a model to solve several classification problems at the same time.

ACTIVATION FUNCTIONS
Relu (Rectified Linear Unit)
The rectified linear activation function, or ReLU, is a piecewise linear function that outputs the input directly if it is positive and zero if it is negative. Many type of neural networks use it as the default activation function since it is quicker to train and typically results in higher performance.

The rectified linear activation function is a simple computation that returns the input value, or 0.0 if the input is 0.0 or less.
A piecewise linear function, also known as a hinge function, is defined as a function that is linear in half of the input domain but nonlinear in the other.

Softmax
Softmax is a mathematical process that converts a vector of integers into a vector of probabilities, with the probabilities of each value proportional to the vector's relative scale.

The softmax function is most commonly used in applied machine learning as an activation function in a neural network model. The network is set up to produce N values, one for each class in the classification task, and the softmax function is used to normalise the outputs, converting them from weighted sum values to probabilities that add up to one. Each number in the softmax function's output is interpreted as the likelihood of membership in each class.

In the output units of deep neural networks that estimate a multinomial probability distribution, the softmax function is used as the activation function.
Softmax is used as the activation function for multi-class classification problems requiring class membership on more than two labels.
The function can be used as an activation function for a hidden layer in a neural network, although this is less common. It may be used when the model internally needs to choose or weight multiple different inputs at a bottleneck or concatenation layer.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector

Fig 23 Softmax function

OPTIMIZER

Adam

Adam is a optimization algorithm which can be used to repeatedly adjust network weights based on training data instead of the traditional stochastic gradient descent procedure.

In their 2o15 ICLR paper (poster) titled "Adam: A Method for Stochastic optimization," Diederik Kingma from openAI and Jimmy Ba from the University of Toronto presented Adam.

Adam, according to the authors, presents the benefits of two other stochastic gradient descent extensions. Specifically:

- Adaptive Gradient Algorithm (AdaGrad), which maintains a per-parameter learning rate that increased efficiency on applications with sparse gradients (e.g. natural language and computer vision).

- Root Mean Square Propagation (RMSProp), which also maintains per-parameter learning rates that are adjusted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it changes). This means that the algorithm performs well

on both online and non-stationary problems (such as noisy data).Adam realizes the benefits of both AdaGrad and RMSProp.

Rather of modifying parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam uses the average of the gradients' second moments (the uncentered variance).

The method calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 govern the decay rates of these moving averages.
The beginning values of the moving averages, as well as beta1 and beta2 values close to 1.0 (recommended), result in a bias of moment equations towards zero. This prejudice is overcome by first calculating bias-corrected estimates and then bias-corrected estimates.

MODULE SUMMARY

VGG-16

Below is the model summary for the model used VGG-16. We have used the pretrained layers of VGG-16 and added Dense, Dropout layers along with a Softmax function.

Layer (type)	Output Shape	Param #
input_1 (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

```
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

average_pooling2d (AverageP  (None, 1, 1, 512)        0
ooling2D)

flatten (Flatten)           (None, 512)               0

dense (Dense)               (None, 64)                32832

dropout (Dropout)           (None, 64)                0

dense_1 (Dense)             (None, 2)                 130

=================================================================
Total params: 14,747,650
Trainable params: 32,962
Non-trainable params: 14,714,688
```

DENSE NET121

Below is the model summary for the model used DenseNet121. We have used the pretrained layers of DenseNet121 and added Dense, Dropout layers along with a Softmax function.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 224, 224, 3)]	0	[]
zero_padding2d (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_2[0][0]']
conv1/conv (Conv2D)	(None, 112, 112, 64)	9408	['zero_padding2d[0][0]']
conv1/bn (BatchNormalization)	(None, 112, 112, 64)	256	['conv1/conv[0][0]']
conv1/relu (Activation)	(None, 112, 112, 64)	0	['conv1/bn[0][0]']
zero_padding2d_1 (ZeroPadding2D)	(None, 114, 114, 64)	0	['conv1/relu[0][0]']
pool1 (MaxPooling2D)	(None, 56, 56, 64)	0	['zero_padding2d_1[0][0]']
conv2_block1_0_bn (BatchNormalization)	(None, 56, 56, 64)	256	['pool1[0][0]']
conv2_block1_0_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block1_0_bn[0][0]']
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 128)	8192	['conv2_block1_0_relu[0][0]']
conv2_block1_1_bn (BatchNormalization)	(None, 56, 56, 128)	512	['conv2_block1_1_conv[0][0]']
conv2_block1_1_relu (Activation)	(None, 56, 56, 128)	0	['conv2_block1_1_bn[0][0]']

```
conv5_block16_2_conv (Conv2D)  (None, 7, 7, 32)     36864       ['conv5_block16_1_relu[0][0]']

conv5_block16_concat (Concaten  (None, 7, 7, 1024)  0           ['conv5_block15_concat[0][0]',
ate)                                                             'conv5_block16_2_conv[0][0]']

bn (BatchNormalization)        (None, 7, 7, 1024)   4096        ['conv5_block16_concat[0][0]']

relu (Activation)              (None, 7, 7, 1024)   0           ['bn[0][0]']

batch_normalization (BatchNorm  (None, 7, 7, 1024)  4096        ['relu[0][0]']
alization)

average_pooling2d_1 (AveragePo  (None, 1, 1, 1024)  0           ['batch_normalization[0][0]']
oling2D)

flatten (Flatten)              (None, 1024)         0           ['average_pooling2d_1[0][0]']

dense_2 (Dense)                (None, 64)           65600       ['flatten[0][0]']

dropout_1 (Dropout)            (None, 64)           0           ['dense_2[0][0]']

dense_3 (Dense)                (None, 2)            130         ['dropout_1[0][0]']

==================================================================================================
Total params: 7,107,330
Trainable params: 67,778
Non-trainable params: 7,039,552
__________________________________________________________________________________________________
```

RESNET50

Below is the model summary for the model used ResNet5o. We have used the pretrained layers of ResNet50 and added Dense, Dropout layers along with a Softmax function.

```
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_3[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           ['pool1_pad[0][0]']
```

```
batch_normalization_1 (BatchNo  (None, 7, 7, 2048)  8192        ['conv5_block3_out[0][0]']
rmalization)

average_pooling2d_2 (AveragePo  (None, 1, 1, 2048)  0           ['batch_normalization_1[0][0]']
oling2D)

flatten (Flatten)              (None, 2048)         0           ['average_pooling2d_2[0][0]']

dense_4 (Dense)                (None, 64)           131136      ['flatten[0][0]']

dropout_2 (Dropout)            (None, 64)           0           ['dense_4[0][0]']

dense_5 (Dense)                (None, 2)            130         ['dropout_2[0][0]']

==================================================================================================
Total params: 23,727,170
Trainable params: 135,362
Non-trainable params: 23,591,808
```

EVALUATION METRICS

We'll be using the confusion matrix to calculate the precision, recall, and accuracy for assessing the model.
These were calculate using the following parameters and equations:

• TP (True Positivity): If a CoVID-19 infected person is identified as CoVID-19 infected,

• TN (True Negative): If a person's non-CoVID-19 status is correctly identified.

• FP (False Positive): represents incorrect detection when a healthy person is found to be positive for CoVID-19.

• FN (False Negative): represents incorrect detection when a CoVID-19-infected person is detected as a normal one.

The percent of CoVID-19 correct positive detection i.e (how correct our model is) is calculated with precision.
The percent of CoVID-19 correct actual detection i.e (how many actual COVID-19 patients were correctly classified by our model) is calculated with recall.

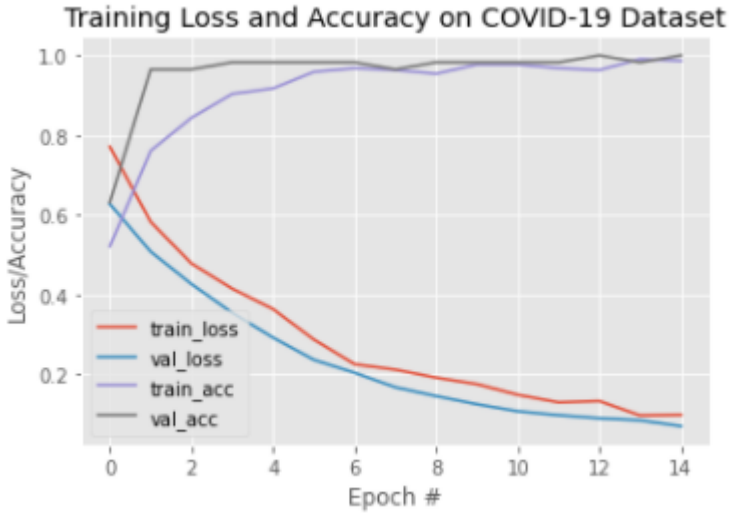
$$\text{Precision} = \frac{TP}{TP+FP} * 100$$

$$\text{Recall} = \frac{TP}{TP+FN} * 100$$

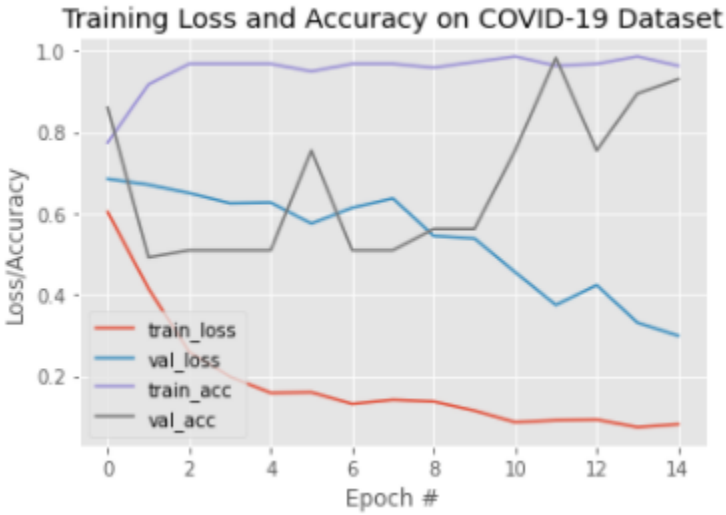
$$\text{Accuracy} = \frac{TP+TN}{Total\ no.of\ predictions}$$

RESULTS AND DISCUSSION

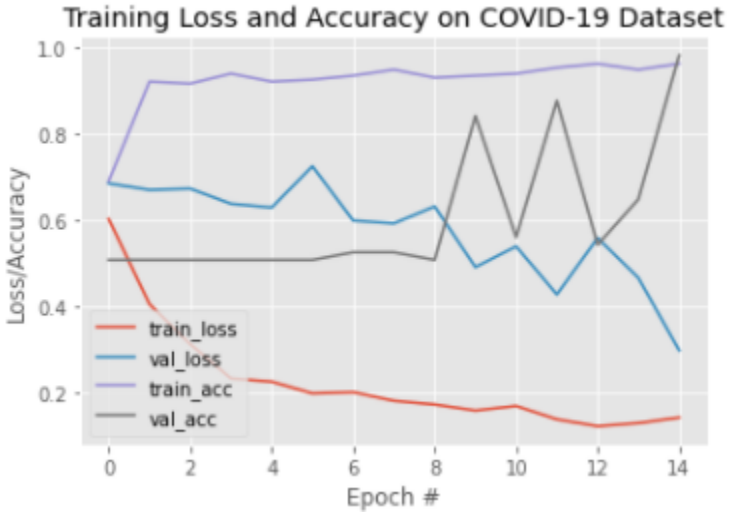
We trained all the models for 15 epochs and used accuracy and loss as the optimization matrix, and then produced the confusion matrix to get more intuitive results.
Also, we plotted the curve of training loss and accuracy vs. epochs for each model.



VGG16



DenseNet121



ResNet50

We also produced confusion matrix and calculated precision and recall for our models. The results are produced using the test set.

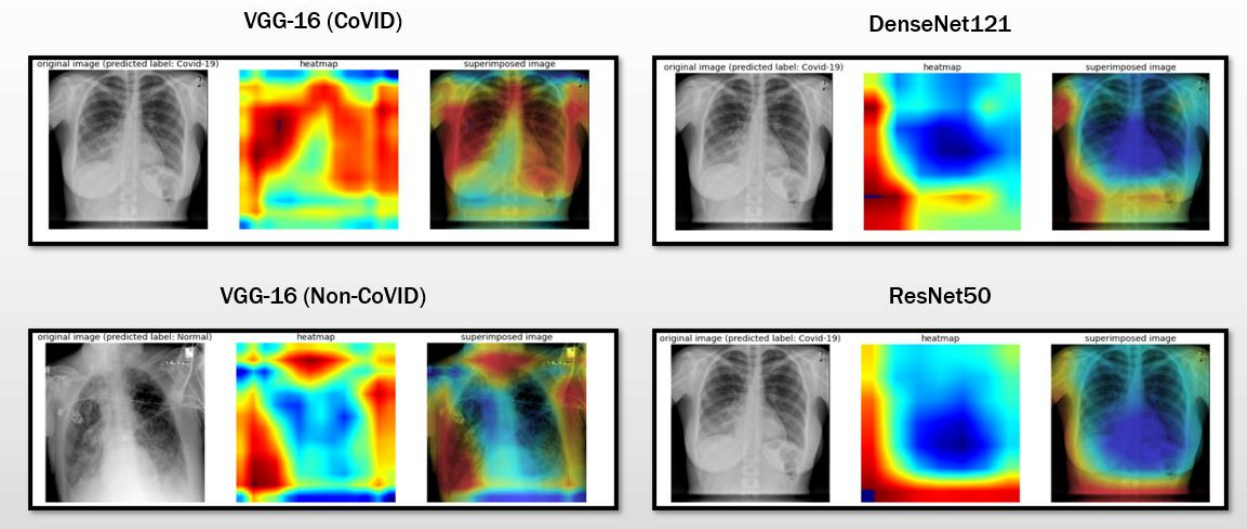
VGG16: [[27 1]
[0 29]]

DenseNet121: [[20 8]
[0 29]]

ResNet50: [[19 9]
[0 29]]

Model	Precision	Recall	Accuracy	F-Score
VGG16	0.9643	1.000	0.9825	0.9818
DenseNet121	0.7143	1.000	0.8596	0.8333
ResNet50	0.6786	1.000	0.8421	0.8085

For producing visual explanations for our Convolutional Neural Network (CNN) –based models we use Grad – CAM based heatmaps. Below are the heatmaps produced by each model for the same image of the class: diagnosed as CoVID-19. Note that these heatmaps are produced from weights calculated by the last convolutional layer in each network.



Comparative Study

Basic comparative studies against the paper which we found similar and relevant to out project.

Parameters	[19]	[31]	Proposed
Dataset	COVID-19 Radiography Database	-No longer available-	COVID-19 Radiography Database, Chest X-Ray Images (Pneumonia)
Training – Test Ratio	80:20	70:30	80:20
Validation Accuracy	-	-	96.77%, 85.96%, 84.21%
Testing Accuracy	97.69%	98.82	98.25%, 98.16%, 96.31%
Precision	-	0.95	0.9643, 0.7143, 0.6786
F-Score	-	0.94	0.9818, 0.8333, 0.8085
Recall	-	0.98	1.0, 1.0, 1.0
Epochs	45	20	15
Model	ResNet32	nCOVnet	VGG-16, ResNet50, DenseNet121

Conclusion and Future Work

In this work, we used binary image classification to identify CoVID-19 and Non-CoVID19 positive patients. According to the study, non-CoVID-19 positive patients may have Pneumonia or other respiratory problems. Several experiments have employed CXR and CT-Scan images of the chest to detect CoVID-19 patients.

In order to make our deep learning model more interpretable and explainable, we used a color visualisation method utilizing the GRAD-CAM technique in our trials which could be further used in other research or acedemia purposes, for eg, using them in a bigger application or passing them in a pipeline.

References

[1]Neural network powered COVID-19 spread forecasting model
[2]Performance analysis of lightweight CNN models to segment infectious lung tissues of COVID-19 cases from tomographic images
[3]COVID-19 Detection Through Transfer Learning Using Multimodal Imaging Data
[4]COVID 19, Pneumonia and Other Disease Classification using Chest X-Ray images
[5]COVID-19 Diagnosis from Chest Radiography Images using Deep Residual Network
[6]SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation
[7]Small Lung Nodules Detection Based on Fuzzy-Logic and Probabilistic Neural Network With Bioinspired Reinforcement Learning
[8]Lung Cancer Detection of CT Lung Images
[9]Artificial Intelligence and COVID-19: Deep Learning Approaches for Diagnosis and Treatment
[1o]A neuro-heuristic approach for recognition of lung diseases from X-ray images
[11]A U-Net Deep Learning Framework for High Performance Vessel Segmentation in Patients With Cerebrovascular Disease
[12]Neural network powered COVID-19 spread forecasting model
[13]A Novel Coronavirus from Patients with Pneumonia in China, 2o19
[14]High-Resolution Representations for Labeling Pixels and Regions
[15]Deep High-Resolution Representation Learning for Human Pose Estimation
[16]Deep Learning Assisted Covid-19 Detection using full CT-scans
[17]A Survey of Deep Learning for Lung Disease Detection on Medical Images: State-of-the-Art, Taxonomy, Issues and Future Directions
[18] COVID-19 Pandemic in India: Through the Lens of Modeling

[19] Attention Based Residual Network for Effective Detection of COVID-19 and Viral Pneumonia
[2o] Evaluation of Convolutional Neural Networks for COVID-19 Detection from Chest X-Ray Images
[21] Local Binary Pattern Based COVID-19 Detection Method Using Chest X-Ray Images
[22] Covid Detection from CXR Scans using Deep Multi-layered CNN
[23] Using CNN-XGBoost Deep Networks for COVID-19 Detection in Chest X-ray Images
[24] Application of Deep Learning for Early Detection of COVID-19 Using CT-Scan Images
[25] COVID-19 Detection using Convolutional Neural Network Architectures based upon Chest X-rays Images
[26] Deep Learning Approach for COVID-19 Detection Based on X-Ray Images
[27] Automatic Detection of COVID-19 from Chest X-ray Images with Convolutional Neural Networks
[28] Performance Analysis of Deep Learning Frameworks for COVID 19 Detection
[29] A Transfer Learning Model for COVID-19 Detection with Computed Tomography and Sonogram Images
[3o] Detection of Covid-19 in Chest X-ray Image using CLAHE and Convolutional Neural Network
[31]Application of deep learning for fast detection of COVID-19 in X-Rays using nCOVnet
[32]CoroNet: A deep neural network for detection and diagnosis of COVID-19 from chest x-ray images
[33]Novel Coronavirus Identification from ChestX-Ray Images By Using VGG
[34]NIA-Network: Towards improving lung CT infection detection for COVID-19 diagnosis
[35]A new COVID-19 Patients Detection Strategy (CPDS) based on hybrid feature selection and enhanced KNN classifier
[36]A deep learning and grad-CAM based color visualization approach for fast detection of COVID-19 cases using chest X-ray and CT-Scan images
[37]DenseCapsNet: Detection of COVID-19 from X-ray images using a capsule neural network
[38]Accurate detection of COVID-19 using deep features based on X-Ray images and feature selection methods
[39]Automated detection of COVID-19 cases using deep neural networks with X-ray images
[4o]A deep learning algorithm using CT images to screen for Corona Virus Disease (COVID-19)
[41]Deep Transfer Learning Based Classification Model for COVID-19 Disease
[42]Transfer Learning with Deep Convolutional Neural Network (CNN) for Pneumonia Detection Using Chest X-ray

[43]COVIDX-Net: A Framework of Deep Learning Classifiers to Diagnose COVID-19 in X-Ray Images
[44]COVIDiagnosis-Net: Deep Bayes-SqueezeNet based diagnosis of the coronavirus disease 2o19 (COVID-19) from X-ray images
[45]Classification of the COVID-19 infected patients using DenseNet2o1 based deep transfer learning

CODE:

```
!pip install -q kaggle

from google.colab import files

files.upload()

!mkdir ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d bachrr/covid-chest-xray

!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia

!cd /content
!ls

!mkdir -p input/covid-chest-xray
!mkdir -p input/chest-xray-pneumonia

!rm -rf dataset
!mkdir -p dataset/covid
!mkdir -p dataset/normal

!unzip covid-chest-xray.zip -d input/covid-chest-xray

!unzip chest-xray-pneumonia.zip -d input/chest-xray-pneumonia

!pip install imutils

from keras import backend as K
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelBinarizer
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from imutils import paths
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import random
import shutil
import cv2
import os

import warnings
warnings.filterwarnings("ignore")

dataset_path='./dataset'

"""Covid X-ray"""
samples = 141
covid_dataset_path = '../content/input/covid-chest-xray'
csvPath = os.path.sep.join([covid_dataset_path, "metadata.csv"])
df = pd.read_csv(csvPath)
for (i, row) in df.iterrows():
  if row["finding"] != "COVID-19" or row["view"] != "PA":
    continue
  imagePath = os.path.sep.join([covid_dataset_path, "images", row["filename"]])
  if not os.path.exists(imagePath):
    continue
  filename = row["filename"].split(os.path.sep)[-1]
  outputPath = os.path.sep.join([f"{dataset_path}/covid", filename])
  shutil.copy2(imagePath, outputPath)

"""Normal X-ray dataset"""
pneumonia_dataset_path ='../content/input/chest-xray-pneumonia/chest_xray'
basePath = os.path.sep.join([pneumonia_dataset_path, "train", "NORMAL"])
imagePaths = list(paths.list_images(basePath))
print(len(imagePaths))
random.seed(42)
random.shuffle(imagePaths)
imagePaths = imagePaths[:samples]
for (i, imagePath) in enumerate(imagePaths):
  filename = imagePath.split(os.path.sep)[-1]
  outputPath = os.path.sep.join([f"{dataset_path}/normal", filename])
  shutil.copy2(imagePath, outputPath)

covid_dataset=pd.read_csv("../content/input/covid-chest-xray/metadata.csv")
covid_dataset.head()

# function to plot images in a grid
def ceildiv(a, b):
```

```
  return -(-a // b)

def plots_from_files(imspaths, figsize=(10,5), rows=1, titles=None, maintitle=None):
  """Plot the images in a grid"""
  f = plt.figure(figsize=figsize)
  if maintitle is not None: plt.suptitle(maintitle, fontsize=10)
  for i in range(len(imspaths)):
    sp = f.add_subplot(rows, ceildiv(len(imspaths), rows), i+1)
    sp.axis('off')
    if titles is not None: sp.set_title(titles[i], fontsize=16)
    img = plt.imread(imspaths[i])
    plt.imshow(img)

normal_images = list(paths.list_images(f"{dataset_path}/normal"))
covid_images = list(paths.list_images(f"{dataset_path}/covid"))

print("Normal cases :",len(normal_images))
print("Covid cases :",len(covid_images))

plots_from_files(normal_images, rows=5, maintitle="Normal X-ray images")
plots_from_files(covid_images, rows=5, maintitle="Covid-19 X-ray images")

#learning rate, epoch and batch size
INIT_LR = 1e-3
EPOCHS = 15
BS = 8

class_to_label_map = {'covid' : 1, 'normal' : 0}

imagePaths = list(paths.list_images(dataset_path))
data = []
labels = []
for imagePath in imagePaths:
  label = imagePath.split(os.path.sep)[-2]
  image = cv2.imread(imagePath)
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  image = cv2.resize(image, (224, 224))
  data.append(image)
  labels.append(class_to_label_map[label])
data = np.array(data) / 255.0
labels = np.array(labels)

labels.shape
labels = to_categorical(labels)
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.20,
stratify=labels, random_state=42)
trainAug = ImageDataGenerator(rotation_range=15, fill_mode="nearest")

"""VGG-16"""
baseModel = VGG16(weights="imagenet", include_top=False,
```

```
input_tensor=Input(shape=(224,224,3)))
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
vggmodel = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
vggmodel.summary()

#Using Adam optimizer
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
vggmodel.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy","AUC"])

# train the head of the network
print("[INFO] training head...")
H = vggmodel.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")

#matrices
predIdxs = vggmodel.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
precision = cm[0, 0] / (cm[0, 0] + cm[0, 1])
recall = cm[1, 1] / (cm[1, 0] + cm[1, 1])
print(cm)
print("acc: {:.4f}".format(acc))
```

```
print("precision: {:.4f}".format(precision))
print("recall: {:.4f}".format(recall))

target_names = ['Normal','Covid-19']
from skimage import data, color, io, img_as_float
def get_heatmap(vgg_conv, processed_image, class_idx):
  eps=1e-8
  gradModel = Model(
    inputs=[vgg_conv.inputs],
    outputs=[vgg_conv.get_layer('block5_conv3').output, vgg_conv.output])
  with tf.GradientTape() as tape:
    inputs = tf.cast(processed_image, tf.float32)
    (convoutputs, predictions) = gradModel(inputs)
    loss = predictions[:, class_idx]
  # use automatic differentiation to compute the gradients
  grads = tape.gradient(loss, convoutputs)
  # compute the guided gradients
  castConvoutputs = tf.cast(convoutputs > 0, "float32")
  castGrads = tf.cast(grads > 0, "float32")
  guidedGrads = castConvoutputs * castGrads * grads
  convoutputs = convoutputs[0]
  guidedGrads = guidedGrads[0]
  weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
  cam = tf.reduce_sum(tf.multiply(weights, convoutputs), axis=-1)
  (w, h) = (processed_image.shape[2], processed_image.shape[1])
  heatmap = cv2.resize(cam.numpy(), (w, h))
  numer = heatmap - np.min(heatmap)
  denom = (heatmap.max() - heatmap.min()) + eps
  heatmap = numer / denom
  heatmap = (heatmap * 255).astype("uint8")
  return heatmap

def print_GradCAM(base_model, path_image):
  # select the sample and read the corresponding image and label
  sample_image = cv2.imread(path_image)
  # pre-process the image
  sample_image = cv2.resize(sample_image, (224,224))
  if sample_image.shape[2] ==1:
    sample_image = np.dstack([sample_image, sample_image, sample_image])
  sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)
  sample_image = sample_image.astype(np.float32)/255.
  # sample_label = 1
  #since we pass only one image,we expand dim to include batch size 1
  sample_image_processed = np.expand_dims(sample_image, axis=0)
  # get the label predicted by our original model
  pred_label = np.argmax(base_model.predict(sample_image_processed), axis=-1)[0]
  # get the heatmap for class activation map(CAM)
  heatmap = get_heatmap(base_model, sample_image_processed, pred_label)
  heatmap = cv2.resize(heatmap, (sample_image.shape[0], sample_image.shape[1]))
  heatmap = heatmap *255
```

```
  heatmap = np.clip(heatmap, 0, 255).astype(np.uint8)
  heatmap = 255 - heatmap
  heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
  # f,ax = plt.subplots(1,2, figsize=(16,6))
  plt.figure()
  f, ax = plt.subplots(ncols=3, figsize=(16, 6))
  ax[1].imshow(heatmap)
  ax[1].set_title("heatmap")
  ax[1].axis('off')
  #superimpose the heatmap on the image
  sample_image_hsv = color.rgb2hsv(sample_image)
  heatmap = color.rgb2hsv(heatmap)
  alpha=0.7
  sample_image_hsv[..., 0] = heatmap[..., 0]
  sample_image_hsv[..., 1] = heatmap[..., 1] * alpha
  img_masked = color.hsv2rgb(sample_image_hsv)
  # f,ax = plt.subplots(1,2, figsize=(16,6))
  ax[0].imshow(sample_image)
  ax[0].set_title(f"original image (predicted label: {target_names[pred_label]})")
  ax[0].axis('off')
  ax[2].imshow(img_masked)
  ax[2].set_title("superimposed image")
  ax[2].axis('off')
  plt.show()

outputPath ='../content/input/covid-chest-xray/images/16664_1_1.jpg'
print_GradCAM(vggmodel, outputPath)

outputPath = '../content/input/covid-chest-xray/images/covid-19-rapidly-progressive-
acute-respiratory-distress-syndrome-ards-day-3.jpg'
print_GradCAM(vggmodel, outputPath)

covid_dataset_path='../content/input/covid-chest-xray'
csvPath = os.path.sep.join([covid_dataset_path, "metadata.csv"])
df = pd.read_csv(csvPath)
for (i, row) in df.iterrows():
  # if (1) the current case is not COVID-19 or (2) this is not
  # a 'PA' view, then ignore the row
  target_names = ['Covid-19','Normal']
  if row["finding"] == "COVID-19" or row["view"] != "PA" :
    continue
  imagePath = os.path.sep.join([covid_dataset_path, "images", row["filename"]])
  if not os.path.exists(imagePath):
    continue
  print_GradCAM(vggmodel, imagePath)

!pip install tf-explain

from tf_explain.core.activations import ExtractActivations
from tensorflow.keras.applications.densenet import DenseNet121
```

```
from tensorflow.keras.layers import BatchNormalization

baseModel = DenseNet121(input_shape = (224, 224, 3),include_top = False, weights =
None)
headModel = baseModel.output
headModel=BatchNormalization()(headModel)
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
dense_model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
print(dense_model.summary())

#Using Adam optimizer
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
dense_model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = dense_model.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")

#matrices
predIdxs = dense_model.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
```

```
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
print(cm)
print("acc: {:.4f}".format(acc))
print("precision: {:.4f}".format(sensitivity))
print("recall: {:.4f}".format(specificity))

from skimage import data, color, io, img_as_float
def get_heatmap(vgg_conv, processed_image, class_idx):
  eps=1e-8
  gradModel = Model(
    inputs=[vgg_conv.inputs],
    outputs=[vgg_conv.get_layer('conv5_block16_2_conv').output, vgg_conv.output])
  with tf.GradientTape() as tape:
    inputs = tf.cast(processed_image, tf.float32)
    (convoutputs, predictions) = gradModel(inputs)
    loss = predictions[:, class_idx]
  # use automatic differentiation to compute the gradients
  grads = tape.gradient(loss, convoutputs)
  # compute the guided gradients
  castConvoutputs = tf.cast(convoutputs > 0, "float32")
  castGrads = tf.cast(grads > 0, "float32")
  guidedGrads = castConvoutputs * castGrads * grads
  convoutputs = convoutputs[0]
  guidedGrads = guidedGrads[0]
  weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
  cam = tf.reduce_sum(tf.multiply(weights, convoutputs), axis=-1)
  (w, h) = (processed_image.shape[2], processed_image.shape[1])
  heatmap = cv2.resize(cam.numpy(), (w, h))
  numer = heatmap - np.min(heatmap)
  denom = (heatmap.max() - heatmap.min()) + eps
  heatmap = numer / denom
  heatmap = (heatmap * 255).astype("uint8")
  return heatmap

target_names = ['Normal','Covid-19']
outputPath = '../content/input/covid-chest-xray/images/16664_1_1.jpg'
print_GradCAM(dense_model, outputPath)

outputPath = '../content/input/covid-chest-xray/images/covid-19-rapidly-progressive-
acute-respiratory-distress-syndrome-ards-day-3.jpg'
print_GradCAM(dense_model, outputPath)

from tf_explain.core.activations import ExtractActivations
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import BatchNormalization

baseModel = ResNet50(input_shape = (224, 224, 3),
  include_top = False, weights = None)
headModel = baseModel.output
headModel=BatchNormalization()(headModel)
```

```
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
resnet = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
print(resnet.summary())

#Using Adam optimizer
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
resnet.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = resnet.fit_generator(
  trainAug.flow(trainX, trainY, batch_size=BS),
  steps_per_epoch=len(trainX) // BS,
  validation_data=(testX, testY),
  validation_steps=len(testX) // BS,
  epochs=EPOCHS)

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on COVID-19 Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")

#matrices
predIdxs = resnet.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)
cm = confusion_matrix(testY.argmax(axis=1), predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])
print(cm)
print("acc: {:.4f}".format(acc))
print("precision: {:.4f}".format(sensitivity))
print("recall: {:.4f}".format(specificity))
```

```
from skimage import data, color, io, img_as_float
def get_heatmap(vgg_conv, processed_image, class_idx):
  eps=1e-8
  gradModel = Model(
    inputs=[vgg_conv.inputs],
    outputs=[vgg_conv.get_layer('conv5_block3_3_conv').output, vgg_conv.output])
  with tf.GradientTape() as tape:
    inputs = tf.cast(processed_image, tf.float32)
    (convoutputs, predictions) = gradModel(inputs)
    loss = predictions[:, class_idx]
  # use automatic differentiation to compute the gradients
  grads = tape.gradient(loss, convoutputs)
  # compute the guided gradients
  castConvoutputs = tf.cast(convoutputs > 0, "float32")
  castGrads = tf.cast(grads > 0, "float32")
  guidedGrads = castConvoutputs * castGrads * grads
  convoutputs = convoutputs[0]
  guidedGrads = guidedGrads[0]
  weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
  cam = tf.reduce_sum(tf.multiply(weights, convoutputs), axis=-1)
  (w, h) = (processed_image.shape[2], processed_image.shape[1])
  heatmap = cv2.resize(cam.numpy(), (w, h))
  numer = heatmap - np.min(heatmap)
  denom = (heatmap.max() - heatmap.min()) + eps
  heatmap = numer / denom
  heatmap = (heatmap * 255).astype("uint8")
  return heatmap

outputPath = '../content/input/covid-chest-xray/images/16664_1_1.jpg'
print_GradCAM(resnet, outputPath)
outputPath = '../content/input/covid-chest-xray/images/covid-19-rapidly-progressive-
acute-respiratory-distress-syndrome-ards-day-3.jpg'
print_GradCAM(resnet, outputPath)
```