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

School of Information Technology and Engineering

DIGITAL IMAGE PROCESSING – ‘J’ Component – 1st Review

Title of the project: TUBERCULOSIS DETECTION WITH X-RAY IMAGES

Team Members:

1) GOPINI SAI BHUVAN – 20MIS0104
2) GOKUL R – 20MIS0332

TUBERCULOSIS DETECTION USING CHEST X-RAY IMAGES

Abstract:

Tuberculosis (TB) is a chronic lung disease that occurs due to bacterial infection and is one of the top 10 leading causes of death. It is transmitted by aerosol inhalation of the bacterium Mycobacterium tuberculosis (MTB) from an infected individual. During the course of infection, a wide variety of pulmonary disease lesion presentations may concurrently present within the same host. Accurate and early detection of TB is very important, otherwise, it could be life-threatening. The latest World Health Organization (WHO) study on 2018 is showing that about 1.5 million people died and around 10 million people are infected with tuberculosis (TB) each year. Moreover, more than 4,000 people die every day from TB.

A number of those deaths could have been stopped if the disease was identified sooner. In this work, we have detected TB reliably from the chest X-ray images using image pre-processing, data augmentation, image segmentation, and deep-learning classification techniques. We also used a visualization technique to confirm that CNN learns dominantly from the segmented lung regions that resulted in higher detection accuracy.

Literature Review:

[1] Incorporating DL technique with Unsharp Masking (UM) and High-Frequency Emphasis Filtering (HEF) image enhancement, this paper uses EfficientNet-B4, ResNet-50 and ResNet-18 in order to train the TB images and improve the detection accuracy. The experiments showed that the accuracy of the proposed idea is very competitive. Moreover, in terms of the AUC and accuracy, we also thoroughly compared the results with previous works, the proposed idea achieved better results. The use of an image enhancement system to preprocess the TB images will thus allow the tested pre-trained network to learn better model. Future works will evaluate more image enhancement techniques in order to show a more significant effect of enhancement on DL models

[2] This work presents a transfer learning approach with deep Convolutional Neural Networks for the automatic detection of tuberculosis from the chest radiographs. The performance of nine different CNN models were evaluated for the classification of TB and normal CXR images. ChexNet model outperforms other deep CNN models for the datasets without image segmentation whereas DenseNet201 outperforms for the segmented lungs. The classification accuracy, precision and recall for the detection of TB were found to be 96.47%, 96.62%, and 96.47% without segmentation and 98.6%, 98.57%, and 98.56% with segmentation respectively. It was also shown that image segmentation can significantly improve classification accuracy.

[3] As we all know, TB is a virulent infection disease, and several countries are suffering from a lack of resources, particularly developing countries. Therefore, every single positive case must be identified. The study introduced an approach to combined pretrained CNNs such as ResNet101, VGG19, and DenseNet201 with the XGBoost model to detect TB from CXR images.

[4] This work presents a workable solution for the detection of Tuberculosis from chest X-ray images. Starting from the observation that while existing approaches obtained an encouraging prediction performance, most of them have been evaluated on small and un-diverse datasets, we hypothesize that such a good performance might not hold for heterogeneous data sources, which originate from real world scenarios. Our model has been implemented based on two building blocks: deep convolutional neural networks with EfficientNet and Attention with Vision Transformer as the prediction engines, and effective transfer learning algorithms. One of the main advantages of EfficientNet is that the network family is compact as it is small in size and efficient, allowing us to incorporate various augmented techniques, e.g., Vision Transformer and Transfer Learning. An empirical evaluation on a considerably large dataset combined by using various datasets, which have been widely used in different papers, shows that our system obtains a better prediction performance compared to relevant studies. We conclude that the combination of EfficientNet with Vision Transformer and Learning brings in substantial improvement in performance compared to state- of the-art approaches.

[5] Diagnosis of Pulmonary infection through chest X-ray needs expertise. A diagnostic challenge to the physician is especially because diseases that mimic each other. The misdiagnosis may lead to inappropriate treatment which may risk the life of a patient. In this paper, we have proposed a novel framework to classify TB, Bacterial pneumonia and Viral pneumonia in chest X-ray in by using the Neural Network classifier. The previous works in this field have accuracy less than ours because they took the height and width of the image into consideration but the depth information was lost. And in our framework, we have taken images at different angles and shifting the images horizontally and vertically and rescaling the it

[6] A supervised deep learning model developed by using the training dataset from one population may not always have the same diagnostic performance in another population. Technical specification of CXR images, disease severity distribution, dataset distribution shift, and overdiagnosis should be examined before implementation in other settings.

[7] Computer aided diagnostic methods utilise radiographical data and machine learning algorithms for the early detection of several life-threatening diseases such as Tuberculosis, Pneumonia, COVID19 and Cardiovascular diseases. A robust automated system using non-invasive chest radiography that would be accessed by medical practitioners to detect subtle characteristics of pulmonary Tuberculosis is essential. The proposed scheme studies the effect of ELM and its variant in differentiating healthy and PTB patients in chest radiographs using integrated local texture descriptors. Both the classifiers with significant features are found to localize abnormalities by providing high classification sensitivity. The overall performance of ELM is found to be high. OSELM achieved the highest sensitivity in abnormality detection with minimal number of features.

[8] Efforts to develop effective and safe drugs for treatment of tuberculosis require preclinical evaluation in animal models. Alongside efcacy testing of novel therapies, efects on pulmonary pathology and disease progression are monitored by using histopathology images from these infected animals. To compare the severity of disease across treatment cohorts, pathologists have historically assigned a semi-quantitative histopathology score that may be subjective in terms of their training, experience, and personal bias. Manual histopathology therefore has limitations regarding reproducibility between studies and pathologists, potentially masking successful treatments.

[9] This article compares the improved CNN model with the traditional machine learning algorithm as SVM [8], Naive Bayes classifier [9], CART decision tree and KNN [10], compares and analyzes its accuracy in tuberculosis classification. Table III shows the comparison of test accuracy of different algorithms

[10] The mixture of Gaussians performed best in the first stage of classification. It showed the lowest ratio of incorrectly classified pixels, which translates into few outlier pixels classified as bacilli. It picked up most of the bacilli with their length in the focal plane of an image; the relatively low percentage of correctly classified pixels (75.74%) was mainly due to inaccuracies in detecting object outlines. The MOG classifier performed best in the second stage of classification, using all features. Among the different feature sets, eccentricity and compactness produced the highest accuracy for all classifiers (Table II); the addition of Fourier features and moments increased specificity and reduced sensitivity for the Gaussian and MOG classifiers, and reduced overall performance for the PCA and KNN classifiers. The PCA classifier performed poorly on the linear Fisher mapped test set because it requires variance of features, which is removed by Fisher mapping. Fisher mapping improved specificity but reduced sensitivity for the other classifiers.

[11] This work provides a proof of concept on how image processing techniques can be applied to automatically detect bacilli in microscopic images of sputum treated with the ZN stain. This staining procedure introduces several strange objects in the detection process as opposed to the Auramine staining process. Even with simple techniques for acquisition of images and classification of objects, the results are close to previously reported attempts.

[12] In this paper they propose a novel study for automatic diagnosis of TB based on image classification and plasmonic ELISA. This research study has two research contributions. First, it integrates a biosensing mechanism (i.e., plasmonic ELISA) with computational intelligence to detect TB. Second, it compares the classification performance of various types of classifiers. The results of applying the classifiers on the testing dataset (25% of the whole dataset) show high accuracy rate (>94%) despite blurriness in the images. The bagged tree method uses random forest classifier with decision tree learners. We have varied the number of learners (100 – 300 learners) in our simulations but observed no significant change in the predictive performance.

[13] The best-performing classifier had an AUC of 0.99, which was an ensemble of the AlexNet and GoogLeNet DCNNs. The AUCs of the pretrained models were greater than that of the untrained models (P, .001). Augmenting the dataset further increased accuracy (P values for AlexNet and GoogLeNet were .03 and .02, respectively). The DCNNs had disagreement in 13 of the 150 test cases, which were blindly reviewed by a cardiothoracic radiologist, who correctly interpreted all 13 cases (100%). This radiologist-augmented approach resulted in a sensitivity of 97.3% and specificity 100%.

[14] A scheme to segment and classify TB bacilli from ZN-stained images is presented. The bacilli are segmented by thresholding the hue component by choosing an appropriate range adaptively based on the input image. The beaded structure of the bacilli is obtained by segmenting the saturation

component. The presence of beaded structure and thresholds chosen for thread length, thread width and area parameters are used to identify valid single bacillus. Results presented for various images showed that the scheme performs well in spite of the variations in the images.

[15] We have presented a ConvNet model that uses VGG16 for classifying CXR images to identify patients suffering from TB. Previous research on CXR classification applied complex models for lung segmentation prior to training the model using Support Vector Machines. We show that VGG16 can use the raw data to classify the results with comparable accuracy without any form of pre-processing done in the previous research. To further increase the accuracy VGG16 was reapplied on a subset of data after performing augmentation to see if we could

achieve a higher accuracy. Results indicated that accuracy increases when VGG16 is applied on augmented images.

[16] This work presents an advanced neural network architecture optimized for tuberculosis diagnosis. We can train this specialized architecture from scratch and achieve good results compared to other publications, while reducing the computational, memory and power requirements significantly. We also analyzed the output with saliency maps and grad-CAMs and found that saliency maps offer a good visual explanation of the network decision. Saliency maps were interpreted by an expert radiologist (one of the authors, D.P.) and were found to highlight areas where tuberculosis was visible in many cases.

[17] The developed algorithm detects the TB bacilli automatically. This automated system reduces fatigue by providing images on the screen and avoiding visual inspection of microscopic images. The system has a high degree of accuracy, specificity and better speed in detecting TB bacilli. The method is simple and inexpensive for use in rural/remote areas in the emerging economies. Segmentation algorithm is developed to automate the process of detection of TB using digital microscopic images of different subjects.

[18] The algorithm recognized AFB under wide latitudes of staining, magnification and resolution (Figure 2). In Figure 2a,b nearly all visible bacilli were color-labeled as TB objects (green); conglomerations were labeled possible objects (blue). In Figure 2c,d the single typical TB bacillus was clearly recognized alongside a minor artifact. In Figure 2e,f, all AFB were recognized. In a challenge tissue slide (image not shown), the single TB bacillus was successfully detected without artifacts.

[19] The obtained results allow to conclude that the bacilli segmentation in the digital image by using the proposed methodology has up to 92% effectiveness, under different color and contrast image conditions. For normalized images, the method provides up to 98% effectiveness. The bacilli detection can be performed based on these segmentation results, helping to identify the bacilli by shape and size. In order to increase the robustness of the system, it is necessary to perform preprocessing tasks to eliminate such variability by standardizing the RGB components of the image. In addition, it is necessary to consider the image resolution in order to obtain adequate segmentation results.

[20] An automatic detection of tuberculosis for lung images is presented in this paper. The location of tuberculosis within the lung varies with the stage of infection and age of patient. The X-ray images contain variable lung shapes, a static model is not sufficient to describe the lung regions. In our method, linearly align all training masks to a given input CXRs by using rigid registration. The average mask computed on a subset of most similar training masks is used an approximate lung model for the input CXR. An approximate model is segmented using the watershed segmentation. Moreover, to improve the accuracy of the segmentation results noise and contrast is improved by using wiener filter and histogram equalization. The proposed method is evaluated by JSRT and MC dataset. We compare the global thresholding and active contour method of image segmentation with proposed algorithm and found that the accuracy of the proposed method is 60% compared with active contour and global thresholding

S.NO	PAPERS	DATASET	IMAGE ENHANCEMENT	IMAGE RESTORATION	IMAGE SEGMENTATION	FEATURE EXTRACTION	CLASIFIERS	QUALITY METRICES & RESULTS
1	IMAGE ENHANCEMENT FOR TUBERCULOSIS DETECTION USING DEEP LEARNING	Shenzhen Public dataset	Unsharp Masking,Contrast Limited Adaptive Histogram Equalization	High-Frequency Emphasis Filter	--------	ROI extraction	SVM classifier	accuracy = 89.92%, AUC(area under curve) = 94.8%
2	Reliable Tuberculosis Detection Using Chest X-Ray With Deep Learning, Segmentation and Visualization	Montgomery and Shenzhen datasets,kaggle lung x-ray & masks dataset(No.704 CXR)	-------	1 × 1 and 3 × 3 convolution filters	Score-CAM technique ,t-SNE technique (performed atlast using python)	ResNet18, ResNet50, ResNet101, DenseNet201, ChexNet, SqueezeNet, InceptionV3 , VGG19 and MobileNetV2	computer aided classifier	accuracy = 98.6%, precision = 98.57%,sensitivity = 98.56%, F1-score = 98.56%,specificity = 98.54%
3	Deep pre-trained networks as a feature extractor with XGBoost to detect tuberculosis from chest X-ray	NLM dataset,Belarus , dataset RSNA dataset(normal:10000, affected:20000,total:30000)	Contrast Limited Adaptive Histogram Equalization, Unsharp Masking and High-Frequency Emphasis Filtering	Gabor Filter	prostate segmentation	ResNet101-XGBoost, VGG19-XGBoost and DenseNet201-XGBoost	SVM-based ,XGBoost classifier	AUC 99.93 ± 0.13%, accuracy 99.92 ± 0.14%, precision 99.85 ± 0.20%, sensitivity 100 ± 0.1%, F1-score 99.92 ± 0.14% and specificity 99.85 ± 0.20%

4	Detection of tuberculosis from chest X-ray images : Boosting the performance with vision transformer and transfer learning	ImageNet dataset, Montgomery County (MC) CXR dataset, Shenzhen dataset, Belarus dataset	EfficientNet-B0,EfficientNet-B1	--------	[panoptic segmentation]	DenseNet, VGG16,Hybrid VCG	SVM, CNN AlexaNet, GoogLeNet	accuracy = 97.72%, AUC = 99.99%, precision = 97.43%
5	An efficient frame work for identification of Tuberculosis in chest X-ray images using Neural Network	Shenzhen chest X-ray set(336 affected,326 normal)	-------	convolution layer 3 × 3, 32 and fourth convolution layer of 3 × 3, 64	registration-based segmentation methods	Max-pooling	minimum distance classifier	accuracy = 99.01%
6	Deep learning for automated classification of tuberculosis-	National Library of Medicine Shenzhen No.3 Hospital,National Institute of Health Clinical Center	Tensorflow framework, Inception V3	--------	rotational methods	VGGNet or ResNet	---------	AUC = 98.45%, sensitivity = 72% , specificity = 82%

	related chest X-Ray: dataset distribution shift limits diagnostic performance generalizabilit							
7	Extreme Learning Machine based Differentiation of Pulmonary Tuberculosis in Chest Radiographs using Integrated Local Feature Descriptors	Montgomery County (MC) public dataset	-------	Median filter responses	RDLS segmented masks,Reaction Diffusion Level Set method	Local Histogram-based Descriptors	---------	accuracy and sensitivity > 98% , highiest sensitivity is observed with OSELM
8	Digital Image Analysis of Heterogeneous Tuberculosis Pulmonary Pathology in NonClinical Animal Models using	obtained from two diferent research laboratories at CSU. Mtb (samples)	Unsharp Masking technique	--------	pre-trained neural networks, histogram of oriented gradients (HOG)	pathology features--collagen rim and a caseous necrotic core	Histopathology classifcations	accuracy = 96.89%,sencitivity = 95.96%

	Deep Convolutional Neural Networks							
9	AE-CNN Classification of Pulmonary Tuberculosis Based on CT images	laboratory cooperating hospital	-------	--------	CT image segmentation technology	Conv and the unsupervised features of AutoEncoder	SVM, Naive Bayes classifier, CART decision tree, KNN	accuracy = 80.29%,recall = 80.67%,F1 =80.42%
10	Detection of tuberculosis in sputum smear images using two one -class classifier	kaggle	-------	Moment invariants, and eccentricity and compactness	colour-based Bayesian segmentation	geometric transformation invariant features	pixel classifier, one-class object classifier	Sensitivity of 97.89% and specificity of 94.67%
11	Automated Tuberculosis Screening Using image Processing Tools	Hospital Nacional Dos de Mayo	filtered with heuristics including size, eccentricity and color	Canny edge detection applied into the Q layer	--------	Fukunaga’s criterion	Mahalanobis distance was implimented	sensitivity = 60%, specificity = 92%
12	Automatic Diagnosis of Tuberculosis Disease Based on Plasmonic ELISA and	UK National Health Service	five-fold cross validation	noise filter. Pixels were thresholded based on L*	K-Means Clustering	color based feature extraction (color histogram features)	decision trees, support vector machines (SVMs), kNearest Neighbors algorithm (k-NN) classifiers and	accuracy = 97.2%, sensitivity = 97.1 %,specificity = 97.2

	Color-based Image Classification						ensemble classifiers	
13	Deep Learning at Chest Radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks	HIPAA-compliant datasets	Histogram equilization	--------	--------	texture and shape feature extraction	AlexNet and GoogLeNet	AUC = 99%,sencitivity = 97.3%,specificity = 100%
14	Segmentation and Classification of Tuberculosis Bacilli from ZN-stained Sputum Smear Images	kaggle	-------	color filtering method	fuzzy segmentation, phase-only correlation	chromatic channel thresholding	autofocus algorithm and a k-means clustering	accuracy = 94.67%, specificity = 94.34%
15	Application of a Convolutional Neural Network using transfer learning for tuberculosis	Human Services of Montgomery County (MC), Maryland, USA, Shenzhen No.3 Hospital	-------	median filter	VGG16	-------	ConvNet, AlexNet	accuracy = 92.63%,Sencitivity = 94%

	detection.							
16	Efficient Deep Network Architectures for Fast Chest X-Ray Tuberculosis Screening and Visualization	Maryland and Shenzhen dataset	Histogram equilization	--------	SIFT segmentation	shape descriptor histograms and using a simple neural network	SVM classifier	accuracy = 965.6%, AUC = 99%
17	Detection of Tuberculosis Bacilli using Image Processing Techniques	NLm dataset	-------	local thresholding and a median filter	Otsu thresholding and k-means clustering	texture and shape feature extraction	---------	accuracy = 98.91% , sensitivity = 99.22%, specificity = 98.73%
18	Image processing techniques for identifying Mycobacterium tuberculosis in Ziehl-Neelsen stains	Public Health Image Library	Histogram equilization	--------	Automated, multi-stage, color-based Bayesian segmentation	Shape extraction	pixel classifier, one-class object classifier	AUC=98.99%,sencitivity = 98.65%
19	Image processing for AFB segmentation in bacilloscopies of	https://figshare.com/s/9e3960a5e9684f7e	-------	adaptive filtering	K-means algorithm	MATLAB program (technique not mentioned)	Bayes classifier with Gaussian mixture	accuracy = 98.6602%, 93.3% sensitivity and 87% specificity, 93.3% sensitivity and 87% specificity.

	pulmonary tuberculosis diagnosis							
20	Automatic detection of Pulmonary tuberculosis using image processing techniques	Montgomery country (MC) and Japanese society of radiology(JSRT) dataset	contrast enhancement	Wiener filter	Watershed segmentation	-------	---------	accuracy = 92.78%,specificity = 92.11%,AUC = 97.94%
21	Tuberculosis Detection In Chest X-Ray Images Using Optimized Gray Level Co-Occurrence Matrix Features	Two public chest x-ray datasets for computer aided screening for pulmonary diseases.			Region Of Interest (ROI) segmentation	Twelve GLCM features from the image extraction process are optimized using PCA.	SVM classifier.	Accuracy = 98.72% for PTB & STB
22	Tuberculosis detection based on chest X-Ray using Ensemble method with CNN feature extraction	Kaggle Tuberculosis chest x-ray database	Combination of Convolution Neural Network (CNN) feature.				Random Forest (RF) and Extreme Gradient Boosting (XGBoost).	Accuracy= 98.67%, 98.993% (using CNN RF) & 98.367% and 99.886% (using CNN XGBoost)

23	Comparitive study for Tuberculosis Detection using Deep learning	Montgomery Country (MC) CXR dataset.				VGG16, VGG19, DenseNet121, MobileNet and InceptionV3	SVM classifiers.	Accuracy= 98.9% and area under curve (AUC) is 1.00
24	Deep Neural Network for Foreign Object Detection in chest X-ray	Region-based Convolution neural network			Without Lung Segmentation, With Lung Segmentation		Support Vector Machine (SVM) classifiers.	Accuracy = 97% precision, 90% recall, 93% F1-score.
25	Automatic detection of tuberculosis related abnormalities in Chest X-ray images using hierarchical feature extraction scheme	Montgomery dataset and Shenzhen dataset			Atlas-based segmentation	ROI, Hierarchical feature extraction	SVM classifiers	accuracy = 95.60 ± 5.07% and area under curve (AUC) = 0.95 ± 0.06 for Montgomery collection, and accuracy = 99.40 ± 1.05% and AUC = 0.99 ± 0.01 for Shenzhen collection

26	Detection of respiratory diseases from chest X rays using Nesterov accelerated adaptive moment estimation Au	He, Xuehai, et al dataset, ImageNet dataset, https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia	stochastic gradient descent algorithm			Convolutional Neural Network (CNN) feature. VGG 16 model, Xception		Accuracy = 97% (for normal models), VGG 16 model = 90.54% & Xception = 87.69%
27	An efficient mixture of deep and machine learning models for COVID-19 and Tuberculosis detection using x-ray images in resource limited settings	COVID-19 dataset, Normal dataset, Pneumonia-bacterial dataset, Pneumonia-viral dataset, tuberculosis dataset			Segmentation in chest radiographs using anatomical atlases with nongrid registration.	CNN such as VGG-19, DenseNet-201, MobileNet-v2, ResNet-50.	DF extraction feature with traditional machine learning classifiers	Accuracy= 91.6± 2.6% (accuracy ± Confidence Interval (CI) at 95% confidence Level

28.	Early Detection of Tuberculosis using Chest X-Ray (CXR) with Computer-Aided Diagnosis		Median filter, Hormonic filter and Contrast Limited Adaptive Histogram Equalization (CLAHE).		Active Shape model, Active Contour Model	Matlab consisting of image’s mean, variance, skewness, kurtosis and entropy	SVM classifiers	Accuracy = 76%, sensitivity of the system = 66.67% and specificity = 86%
29.	A Hybridized Pre-Processing Method for Detecting Tuberculosis using Deep Learning	MC dataset, Shenzhen dataset,			-----------	Computer aided detection	Automatic classifiers (Artificial intelligence technique)	Accuracy = 84% (for training set) and 82.6% (for test set)
30	Automated Tuberculosis Detection Using Pre-Trained CNN and SVM	Shenzhen dataset, Montgomery, Korean Institute of Tuberculosis (KIT)				VGG16, MobileNet	Vector machine classifiers	Accuracy = 96.9% and Area under curve = 0.99
31	Chest X-Ray Patch Classification for Tuberculosis Detection	Picture Archive and Communication Systems (PACS)				Gray Level Co-occurrence Matrix (GLCM) Feature	SVM classifiers	Accuracy = 91.2%, sensitivity = 97.1% And specificity = 87.2%

DATASET:

Our data set contains 3500 normal and 700 affected (Tuberculosis) images. They are saved in two different folders namely “Nor mal” and “Tuberculosis

Link to our dataset:https://www.kaggle.com/tawsifurrahman/tuberculosis-tb-chest-xray-dataset

REFFERENCES:

Munadi, K., Muchtar, K., Maulina, N., & Pradhan, B. (2020). Image Enhancement for Tuberculosis Detection Using Deep Learning. *IEEE Access*, *8*, 217897-217907 .http://ieeexplore.ieee.org.egateway.vit.ac.in/document/9277528

Rahman, T., Khandakar, A., Kadir, M. A., Islam, K. R., Islam, K. F., Mazhar, R., ... & Chowdhury, M. E. (2020). Reliable tuberculosis detection using chest X-ray with deep learning, segmentation and visualization. *IEEE Access*, *8*, 191586-191601.
http://ieeexplore.ieee.org.egateway.vit.ac.in/document/9224622

Rahman, M., Cao, Y., Sun, X., Li, B., & Hao, Y. (2021). Deep pre-trained networks as a feature extractor with XGBoost to detect tuberculosis from chest X-ray. *Computers & Electrical Engineering*, *93*, 107252.
http://www.sciencedirect.com.egateway.vit.ac.in/science/article/pii/S004579062100238X

Duong, L. T., Le, N. H., Tran, T. B., Ngo, V. M., & Nguyen, P. T. (2021). Detection of tuberculosis from chest X-ray images: Boosting the performance with vision transformer and transfer learning. *Expert Systems with Applications*, *184*, 115519.http://www.sciencedirect.com.egateway.vit.ac.in/science/article/pii/S0957417421009295

Verma, D., Bose, C., Tufchi, N., Pant, K., Tripathi, V., & Thapliyal, A. (2020). An efficient framework for identification of Tuberculosis and Pneumonia in chest X-ray images using Neural

Network. *Procedia Computer Science*, *171*, 217-224.
http://www.sciencedirect.com.egateway.vit.ac.in/science/article/pii/S1877050920309881

Sathitratanacheewin, S., Sunanta, P., & Pongpirul, K. (2020). Deep learning for automated classification of tuberculosis-related chest X-Ray: dataset distribution shift limits diagnostic performance generalizability. *Heliyon*, *6*(8), e04614.http://www.sciencedirect.com.egateway.vit.ac.in/science/article/pii/S2405844020314584

Govindarajan, S., & Swaminathan, R. (2021). Extreme Learning Machine based Differentiation of Pulmonary Tuberculosis in Chest Radiographs using Integrated Local Feature Descriptors. *Computer Methods and Programs in Biomedicine*, *204*, 106058.
http://www.sciencedirect.com.egateway.vit.ac.in/science/article/pii/S0169260721001334

Asay, B. C., Edwards, B. B., Andrews, J., Ramey, M. E., Richard, J. D., Podell, B. K., ... & Lenaerts, A. J. (2020). Digital image analysis of heterogeneous tuberculosis pulmonary pathology in non-clinical animal models using deep convolutional neural networks. *Scientific reports*, *10*(1), 1-14.https://www.nature.com.egateway.vit.ac.in/articles/s41598-020-62960-6

Li, L., Huang, H., & Jin, X. (2018, October). AE-CNN classification of pulmonary tuberculosis based on CT images. In *2018 9th International Conference on Information Technology in Medicine and Education (ITME)* (pp. 39-42). IEEE.
http://ieeexplore.ieee.org.egateway.vit.ac.in/document/8589252

Khutlang, R., Krishnan, S., Whitelaw, A., & Douglas, T. S. (2009, June). Detection of tuberculosis in sputum smear images using two one-class classifiers. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro* (pp. 1007-1010). IEEE.
http://ieeexplore.ieee.org.egateway.vit.ac.in/document/5193225

Castaneda, B., Aguilar, N. G., Ticona, J., Kanashiro, D., Lavarello, R., & Huaroto, L. (2010, March). Automated Tuberculosis screening using image processing tools. In *2010 Pan American Health Care Exchanges* (pp. 111-111). IEEE.
http://ieeexplore.ieee.org.egateway.vit.ac.in/stamp/stamp.jsp?tp=&arnumber=5474590

AbuHassan, K. J., Bakhori, N. M., Kusnin, N., Azmi, U. Z., Tania, M. H., Evans, B. A., ... & Hossain, M. A. (2017, July). Automatic diagnosis of tuberculosis disease based on Plasmonic ELISA and color-based image classification. In *2017 39th annual international conference of the IEEE engineering in medicine and biology society (EMBC)* (pp. 4512-4515). IEEE.
https://ieeexplore.ieee.org/abstract/document/8037859

Makkapati, V., Agrawal, R., & Acharya, R. (2009, August). Segmentation and classification of tuberculosis bacilli from ZN-stained sputum smear images. In *2009 IEEE International Conference on Automation Science and Engineering* (pp. 217-220). IEEE..
https://pubs.rsna.org/doi/full/10.1148/radiol.2017162326

Ahsan, M., Gomes, R., & Denton, A. (2019, May). Application of a Convolutional Neural Network using transfer learning for tuberculosis detection. In *2019 IEEE International Conference on Electro Information Technology (EIT)* (pp. 427-433). IEEE.
http://ieeexplore.ieee.org.egateway.vit.ac.in/document/5234173

Rachna, H. B., & Swamy, M. M. (2013). Detection of Tuberculosis bacilli using image processing techniques. *International Journal of Soft Computing and Engineering (IJSCE)*, *3*(4).
http://ieeexplore.ieee.org.egateway.vit.ac.in/document/8833768

Sadaphal, P., Rao, J., Comstock, G. W., & Beg, M. F. (2008). Image processing techniques for identifying Mycobacterium tuberculosis in Ziehl-Neelsen stains. *The International Journal of Tuberculosis and Lung Disease*, *12*(5), 579-582.

https://www.nature.com/articles/s41598-019-42557-4

Díaz-Huerta, J. L., Téllez-Anguiano, A. D. C., Fraga-Aguilar, M., Gutiérrez-Gnecchi, J. A., & Arellano-Calderón, S. (2019). Image processing for AFB segmentation in bacilloscopies of pulmonary tuberculosis diagnosis. *Plos one*, *14*(7), e0218861.

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.412.8178&rep=rep1&type=pdf

Poornimadevi, C. S., & Sulochana, H. (2016, March). Automatic detection of pulmonary tuberculosis using image processing techniques. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)* (pp. 798-802). IEEE.

https://www.ingentaconnect.com/content/iuatld/ijtld/2008/00000012/00000005/art00018

Jorge Luis Dı´az-Huerta1, Adriana del Carmen Te´llez-AnguianoID1*, Miguelangel FragaAguilar1, Jose´ Antonio Gutie´rrez-Gnecchi1, Sergio Arellano-Caldero´n2 “Image processing for AFB segmentation in bacilloscopies of pulmonary tuberculosis diagnosis”

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0218861

Poomimadevi.CS, Helen Sulochana. C “AUTOMATIC DETECTION OF PULMONARY TUBERCULOSIS USING IMAGE PROCESSING TECHNIQUES”

http://ieeexplore.ieee.org.egateway.vit.ac.in/document/7566243/——

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

School of Information Technology and Engineering

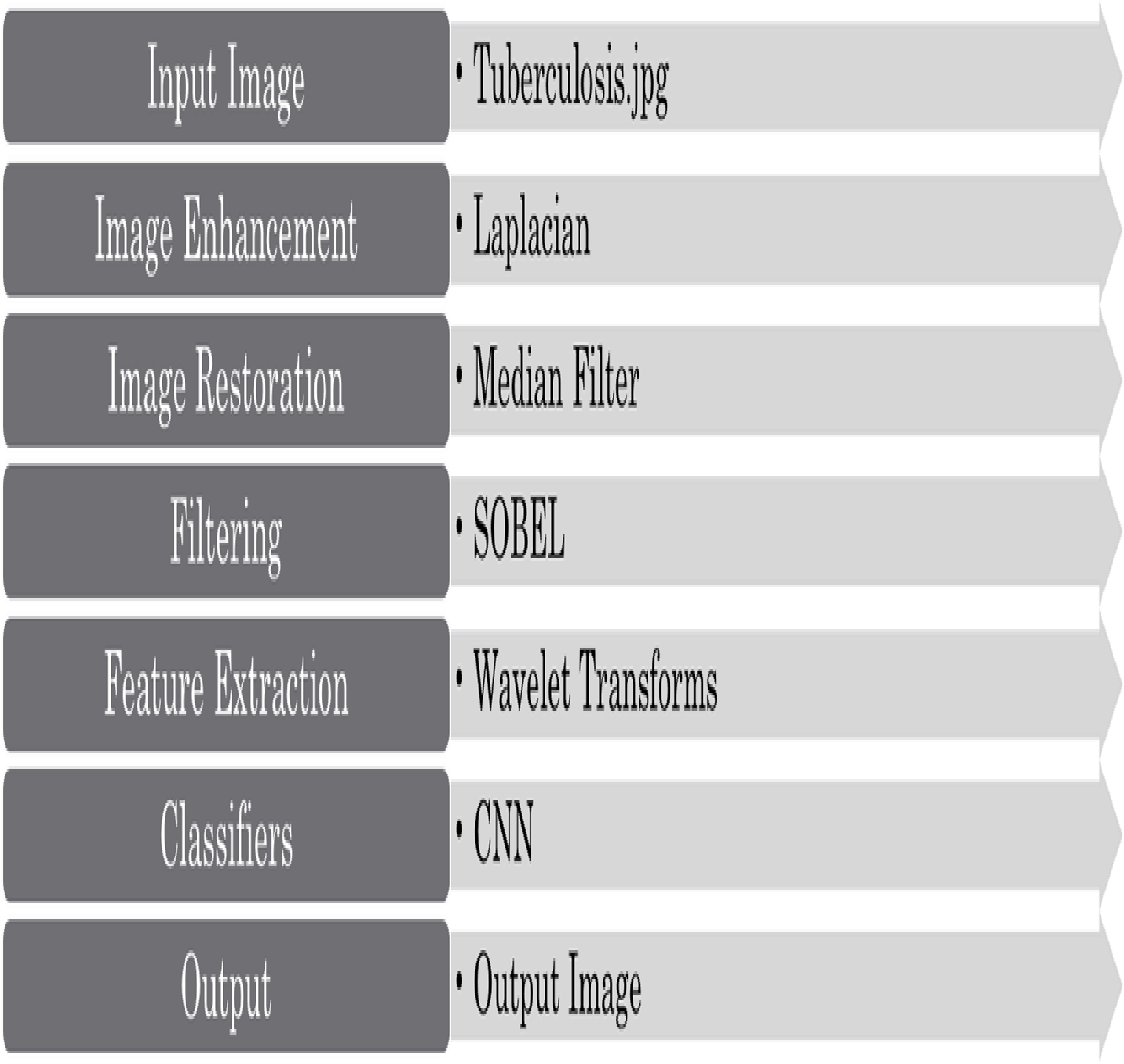
DIGITAL IMAGE PROCESSING – ‘J’ Component – 2nd Review

Title of the project: TUBERCULOSIS DETECTION WITH X-RAY IMAGES

Team Members:

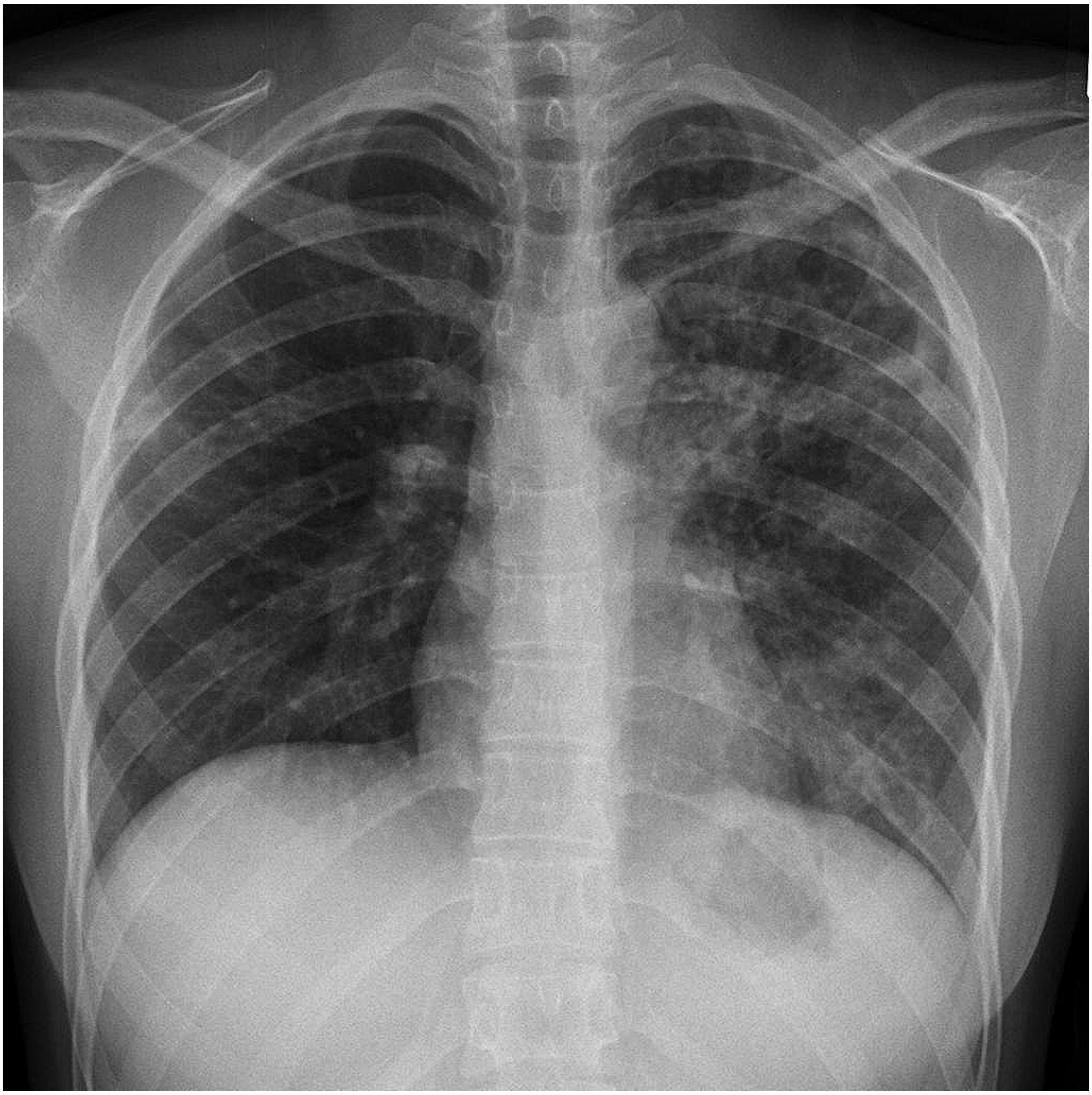
1) GOPINI SAI BHUVAN – 20MIS0104
2) GOKUL R – 20MIS0332

STEPS INVOLVED IN THIS REVIEW:



Input Image	• Tuberculosis.jpg
Image Enhancement	• Laplacian
Image Restoration	• Median Filter
Filtering	• SOBEL
Feature Extraction	• Wavelet Transforms
Classifiers	• CNN
Output	• Output Image

INPUT IMAGE:

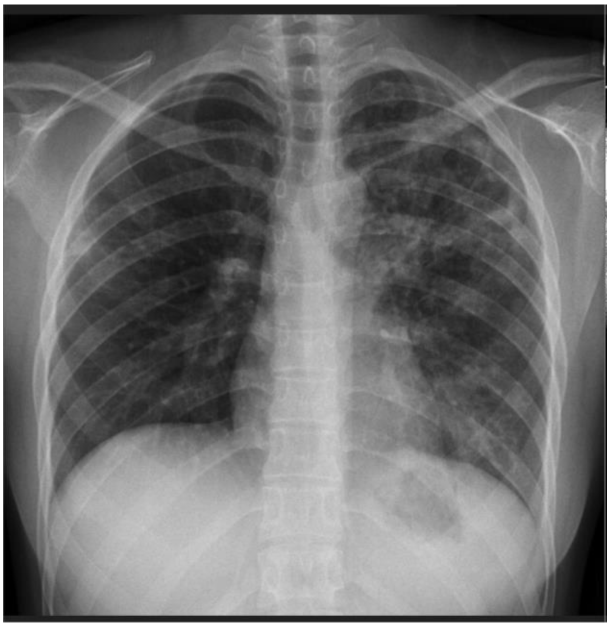


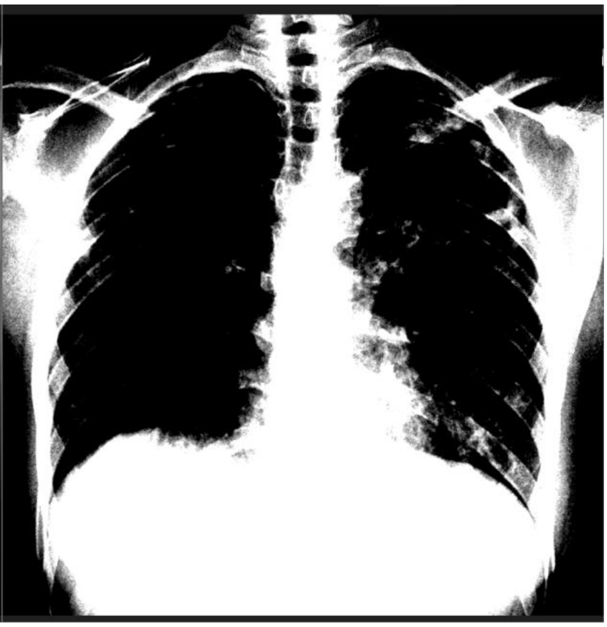
LANGUAGE USED:

PYTHON, R Programming.

IMAGE ENHANCEMENT:

```
from PIL import Image
from PIL import ImageEnhance

#To open the image
image =
Image.open("C:\\Users\\gokul\\PycharmProjects\DIP\\tuberculosis.jpg")

#To show the image
image.show()

#Enhance sharpness
curvedImage = ImageEnhance.Contrast(image)
NewSharp = 8.3

#Sharpness enhanced by a factor of 8.3
SharpedImage = curvedImage.enhance(NewSharp)

SharpedImage.show()
```



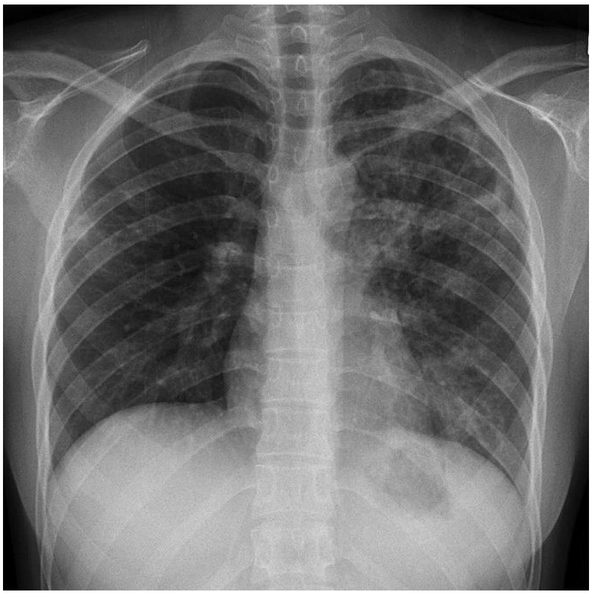


ORIGINAL IMAGE

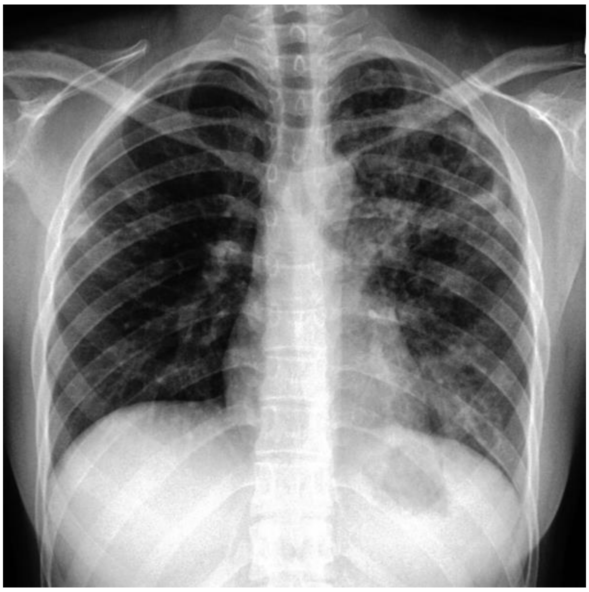
ENHANCED IMAGE

HISTOGRAM EQUALIZATION:

```
import numpy as np
from PIL import Image

image_filename =
"C:\\Users\\gokul\\PycharmProjects\DIP\\tuberculosis.jpg"
save_filename = 'output_image.jpg'

img = Image.open(image_filename)

#convert to grayscale
imagray = img.convert(mode = 'L')

#convert to numpy array
img_array = np.asarray(imagray)
"""
STEP 1: Normalized Cumulative histogram
"""

#flatten image array and calculate histogram via binning
histogram_array = np.bincount(img_array.flatten(),
minlength=256)

#normalize
num_pixels = np.sum(histogram_array)
histogram_array = histogram_array/num_pixels

#normalized cumulative histogram

chistogram_array = np.cumsum(histogram_array)

"""
STEP 2: Pixel mapping lookup table
"""
transform_map = np.floor(255 *
chistogram_array).astype(np.uint8)
"""
STEP 3: Transformation
"""
# flatten image array into 1D list
img_list = list(img_array.flatten())

# transform pixel values to equalize
eq_img_list = [transform_map[p] for p in img_list]

# reshape and write back into img_array
eq_img_array = np.reshape(np.asarray(eq_img_list),
img_array.shape)
eq_img = Image.fromarray(eq_img_array, mode='L')
eq_img.save(save_filename)
```



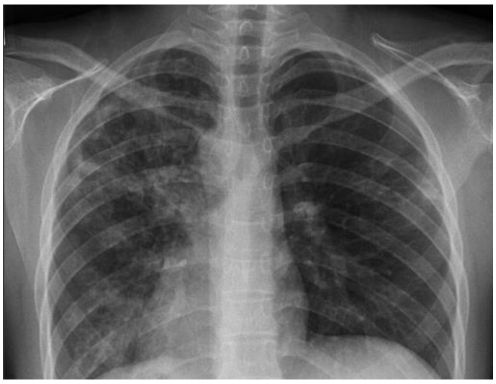
ORIGINAL



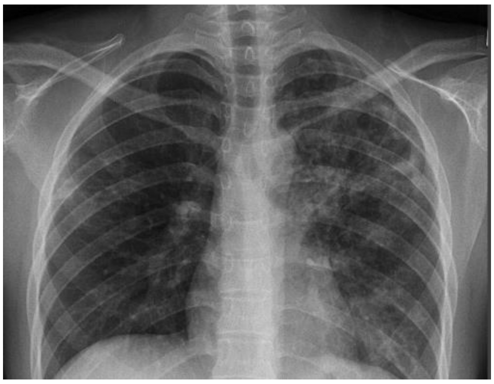
HISTOGRAMISED IMAGE

IMAGE RESTORATION:

```
import numpy as np
import cv2
img =
cv2.imread("C:\\Users\\gokul\\PycharmProjects\DIP\\tuberculosi
s.jpg")
grayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# edge_kernel = np.array([[-1,-1,-1], [-1,9,-1], [-1,- 1,-1]])

sharpen_kernel = np.array([[0,-1,0], [-1,5,-1], [0,-1,0]])
img = cv2.filter2D(grayscale, -1, sharpen_kernel)

# Smooth out image
# blur = cv2.medianBlur(img, 3)
blur = cv2.GaussianBlur(img, (3,3), 0)

cv2.imshow('img',img)
cv2.imwrite('img.png',img)
cv2.imshow('blur',blur)
cv2.waitKey(0)
```

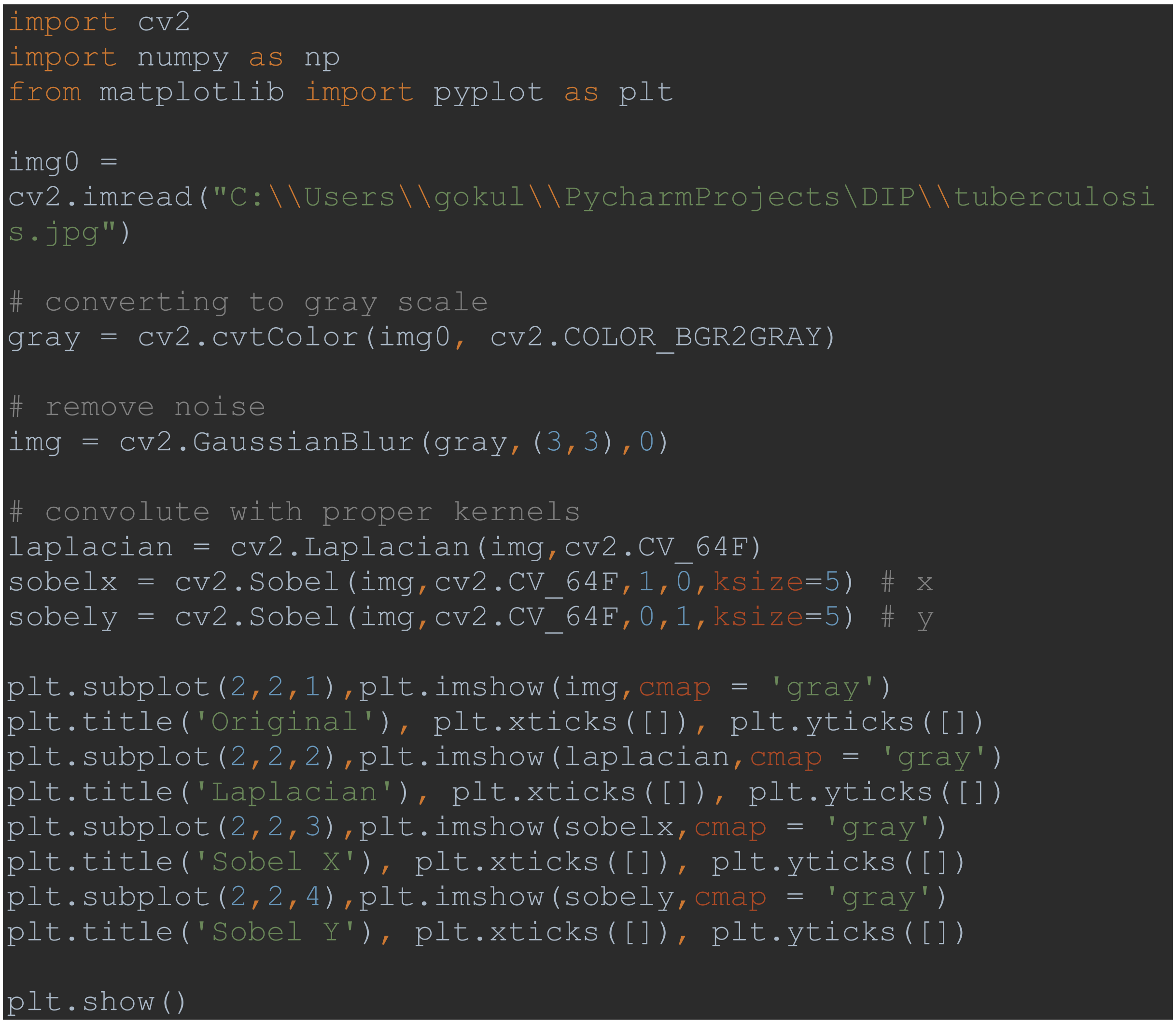


ORIGINAL IMAGE



RESTORED IMAGE

IMAGE FILTERATION:



```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img0 =
cv2.imread("C:\\Users\\gokul\\PycharmProjects\DIP\\tuberculosi
s.jpg")

# converting to gray scale
gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)

# remove noise
img = cv2.GaussianBlur(gray,(3,3),0)

# convolute with proper kernels
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

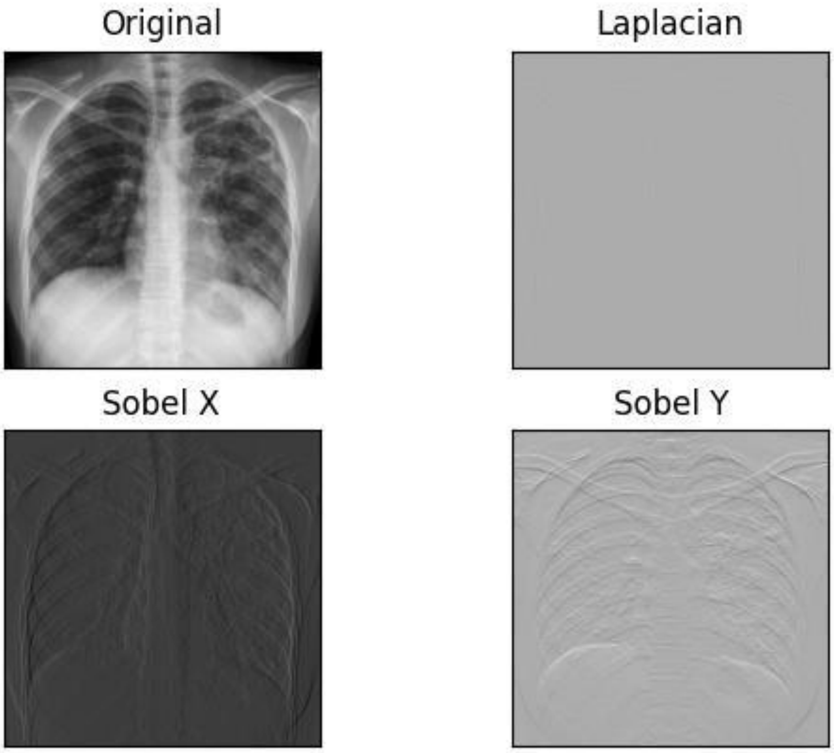
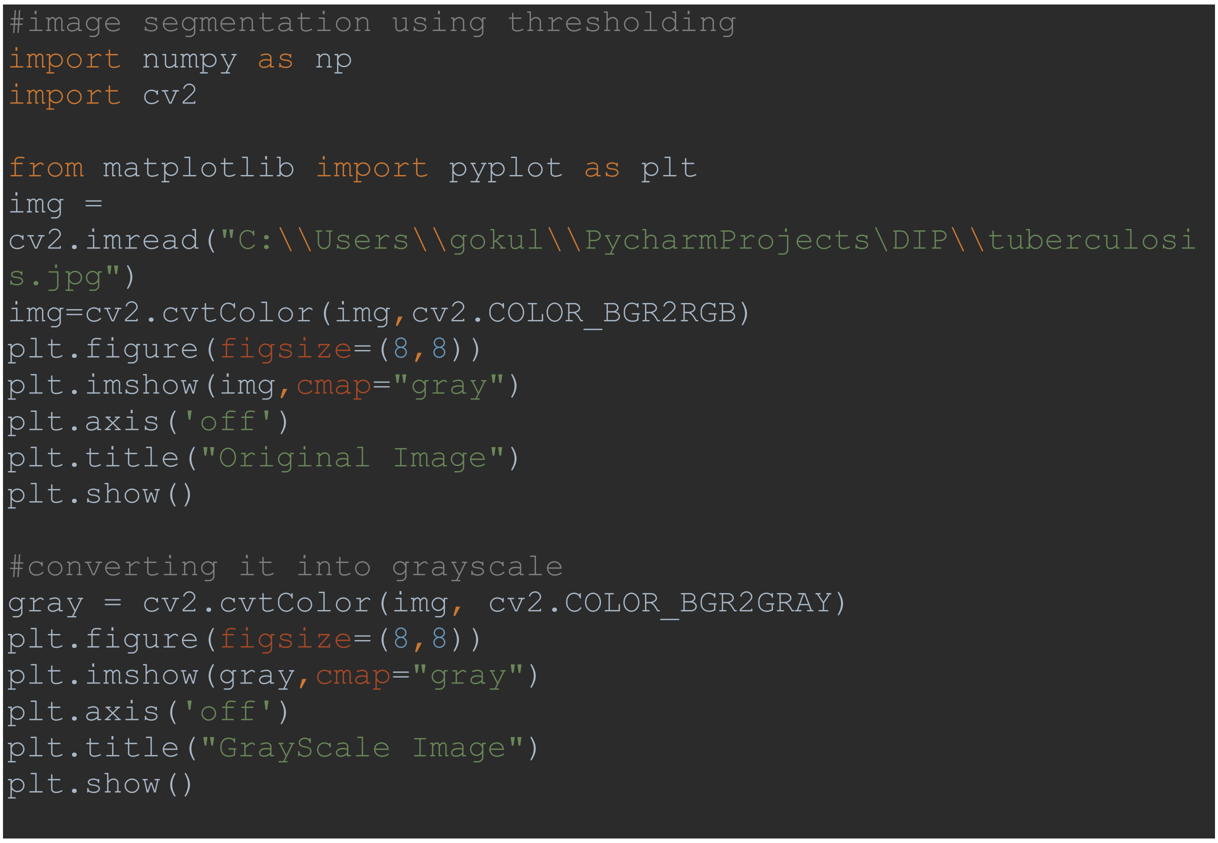
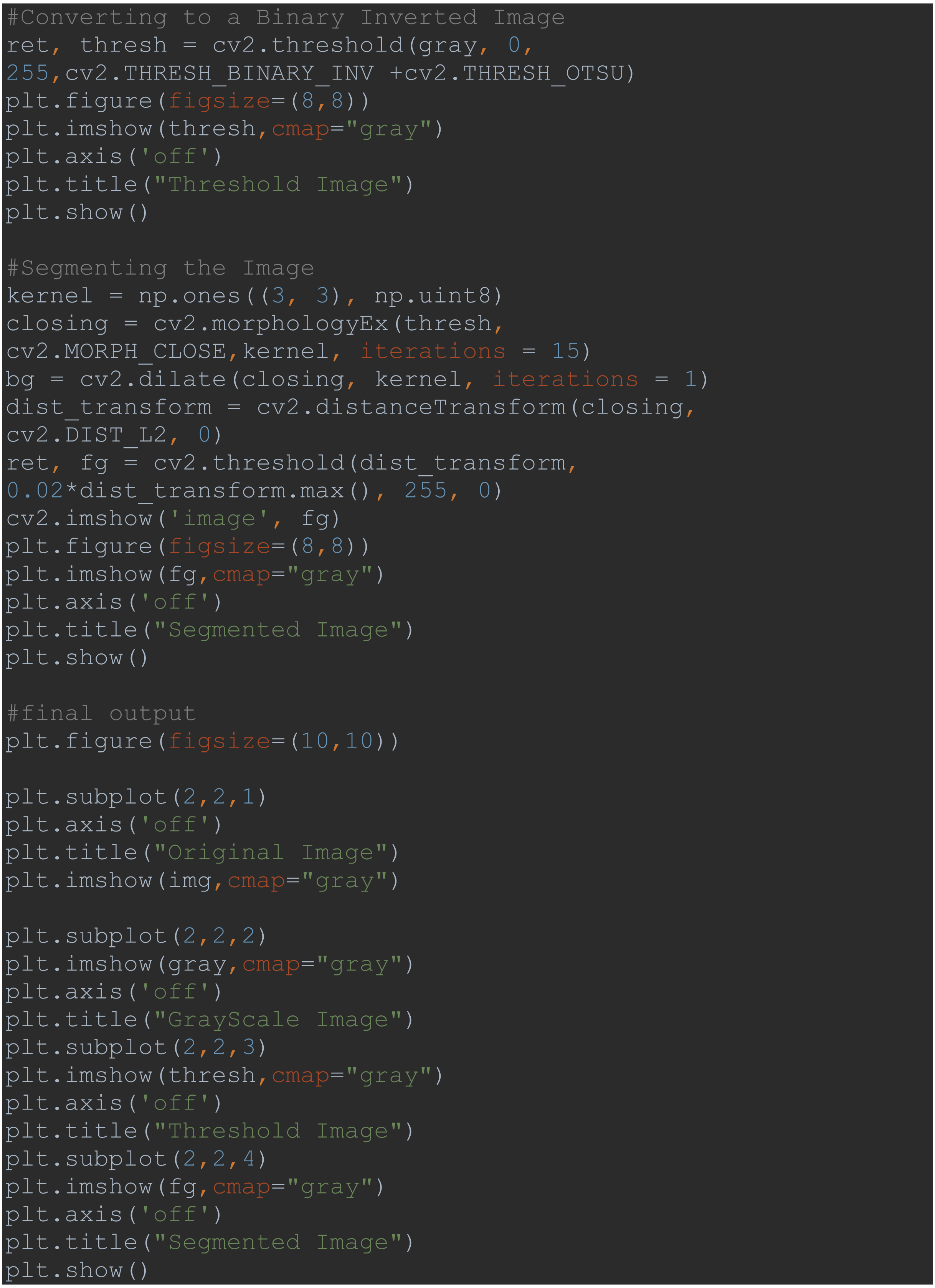


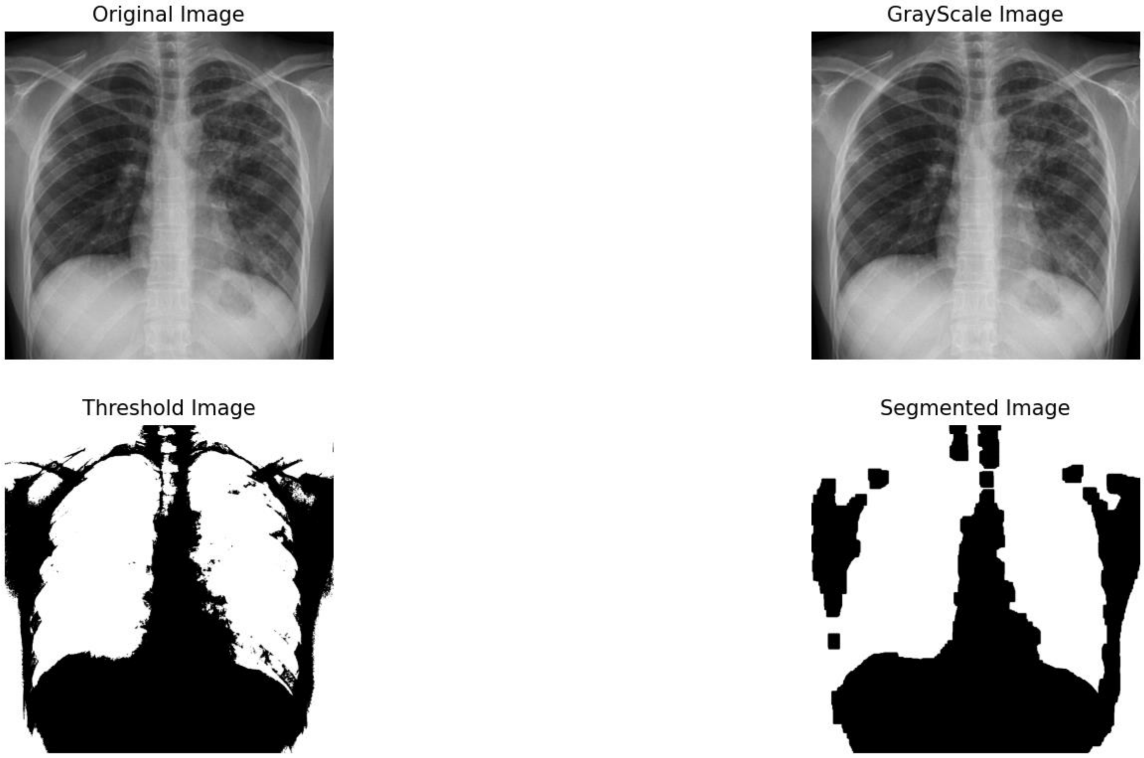
IMAGE SEGMENTATION:



```
#image segmentation using thresholding
import numpy as np
import cv2

from matplotlib import pyplot as plt
img =
cv2.imread("C:\\Users\\gokul\\PycharmProjects\DIP\\tuberculosis.jpg")
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.imshow(img,cmap="gray")
plt.axis('off')
plt.title("Original Image")
plt.show()

#converting it into grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(8,8))
plt.imshow(gray,cmap="gray")
plt.axis('off')
plt.title("GrayScale Image")
plt.show()
```



```
#Converting to a Binary Inverted Image
ret, thresh = cv2.threshold(gray, 0,
255,cv2.THRESH_BINARY_INV +cv2.THRESH_OTSU)
plt.figure(figsize=(8,8))
plt.imshow(thresh,cmap="gray")
plt.axis('off')
plt.title("Threshold Image")
plt.show()

#Segmenting the Image
kernel = np.ones((3, 3), np.uint8)
closing = cv2.morphologyEx(thresh,
cv2.MORPH_CLOSE,kernel, iterations = 15)
bg = cv2.dilate(closing, kernel, iterations = 1)
dist_transform = cv2.distanceTransform(closing,
cv2.DIST_L2, 0)
ret, fg = cv2.threshold(dist_transform,
0.02*dist_transform.max(), 255, 0)
cv2.imshow('image', fg)
plt.figure(figsize=(8,8))
plt.imshow(fg,cmap="gray")
plt.axis('off')
plt.title("Segmented Image")
plt.show()

#final output
plt.figure(figsize=(10,10))

plt.subplot(2,2,1)
plt.axis('off')
plt.title("Original Image")
plt.imshow(img,cmap="gray")

plt.subplot(2,2,2)
plt.imshow(gray,cmap="gray")
plt.axis('off')
plt.title("GrayScale Image")
plt.subplot(2,2,3)
plt.imshow(thresh,cmap="gray")
plt.axis('off')
plt.title("Threshold Image")
plt.subplot(2,2,4)
plt.imshow(fg,cmap="gray")
plt.axis('off')
plt.title("Segmented Image")
plt.show()
```

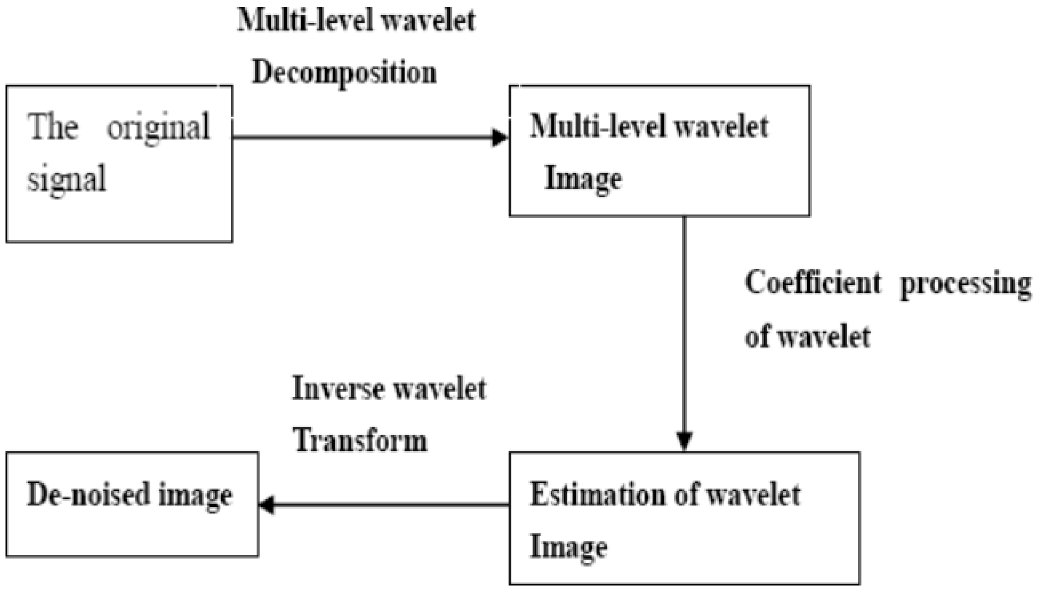


FEATURE EXTRACTION:

The wavelet analysis method is a time-frequency analysis method which selects the appropriate frequency band adaptively based on the characteristics of the signal. Then the frequency band matches the spectrum which improves the time-frequency resolution.

The basic method of the wavelet transform is selecting a function whose integral is zero in time-domain as the basic wavelet. By the expansion and translation of the basic wavelet, we can get a family function which may constitute a framework for the function space. We decompose the signal by projecting the analysis signals on the framework. The signal in original time domain can get a time-scale expression by several scaling in the wavelet transform domain. Then we are able to achieve the most effective signal processing purpose transform domain.

The essence of wavelet de-noising is searching for the best mapping of signals from of the actual space to wavelet function space in order to get the best restoration of the original signal. From the view of the signal processing, the wavelet de-noising is a signal filtering. The wavelet de-noising is able to retain the characteristics of the image successfully. Actually, it is comprehensive with feature extraction and low-pass filtering

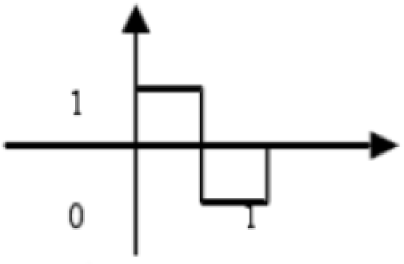

Multi-level wavelet Decomposition
The original signal
Multi-level wavelet Image
Coefficient processing of wavelet
Inverse wavelet Transform
De-noised image
Estimation of wavelet Image

In the digital image processing, the choice of the basic wavelet is very important. Haar wavelet is unique symmetry wavelet in the whole orthogonal wavelet. Haar wavelet's support is very short which can be high-pass and low-pass filter, what's more, it can save the computational complexity. So, this paper chooses Haar wavelet as the basis function for digital image analysis. The expression of Harr wavelet and its scaling function follows as follows

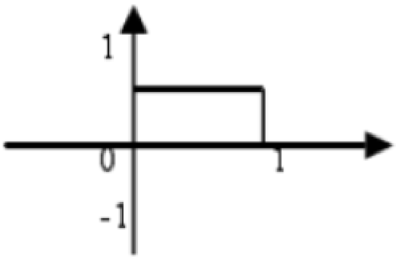
$$\psi_k(x) = \begin{cases} 1 & 0 < \pi \leq 0.5 \\ -1 & 0.5 \leq \mathrm{x} < 1 \\ 0 & \text{else} \end{cases}$$

$$\phi(x) = \begin{cases} 1, & 0 \leq x \leq 1 \\ 0, & \text{else} \end{cases}$$

The corresponding function graphs are shown:



$\psi_h(x)$



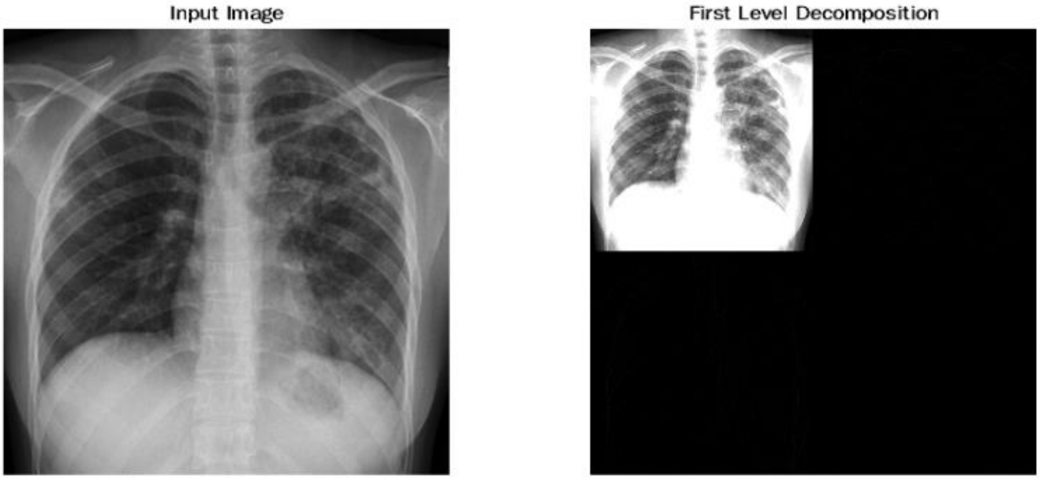
$\phi_h(x)$

The experiment has three steps by using wavelet analysis to deal with image noise.

- Wavelet decomposition of two-dimensional image.
- Quantifying the high-frequency coefficients after the decomposition.
- Reconstruction image signal of two-dimensional wavelet

[Matlab Code]:

```
%Read Input Image
Input_Image=imread('tuberculosis.jpg');

%Red Component of Colour Image
Red_Input_Image=Input_Image(:,:,1);
%Green Component of Colour Image
Green_Input_Image=Input_Image(:,:,2);
%Blue Component of Colour Image
Blue_Input_Image=Input_Image(:,:,3);

%Apply Two Dimensional Discrete Wavelet Transform
[LLr,LHr,HLr,HHr]=dwt2(Red_Input_Image,'haar');
[LLg,LHg,HLg,HHg]=dwt2(Green_Input_Image,'haar');
[LLb,LHb,HLb,HHb]=dwt2(Blue_Input_Image,'haar');

First_Level_Decomposition(:,:,1)=[LLr,LHr;HLr,HHr];
First_Level_Decomposition(:,:,2)=[LLg,LHg;HLg,HHg];
First_Level_Decomposition(:,:,3)=[LLb,LHb;HLb,HHb];
First_Level_Decomposition=uint8(First_Level_Decomposition);

%Display Image
subplot(1,2,1);imshow(Input_Image);title('Input Image');
subplot(1,2,2);imshow(First_Level_Decomposition,[]);title('First Level Decomposition');
```



Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

DIGITAL IMAGE PROCESSING – 'J' Component – 3rd Review

Title of the project: TUBERCULOSIS DETECTION WITH X-RAY IMAGES

Team Members:

1) GOPINI SAI BHUVAN – 20MIS0104
2) GOKUL R – 20MIS0332

INTRODUCTION:

Tuberculosis (TB) is caused by bacteria (Mycobacterium tuberculosis) that most often affect the lungs. Tuberculosis is curable and preventable. TB is spread from person to person through the air. When people with lung TB cough, sneeze or spit, they propel the TB germs into the air. The risk of TUBERCULOSIS is immense for many, especially in developing nations where billions face energy poverty and rely on polluting forms of energy. The WHO estimates that over 4 million premature deaths occur annually from household air pollution-related diseases including pneumonia. Over 150 million people get infected with pneumonia on an annual basis especially children under 5 years old. In such regions, the problem can be further aggravated due to the dearth of medical resources and personnel. For example, in Africa's 57 nations, a gap of 2.3 million doctors and nurses exists. For these populations, accurate and fast diagnosis means everything. It can guarantee timely access to treatment and save much needed time and money for those already experiencing poverty.

This project is a part of the Chest X-Ray Images (TUBERCULOSIS) held on Kaggle.

AIM:

Build an algorithm to automatically identify whether a patient is suffering from TUBERCULOSIS or not by looking at chest X-ray images. The algorithm had to be extremely accurate because lives of people is at stake.

Environment and tools:

1. scikit-learn

2. keras

3. numpy

4. pandas

5. imageio

6. matplotlib

Dataset used:

The dataset can be downloaded from the Kaggle website which can be found here

In this kernel, we will build a model that can look at a chest x-ray and predict whether a person has TB or not. The model will be trained on a dataset of 800 images from two sources:

- Shenzhen, China (Folder: ChinaSet_AllFiles)
- Montgomery, USA (Folder: Montgomery)

The dataset is quite small but by using a CNN and data augmentation, the final accuracy and F1 score that we get will be greater than 0.8. Because we need to use as many images as possible for training, the validation set will contain only 120 images. This is 15% of the data. With a small dataset and a very small validation set, we've deployed the model as a Tensorflowjs and it can be tested.

CONVOLUTION NEURAL NETWORK:

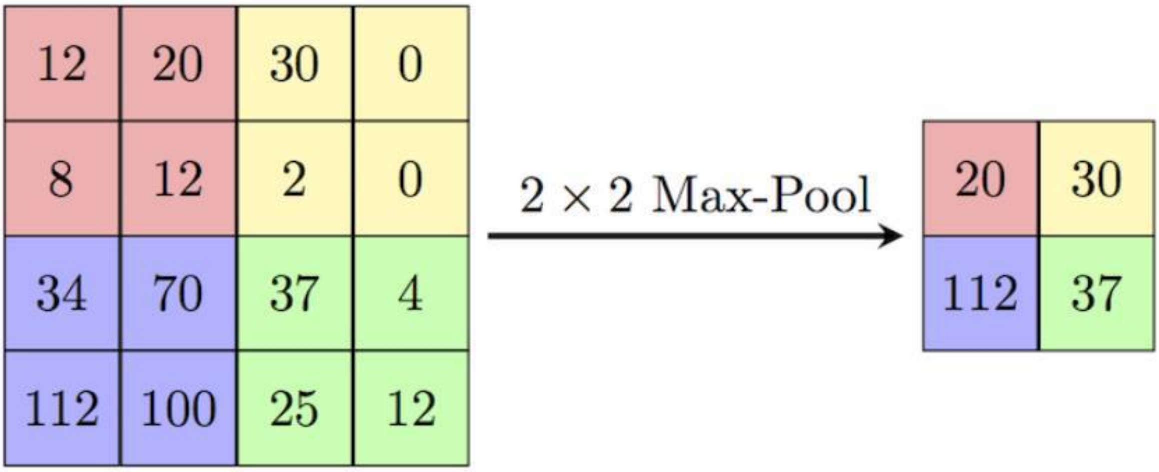
Convolutional Neural Networks are a type of Deep Neural Networks. This NN uses Convolutions to extract meaningful information or patterns from the input features, which is further used to build the subsequent layers of neural network computations.

Convolutional Neural Networks perform amazingly well on Image data and computer vision. Following are a few reasons, why CNNs perform well on image data:

- One important difference between the Dense layer and the Convolutional layer is, dense layers are good at finding global patterns, while convolutional layers are good at finding local patterns.
- Convolutional layers also understand spatial data. Initial layers of the convnets (Convolutional Networks) detect low-level patterns like edges and lines, while the deeper layers detect more complex patterns like ears, nose, eyes, etc.
- Once learned, CNN can detect a pattern anywhere in the image. So, even if the images are sheared or modified, neural networks can still perform well.

Max Pooling:

Max pooling is a technique of aggressive down sampling of the feature map.



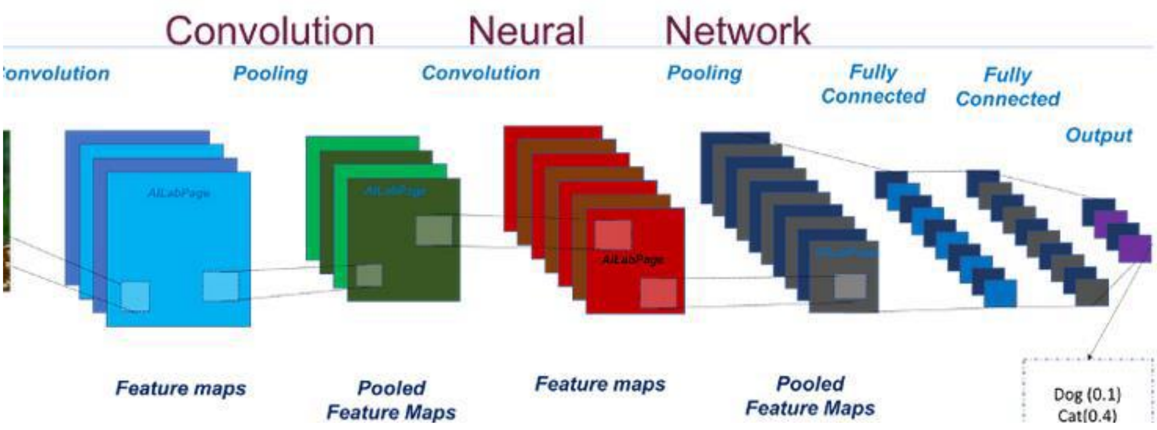
12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

Building a CNN Model,

A Typical CNN:

The following image is a descriptive representation of how a convolutional neural network will look like.



The input image is fed to the neural network. The Convnet then performs convolutions over the input image. Each convolution filter will result in its own output feature map. As we can look at the image, multiple convolutional filters are applied over the input image, as a result, we have transformed a single image into multiple output feature maps.

Each feature map will hold specific information about the image. The number of these layers is called the depth of the channels.

Next, comes the pooling stage. In pooling, we downsize the input feature map, while retaining the most useful information. So, each value in the feature map after max-pooling will represent a larger patch of the input feature map. Max pooling helps convnets to detect more complex patterns with less computing power.

Multiple convolutional layers and max-pooling layers can be arranged successively to form the deep neural network. The number of layers and the depth of each convolutional layer are provided by us, there are no strict guidelines for these hyperparameters and we can experiment on our own to find the combination that works best for our model.

Finally, these convolutional layers are connected to a Dense layer (Fully connected), or a regular neural network. We are free to add multiple layers in this dense layer as well. The final output layer of this neural network will have two nodes, one for each class

CODING:

Image classification using CNN:

Process followed;

Step 1: Choose a Dataset

Step 2: Prepare Dataset for Training

Step 3: Create Training Data.

Step 4: Shuffle the Dataset.

Step 5: Assigning Labels and Features.

Step 6: CREATING A DIRECTORY STRUCTURE

Step 7: copying trained images to aug_dir

Step 8: Model architecture

Step 9: Training and evaluating the model

Step 10: plot the graph (accuracy, loss)

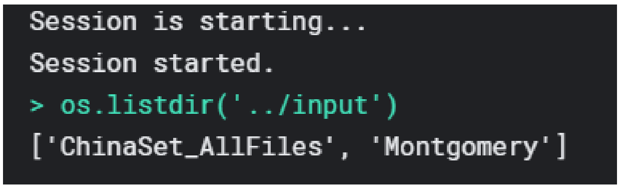
Step 11: confusion matrix

Step 12: Final report

```
from numpy.random import seed
seed(101)
from tensorflow import set_random_seed
set_random_seed(101)

import pandas as pd
import numpy as np

import tensorflow

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D,
MaxPooling2D, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.metrics import binary_accuracy

import os
import cv2

import imageio
import skimage
import skimage.io
import skimage.transform

from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
```

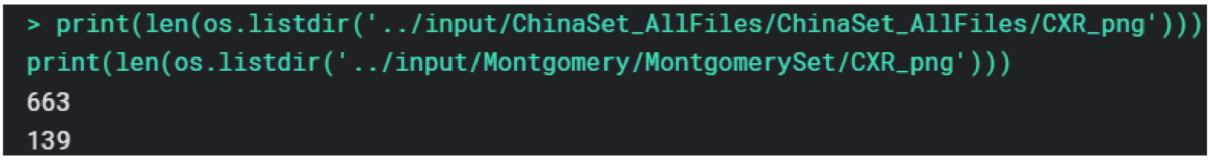
```
# Total number of images we want to have in each class
NUM_AUG_IMAGES_WANTED = 1000
# We will resize the images
IMAGE_HEIGHT = 96
IMAGE_WIDTH = 96
```

Getting Files:

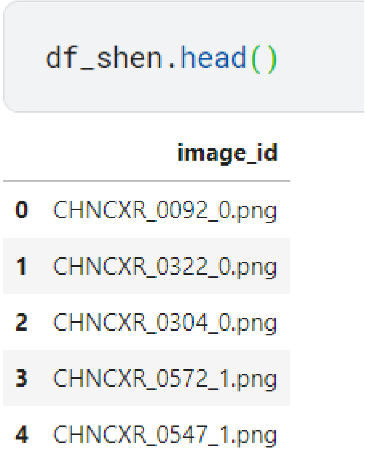


```
Session is starting...
Session started.
> os.listdir('../input')
['ChinaSet_AllFiles', 'Montgomery']
```

Getting number of images in each folder:

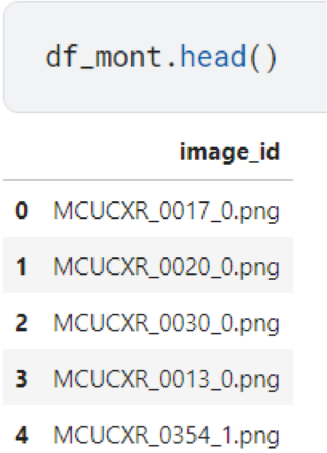


```
> print(len(os.listdir('../input/ChinaSet_AllFiles/ChinaSet_AllFiles/CXR_png')))
print(len(os.listdir('../input/Montgomery/MontgomerySet/CXR_png')))
663
139
```



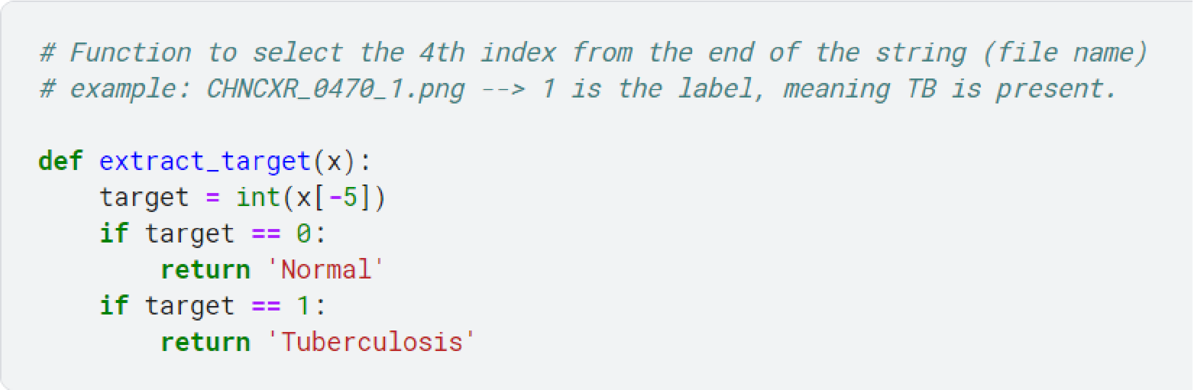
```
df_shen.head()
```

	image_id
0	CHNCXR_0092_0.png
1	CHNCXR_0322_0.png
2	CHNCXR_0304_0.png
3	CHNCXR_0572_1.png
4	CHNCXR_0547_1.png

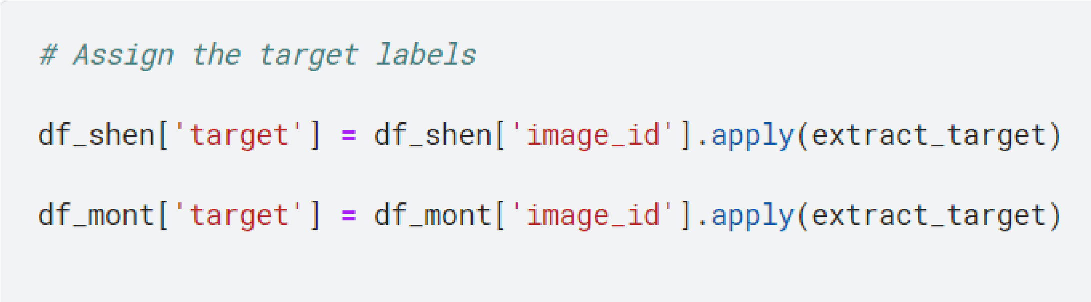


```
df_mont.head()
```

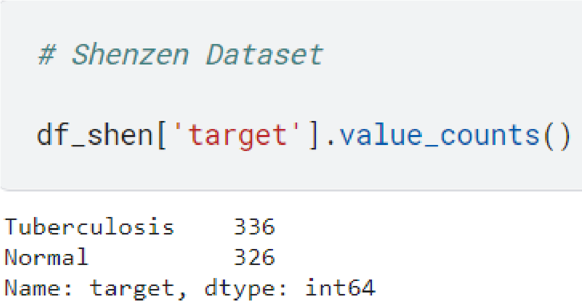
	image_id
0	MCUCXR_0017_0.png
1	MCUCXR_0020_0.png
2	MCUCXR_0030_0.png
3	MCUCXR_0013_0.png
4	MCUCXR_0354_1.png



```python
# Function to select the 4th index from the end of the string (file name)
# example: CHNCXR_0470_1.png --> 1 is the label, meaning TB is present.

def extract_target(x):
    target = int(x[-5])
    if target == 0:
        return 'Normal'
    if target == 1:
        return 'Tuberculosis'
```

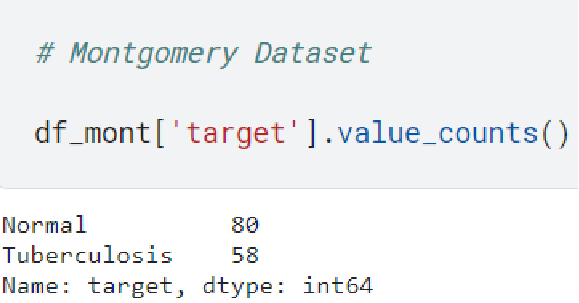


```python
# Assign the target labels

df_shen['target'] = df_shen['image_id'].apply(extract_target)

df_mont['target'] = df_mont['image_id'].apply(extract_target)
```



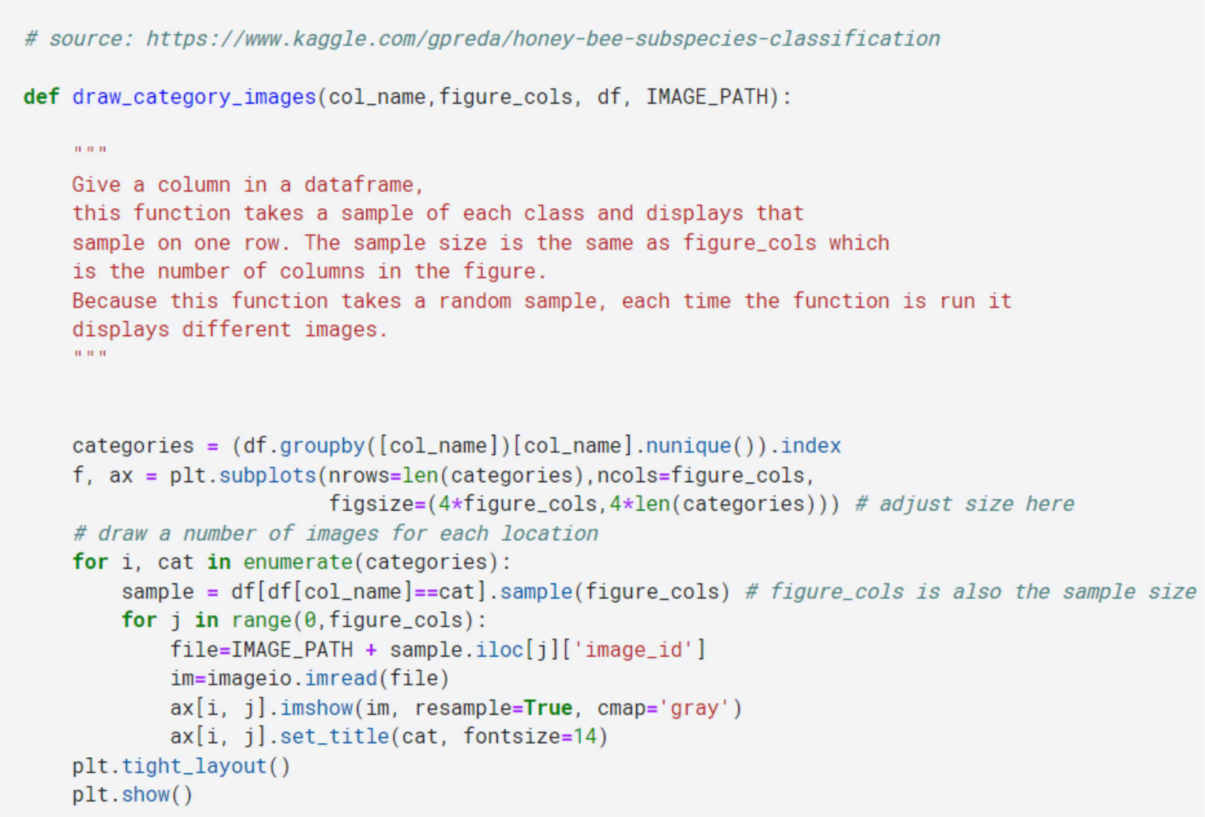
```python
# Shenzen Dataset

df_shen['target'].value_counts()
```

```
Tuberculosis    336
Normal          326
Name: target, dtype: int64
```

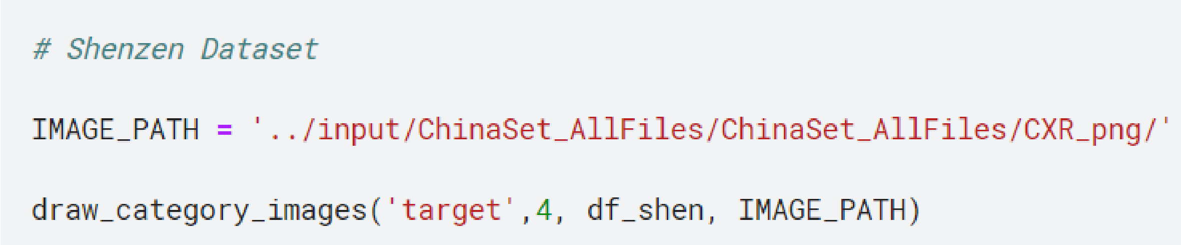


```python
# Montgomery Dataset

df_mont['target'].value_counts()
```

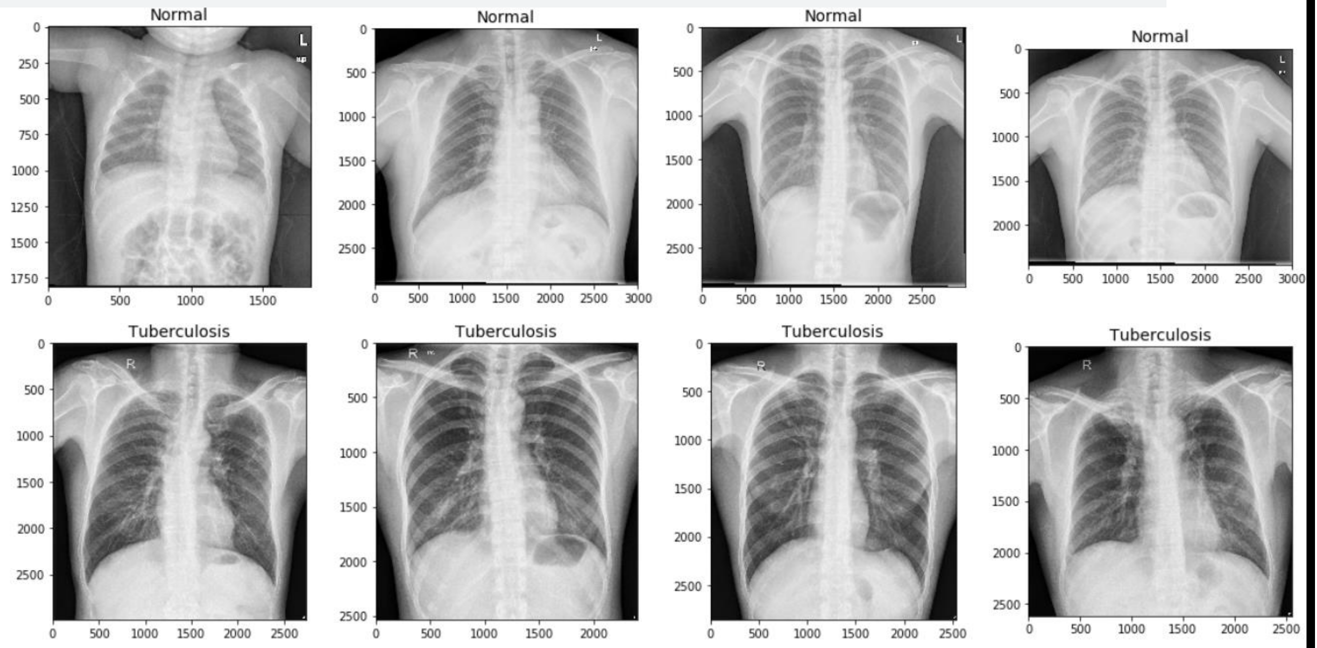
```
Normal          80
Tuberculosis    58
Name: target, dtype: int64
```

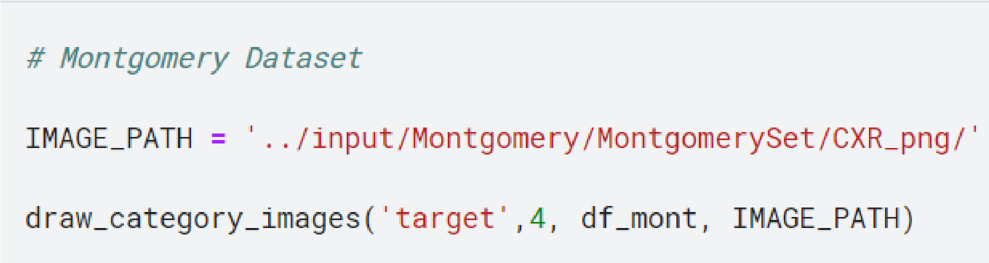


```
# source: https://www.kaggle.com/gpreda/honey-bee-subspecies-classification

def draw_category_images(col_name,figure_cols, df, IMAGE_PATH):

    """
    Give a column in a dataframe,
    this function takes a sample of each class and displays that
    sample on one row. The sample size is the same as figure_cols which
    is the number of columns in the figure.
    Because this function takes a random sample, each time the function is run it
    displays different images.
    """


    categories = (df.groupby([col_name])[col_name].nunique()).index
    f, ax = plt.subplots(nrows=len(categories),ncols=figure_cols,
                         figsize=(4*figure_cols,4*len(categories))) # adjust size here
    # draw a number of images for each location
    for i, cat in enumerate(categories):
        sample = df[df[col_name]==cat].sample(figure_cols) # figure_cols is also the sample size
        for j in range(0,figure_cols):
            file=IMAGE_PATH + sample.iloc[j]['image_id']
            im=imageio.imread(file)
            ax[i, j].imshow(im, resample=True, cmap='gray')
            ax[i, j].set_title(cat, fontsize=14)
    plt.tight_layout()
    plt.show()
```

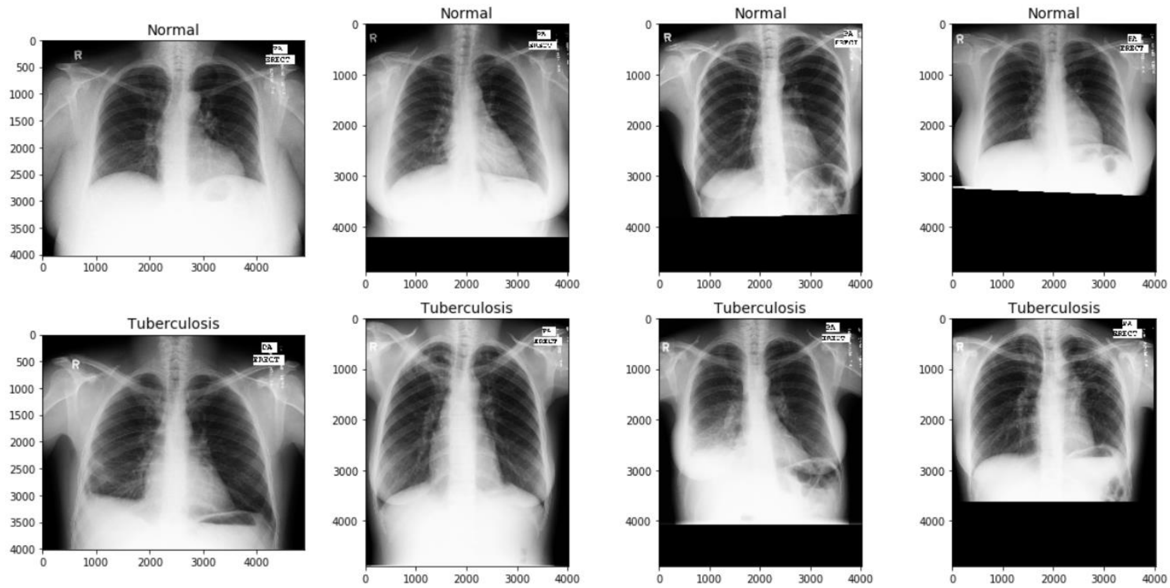


```
# Shenzen Dataset

IMAGE_PATH = '../input/ChinaSet_AllFiles/ChinaSet_AllFiles/CXR_png/'

draw_category_images('target',4, df_shen, IMAGE_PATH)
```

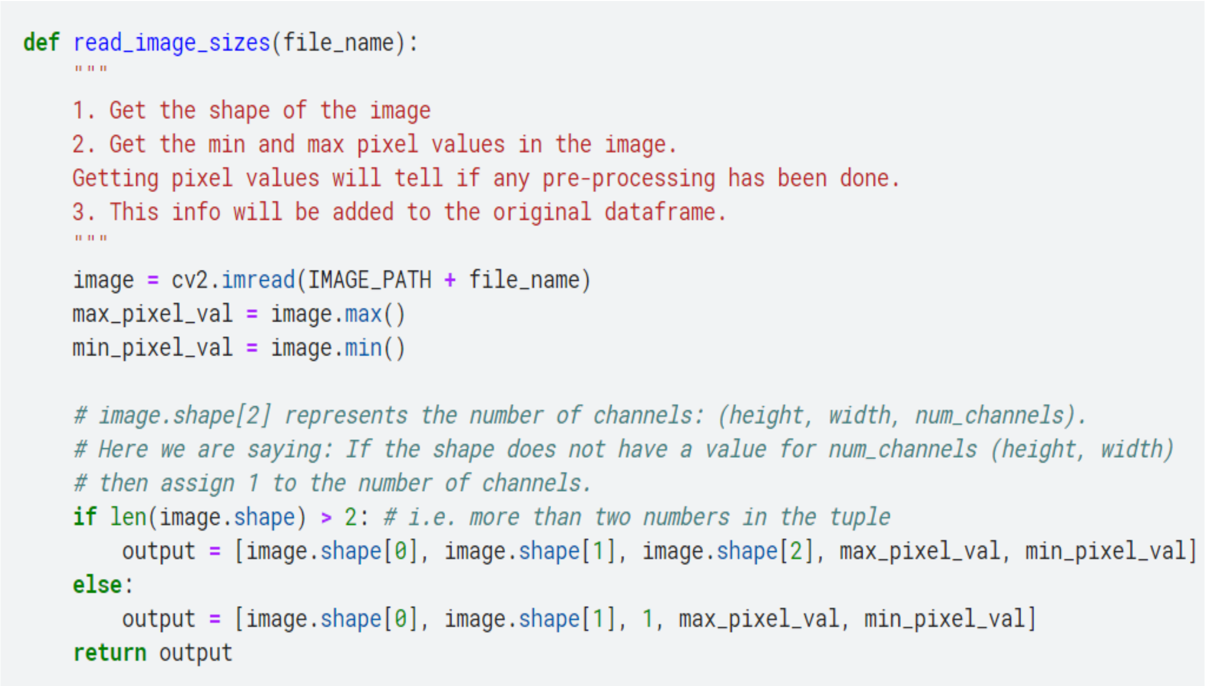




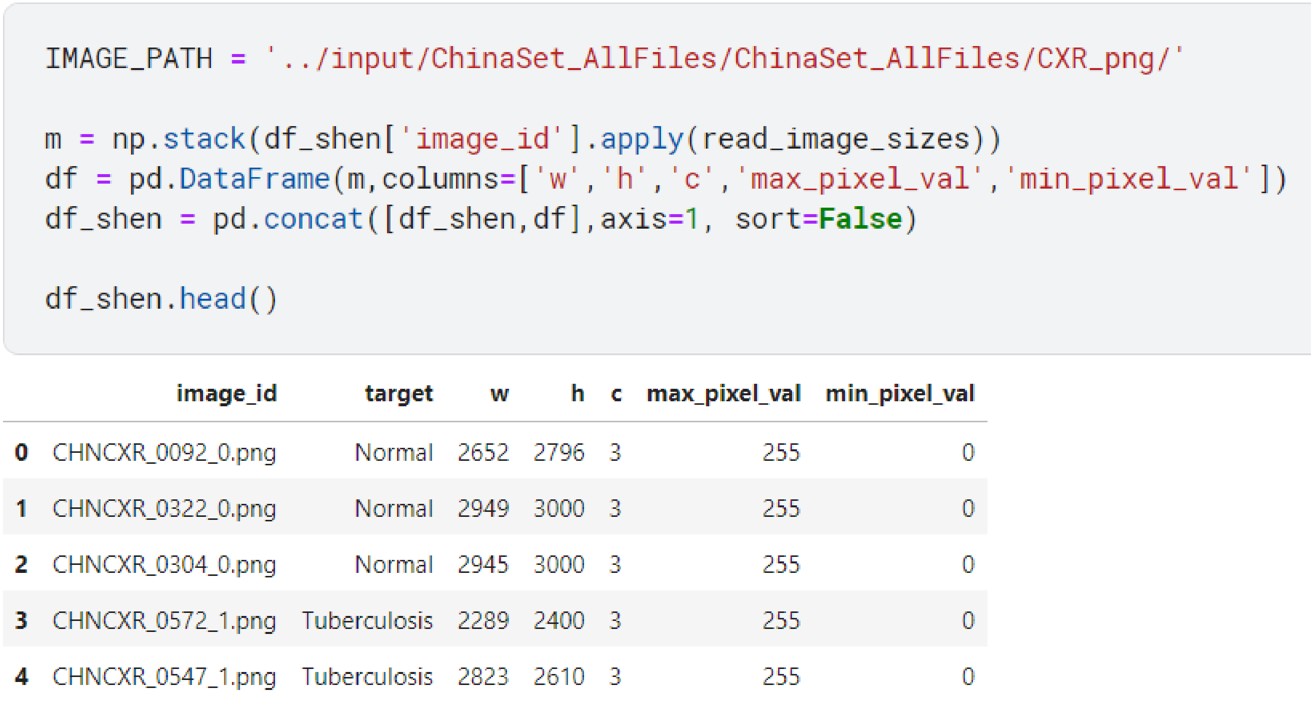
```
# Montgomery Dataset

IMAGE_PATH = '../input/Montgomery/MontgomerySet/CXR_png/'

draw_category_images('target',4, df_mont, IMAGE_PATH)
```



What is the shape of each image and what are its max and min pixel values?

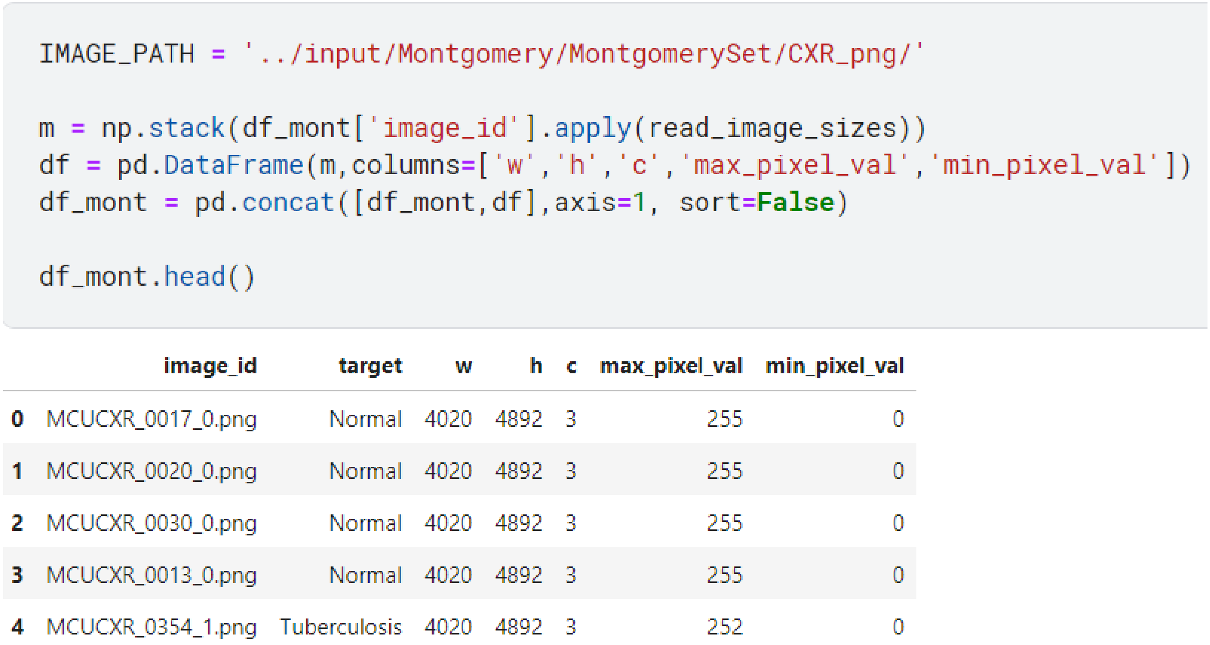


```
def read_image_sizes(file_name):
    """
    1. Get the shape of the image
    2. Get the min and max pixel values in the image.
    Getting pixel values will tell if any pre-processing has been done.
    3. This info will be added to the original dataframe.
    """
    image = cv2.imread(IMAGE_PATH + file_name)
    max_pixel_val = image.max()
    min_pixel_val = image.min()

    # image.shape[2] represents the number of channels: (height, width, num_channels).
    # Here we are saying: If the shape does not have a value for num_channels (height, width)
    # then assign 1 to the number of channels.
    if len(image.shape) > 2: # i.e. more than two numbers in the tuple
        output = [image.shape[0], image.shape[1], image.shape[2], max_pixel_val, min_pixel_val]
    else:
        output = [image.shape[0], image.shape[1], 1, max_pixel_val, min_pixel_val]
    return output
```



```python
IMAGE_PATH = '../input/ChinaSet_AllFiles/ChinaSet_AllFiles/CXR_png/'

m = np.stack(df_shen['image_id'].apply(read_image_sizes))
df = pd.DataFrame(m,columns=['w','h','c','max_pixel_val','min_pixel_val'])
df_shen = pd.concat([df_shen,df],axis=1, sort=False)

df_shen.head()
```

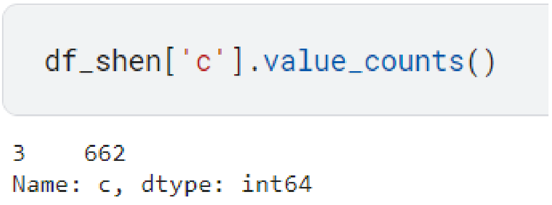
	image_id	target	w	h	c	max_pixel_val	min_pixel_val
0	CHNCXR_0092_0.png	Normal	2652	2796	3	255	0
1	CHNCXR_0322_0.png	Normal	2949	3000	3	255	0
2	CHNCXR_0304_0.png	Normal	2945	3000	3	255	0
3	CHNCXR_0572_1.png	Tuberculosis	2289	2400	3	255	0
4	CHNCXR_0547_1.png	Tuberculosis	2823	2610	3	255	0



```python
IMAGE_PATH = '../input/Montgomery/MontgomerySet/CXR_png/'

m = np.stack(df_mont['image_id'].apply(read_image_sizes))
df = pd.DataFrame(m,columns=['w','h','c','max_pixel_val','min_pixel_val'])
df_mont = pd.concat([df_mont,df],axis=1, sort=False)

df_mont.head()
```

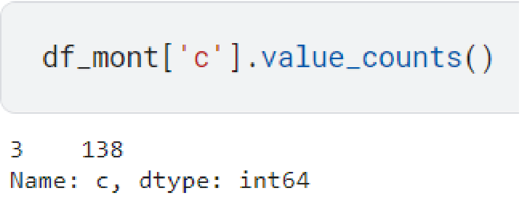
	image_id	target	w	h	c	max_pixel_val	min_pixel_val
0	MCUCXR_0017_0.png	Normal	4020	4892	3	255	0
1	MCUCXR_0020_0.png	Normal	4020	4892	3	255	0
2	MCUCXR_0030_0.png	Normal	4020	4892	3	255	0
3	MCUCXR_0013_0.png	Normal	4020	4892	3	255	0
4	MCUCXR_0354_1.png	Tuberculosis	4020	4892	3	252	0

How many channels do the images in each dataset have?



```python
df_shen['c'].value_counts()
```

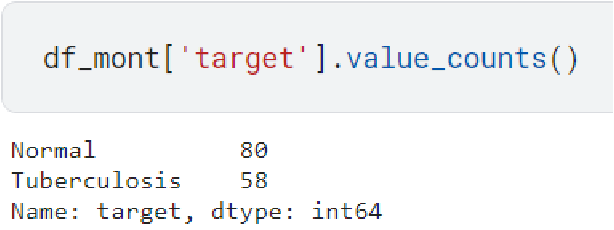
```
3    662
Name: c, dtype: int64
```



```
df_mont['c'].value_counts()
```

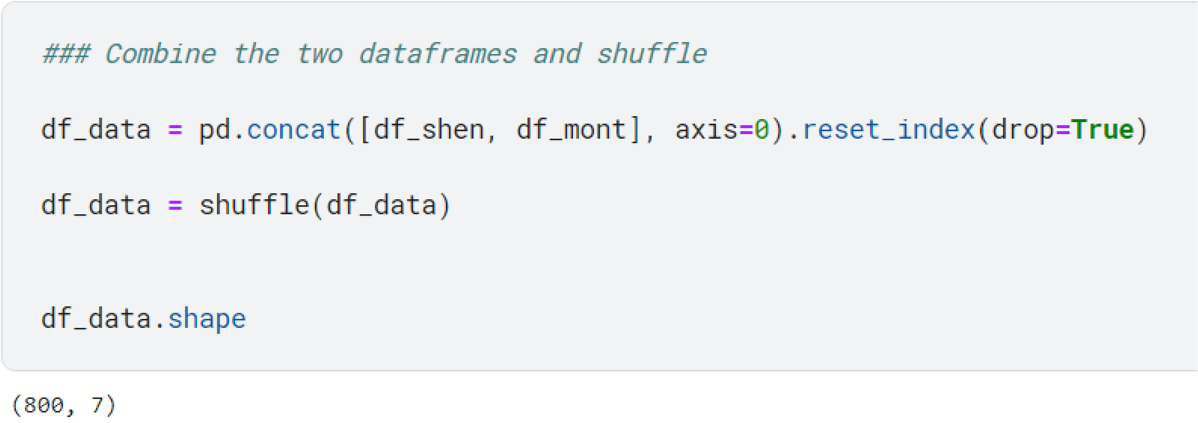
```
3    138
Name: c, dtype: int64
```

Create the Train and Val Sets



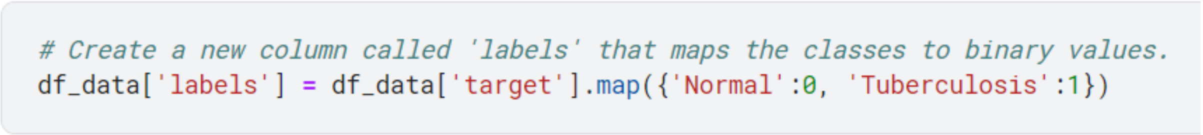
```
df_mont['target'].value_counts()
```

```
Normal          80
Tuberculosis    58
Name: target, dtype: int64
```

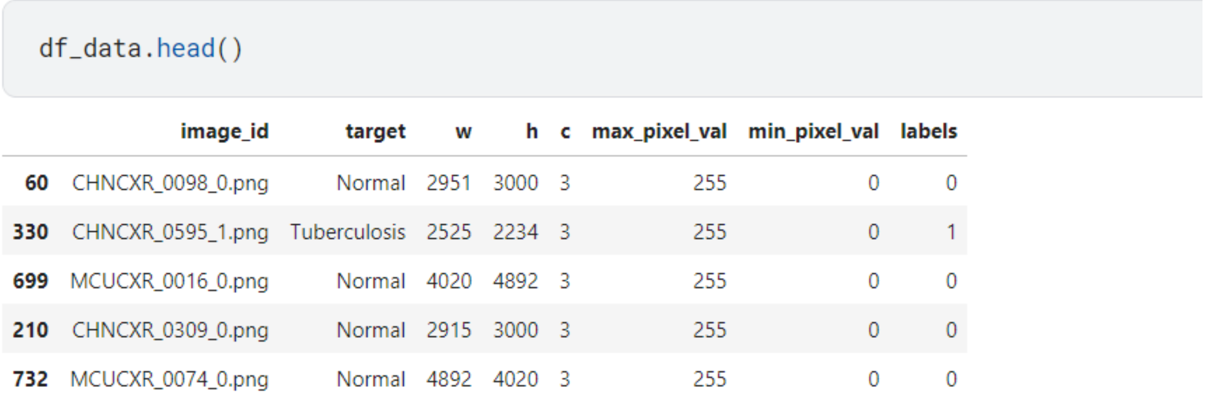


```
### Combine the two dataframes and shuffle

df_data = pd.concat([df_shen, df_mont], axis=0).reset_index(drop=True)

df_data = shuffle(df_data)


df_data.shape
```

```
(800, 7)
```

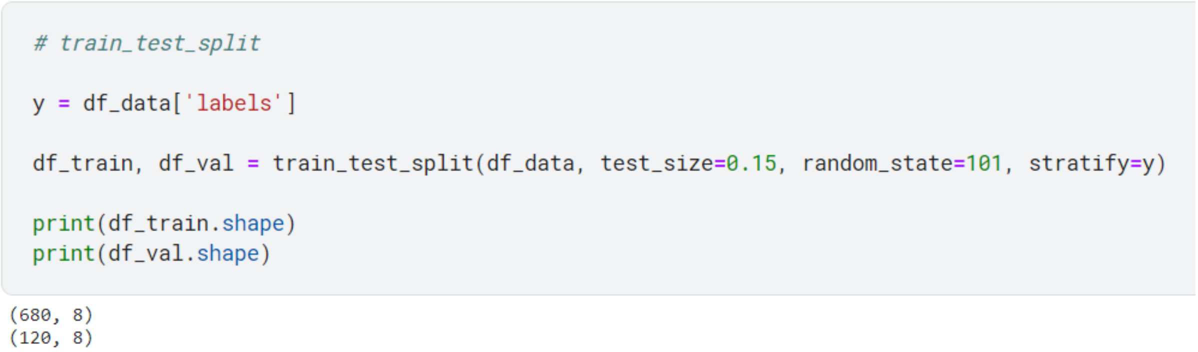


```python
# Create a new column called 'labels' that maps the classes to binary values.
df_data['labels'] = df_data['target'].map({'Normal':0, 'Tuberculosis':1})
```



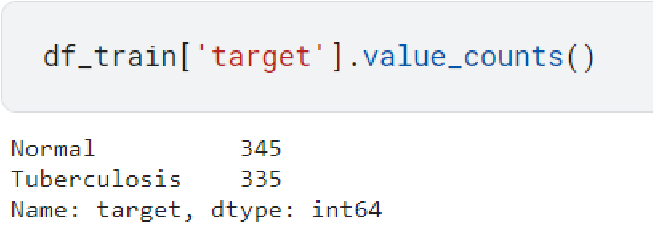
```python
df_data.head()
```

	image_id	target	w	h	c	max_pixel_val	min_pixel_val	labels
60	CHNCXR_0098_0.png	Normal	2951	3000	3	255	0	0
330	CHNCXR_0595_1.png	Tuberculosis	2525	2234	3	255	0	1
699	MCUCXR_0016_0.png	Normal	4020	4892	3	255	0	0
210	CHNCXR_0309_0.png	Normal	2915	3000	3	255	0	0
732	MCUCXR_0074_0.png	Normal	4892	4020	3	255	0	0



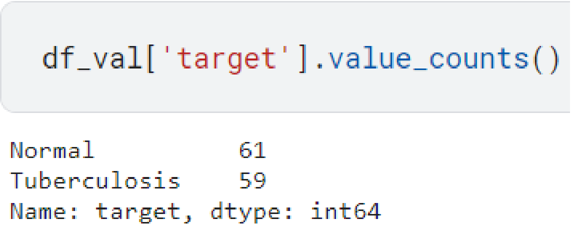
```python
# train_test_split

y = df_data['labels']

df_train, df_val = train_test_split(df_data, test_size=0.15, random_state=101, stratify=y)

print(df_train.shape)
print(df_val.shape)
```

```
(680, 8)
(120, 8)
```



```python
df_train['target'].value_counts()
```

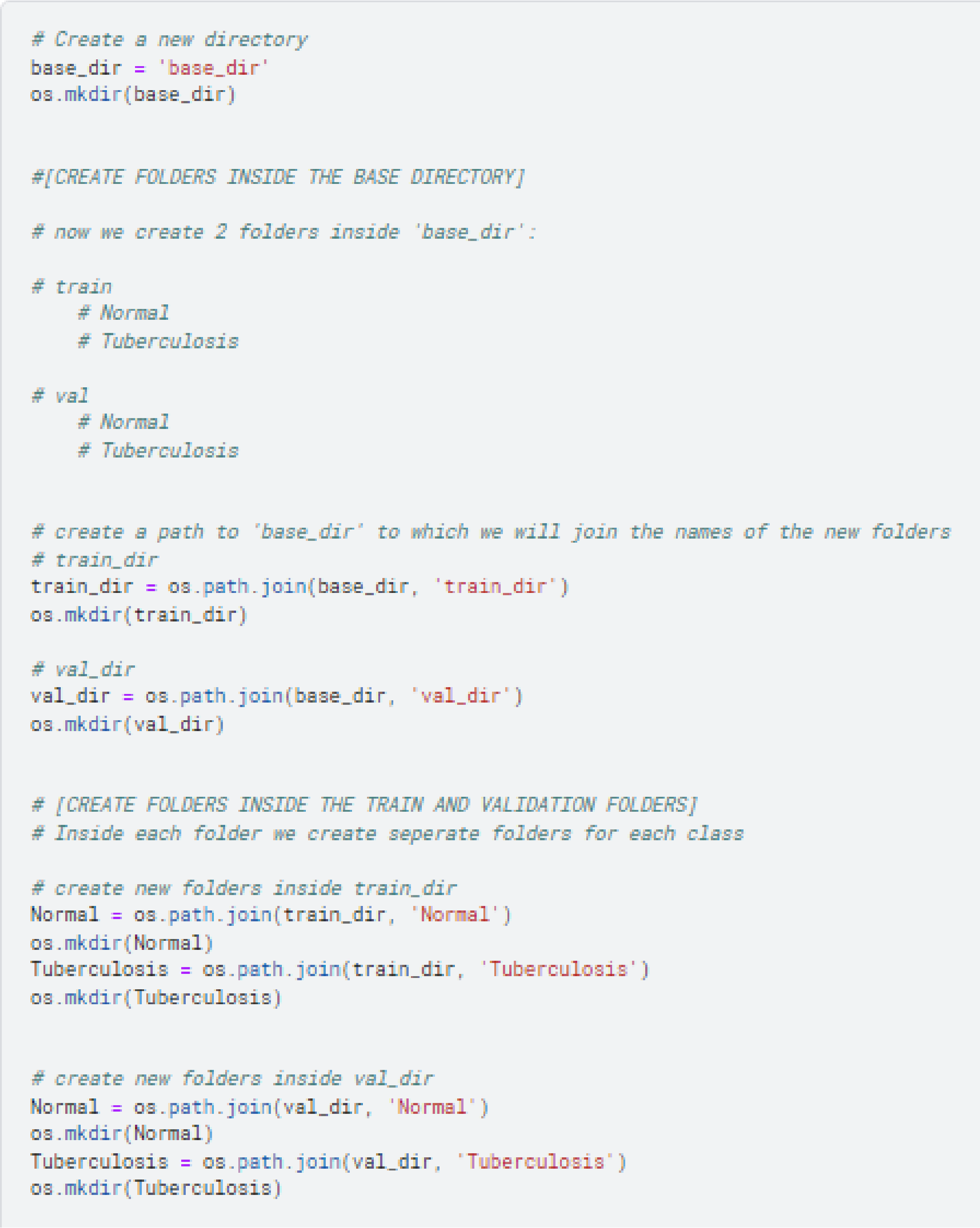
```
Normal          345
Tuberculosis    335
Name: target, dtype: int64
```



```python
df_val['target'].value_counts()
```

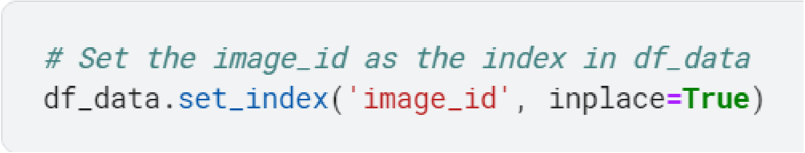
```
Normal          61
Tuberculosis    59
Name: target, dtype: int64
```

Create a directory structure:

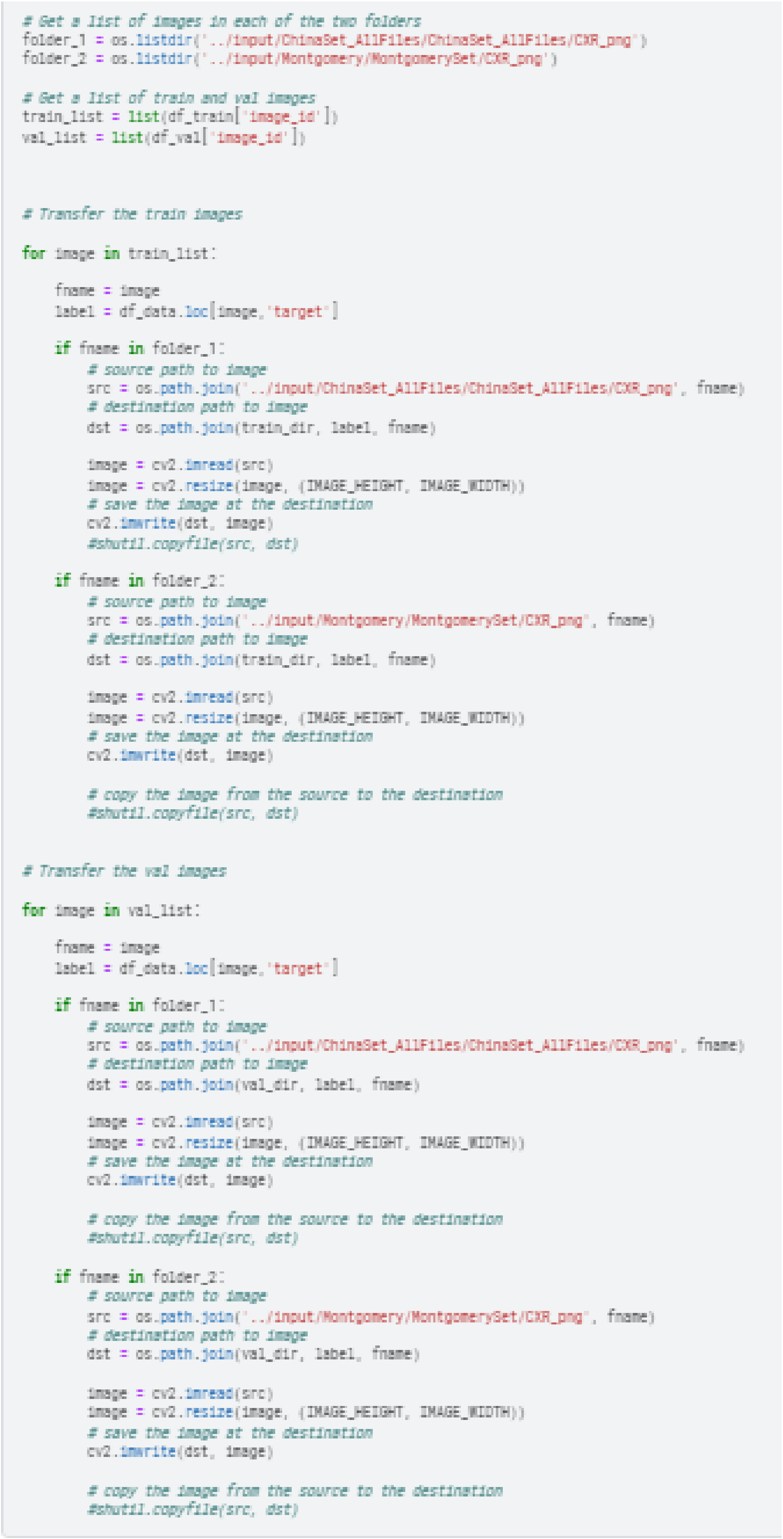


```
# Create a new directory
base_dir = 'base_dir'
os.mkdir(base_dir)


#[CREATE FOLDERS INSIDE THE BASE DIRECTORY]

# now we create 2 folders inside 'base_dir':

# train
    # Normal
    # Tuberculosis

# val
    # Normal
    # Tuberculosis


# create a path to 'base_dir' to which we will join the names of the new folders
# train_dir
train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)

# val_dir
val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)


# [CREATE FOLDERS INSIDE THE TRAIN AND VALIDATION FOLDERS]
# Inside each folder we create seperate folders for each class

# create new folders inside train_dir
Normal = os.path.join(train_dir, 'Normal')
os.mkdir(Normal)
Tuberculosis = os.path.join(train_dir, 'Tuberculosis')
os.mkdir(Tuberculosis)


# create new folders inside val_dir
Normal = os.path.join(val_dir, 'Normal')
os.mkdir(Normal)
Tuberculosis = os.path.join(val_dir, 'Tuberculosis')
os.mkdir(Tuberculosis)
```

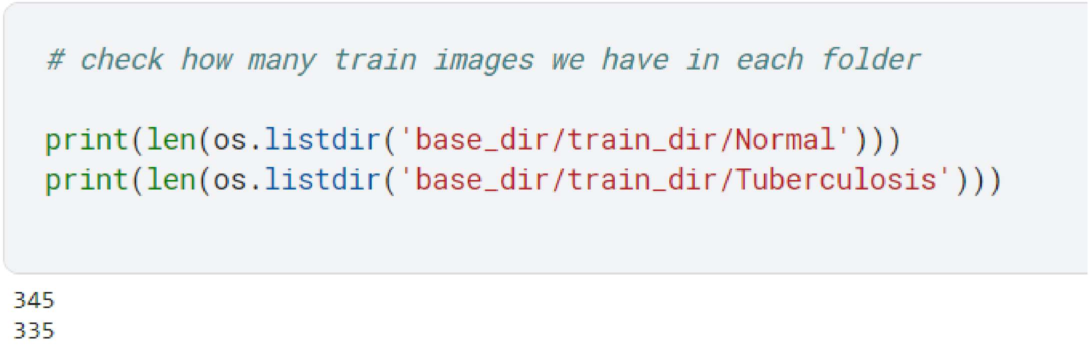
Transfer images into folders:



```
# Set the image_id as the index in df_data
df_data.set_index('image_id', inplace=True)
```



```
# Get a list of images in each of the two folders
folder_1 = os.listdir('../input/ChinaSet_AllFiles/ChinaSet_AllFiles/CXR_png')
folder_2 = os.listdir('../input/Montgomery/MontgomerySet/CXR_png')

# Get a list of train and val images
train_list = list(df_train['image_id'])
val_list = list(df_val['image_id'])



# Transfer the train images

for image in train_list:

    fname = image
    label = df_data.loc[image,'target']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/ChinaSet_AllFiles/ChinaSet_AllFiles/CXR_png', fname)
        # destination path to image
        dst = os.path.join(train_dir, label, fname)

        image = cv2.imread(src)
        image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))
        # save the image at the destination
        cv2.imwrite(dst, image)
        #shutil.copyfile(src, dst)

    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/Montgomery/MontgomerySet/CXR_png', fname)
        # destination path to image
        dst = os.path.join(train_dir, label, fname)

        image = cv2.imread(src)
        image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))
        # save the image at the destination
        cv2.imwrite(dst, image)

        # copy the image from the source to the destination
        #shutil.copyfile(src, dst)


# Transfer the val images

for image in val_list:

    fname = image
    label = df_data.loc[image,'target']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/ChinaSet_AllFiles/ChinaSet_AllFiles/CXR_png', fname)
        # destination path to image
        dst = os.path.join(val_dir, label, fname)

        image = cv2.imread(src)
        image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))
        # save the image at the destination
        cv2.imwrite(dst, image)

        # copy the image from the source to the destination
        #shutil.copyfile(src, dst)

    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/Montgomery/MontgomerySet/CXR_png', fname)
        # destination path to image
        dst = os.path.join(val_dir, label, fname)

        image = cv2.imread(src)
        image = cv2.resize(image, (IMAGE_HEIGHT, IMAGE_WIDTH))
        # save the image at the destination
        cv2.imwrite(dst, image)

        # copy the image from the source to the destination
        #shutil.copyfile(src, dst)
```



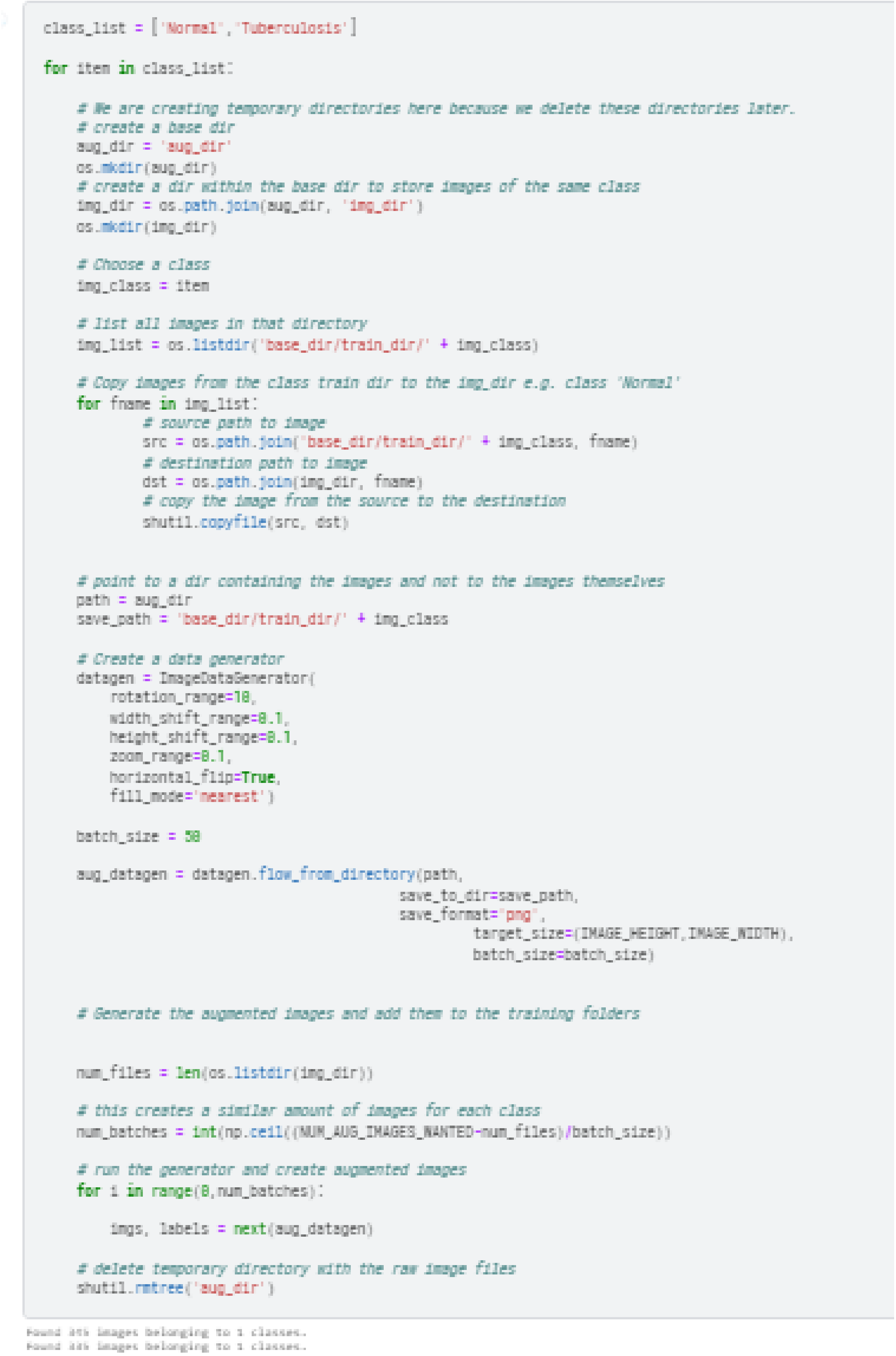
```
# check how many train images we have in each folder

print(len(os.listdir('base_dir/train_dir/Normal')))
print(len(os.listdir('base_dir/train_dir/Tuberculosis')))
```

```
345
335
```

```
# check how many val images we have in each folder

print(len(os.listdir('base_dir/val_dir/Normal')))
print(len(os.listdir('base_dir/val_dir/Tuberculosis')))
```

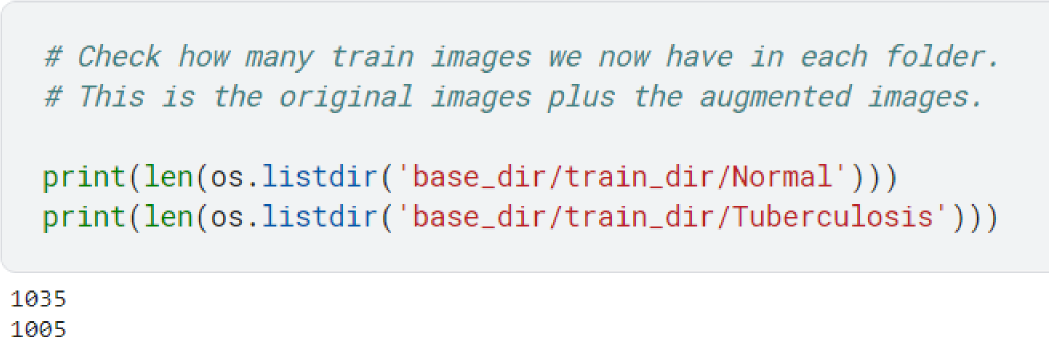
```
61
59
```

Copy the trained images to Aug_directory:



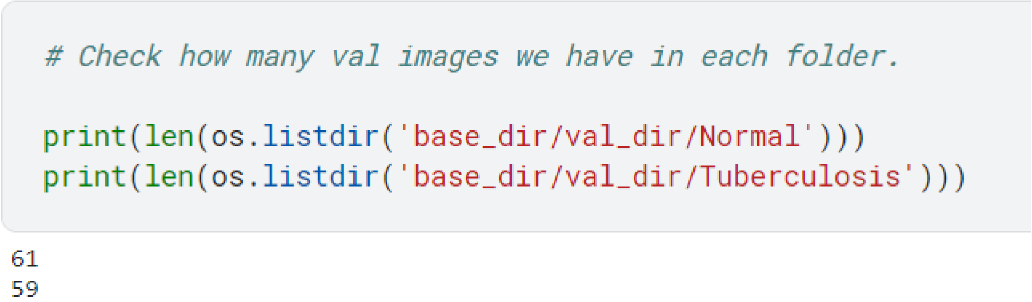
```python
class_list = ['Normal','Tuberculosis']

for item in class_list:

    # We are creating temporary directories here because we delete these directories later.
    # create a base dir
    aug_dir = 'aug_dir'
    os.mkdir(aug_dir)
    # create a dir within the base dir to store images of the same class
    img_dir = os.path.join(aug_dir, 'img_dir')
    os.mkdir(img_dir)

    # Choose a class
    img_class = item

    # list all images in that directory
    img_list = os.listdir('base_dir/train_dir/' + img_class)

    # Copy images from the class train dir to the img_dir e.g. class 'Normal'
    for fname in img_list:
            # source path to image
            src = os.path.join('base_dir/train_dir/' + img_class, fname)
            # destination path to image
            dst = os.path.join(img_dir, fname)
            # copy the image from the source to the destination
            shutil.copyfile(src, dst)


    # point to a dir containing the images and not to the images themselves
    path = aug_dir
    save_path = 'base_dir/train_dir/' + img_class

    # Create a data generator
    datagen = ImageDataGenerator(
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest')

    batch_size = 50

    aug_datagen = datagen.flow_from_directory(path,
                                           save_to_dir=save_path,
                                           save_format='png',
                                                    target_size=(IMAGE_HEIGHT,IMAGE_WIDTH),
                                                    batch_size=batch_size)


    # Generate the augmented images and add them to the training folders


    num_files = len(os.listdir(img_dir))

    # this creates a similar amount of images for each class
    num_batches = int(np.ceil((NUM_AUG_IMAGES_WANTED-num_files)/batch_size))

    # run the generator and create augmented images
    for i in range(0,num_batches):

        imgs, labels = next(aug_datagen)

    # delete temporary directory with the raw image files
    shutil.rmtree('aug_dir')
```

```
Found [illegible] images belonging to 1 classes.
Found [illegible] images belonging to 1 classes.
```



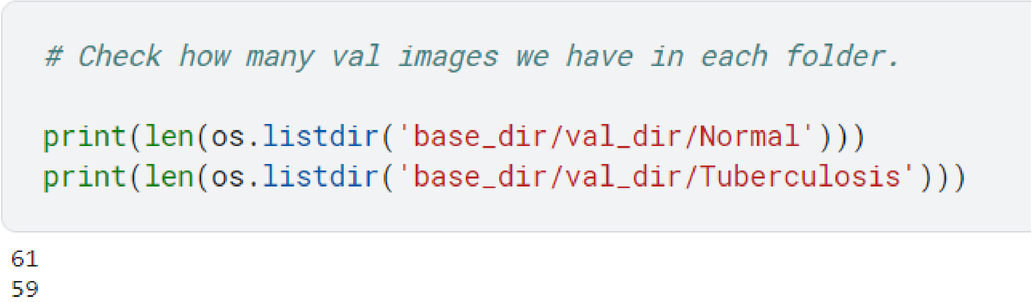
```
# Check how many train images we now have in each folder.
# This is the original images plus the augmented images.

print(len(os.listdir('base_dir/train_dir/Normal')))
print(len(os.listdir('base_dir/train_dir/Tuberculosis')))
```

```
1035
1005
```



```
# Check how many val images we have in each folder.

print(len(os.listdir('base_dir/val_dir/Normal')))
print(len(os.listdir('base_dir/val_dir/Tuberculosis')))
```

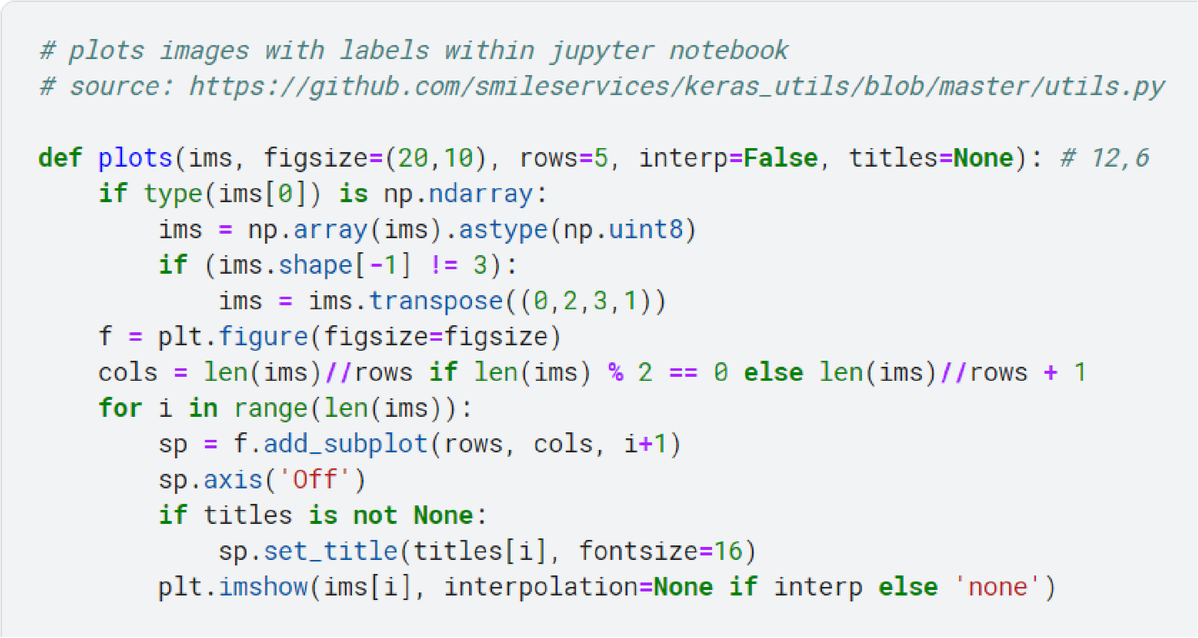
```
61
59
```



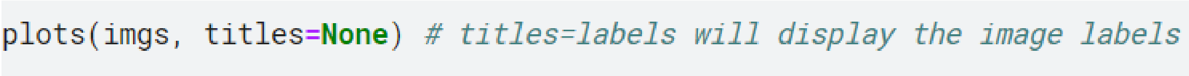
```
# Check how many val images we have in each folder.

print(len(os.listdir('base_dir/val_dir/Normal')))
print(len(os.listdir('base_dir/val_dir/Tuberculosis')))
```

```
61
59
```

Visualize a batch of augmented images:

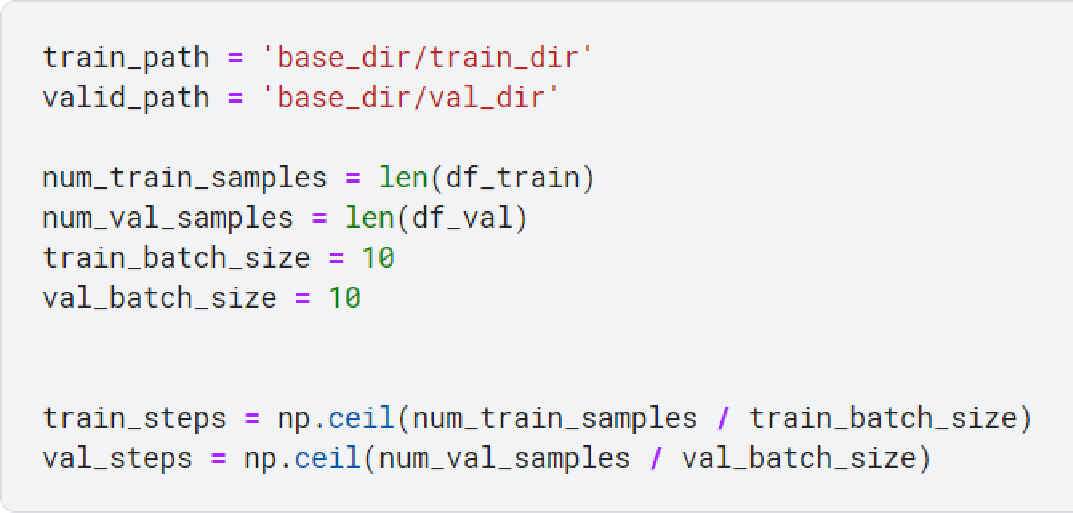


```
# plots images with labels within jupyter notebook
# source: https://github.com/smileservices/keras_utils/blob/master/utils.py

def plots(ims, figsize=(20,10), rows=5, interp=False, titles=None): # 12,6
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

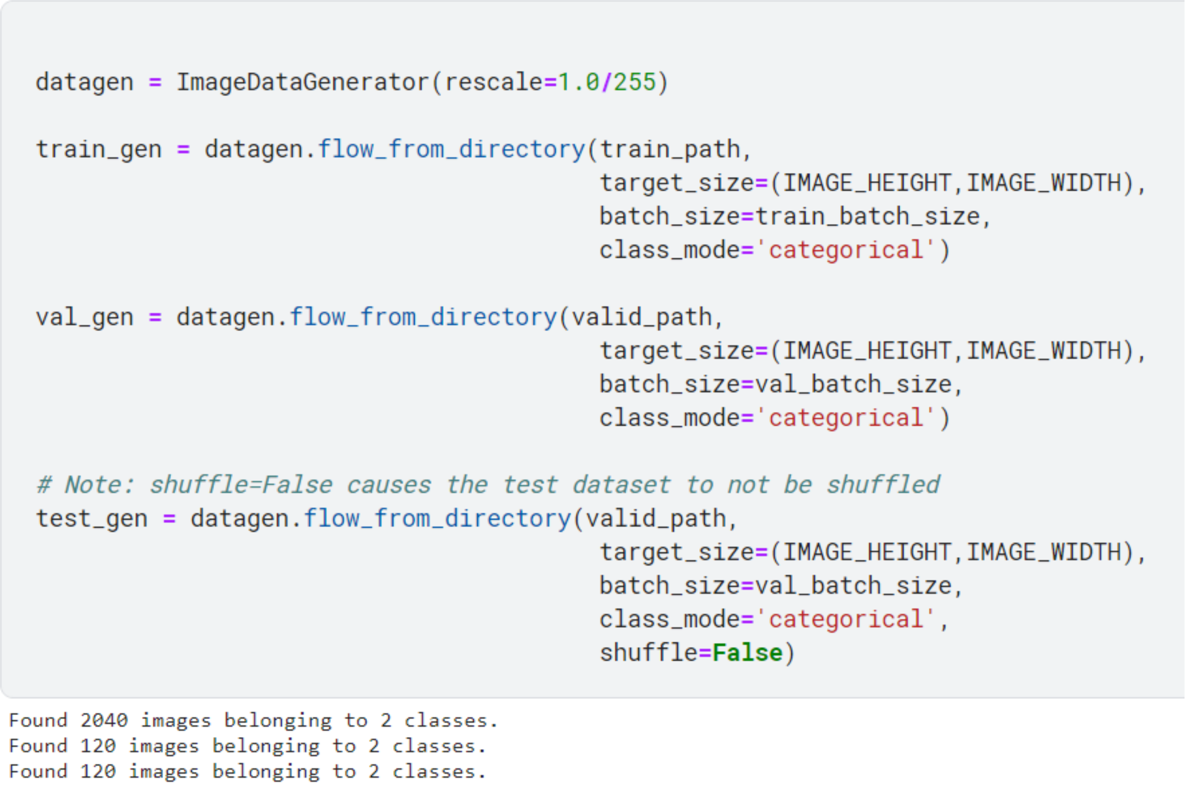


```
plots(imgs, titles=None) # titles=labels will display the image labels
```

Set up the Generators:



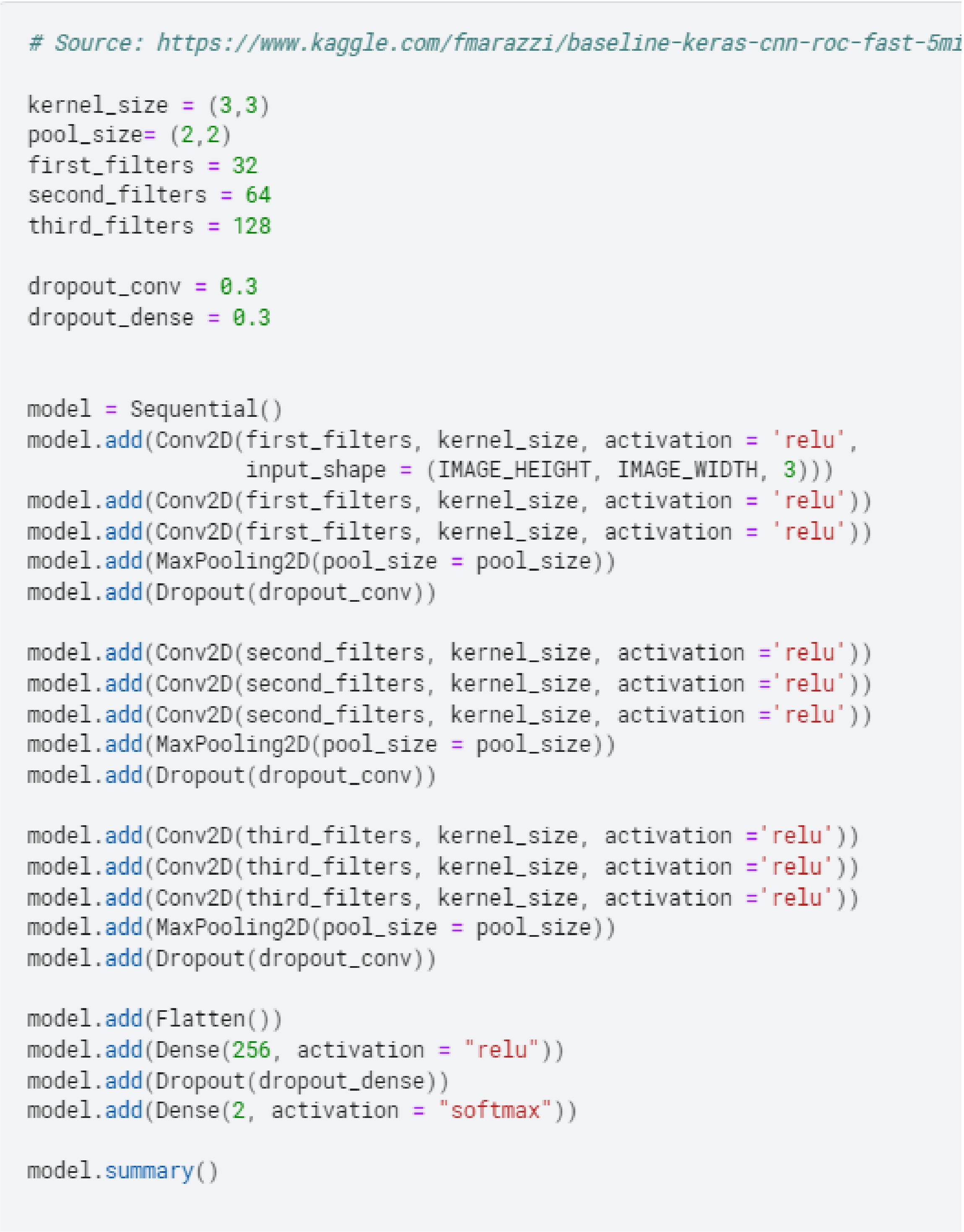
```
train_path = 'base_dir/train_dir'
valid_path = 'base_dir/val_dir'

num_train_samples = len(df_train)
num_val_samples = len(df_val)
train_batch_size = 10
val_batch_size = 10


train_steps = np.ceil(num_train_samples / train_batch_size)
val_steps = np.ceil(num_val_samples / val_batch_size)
```



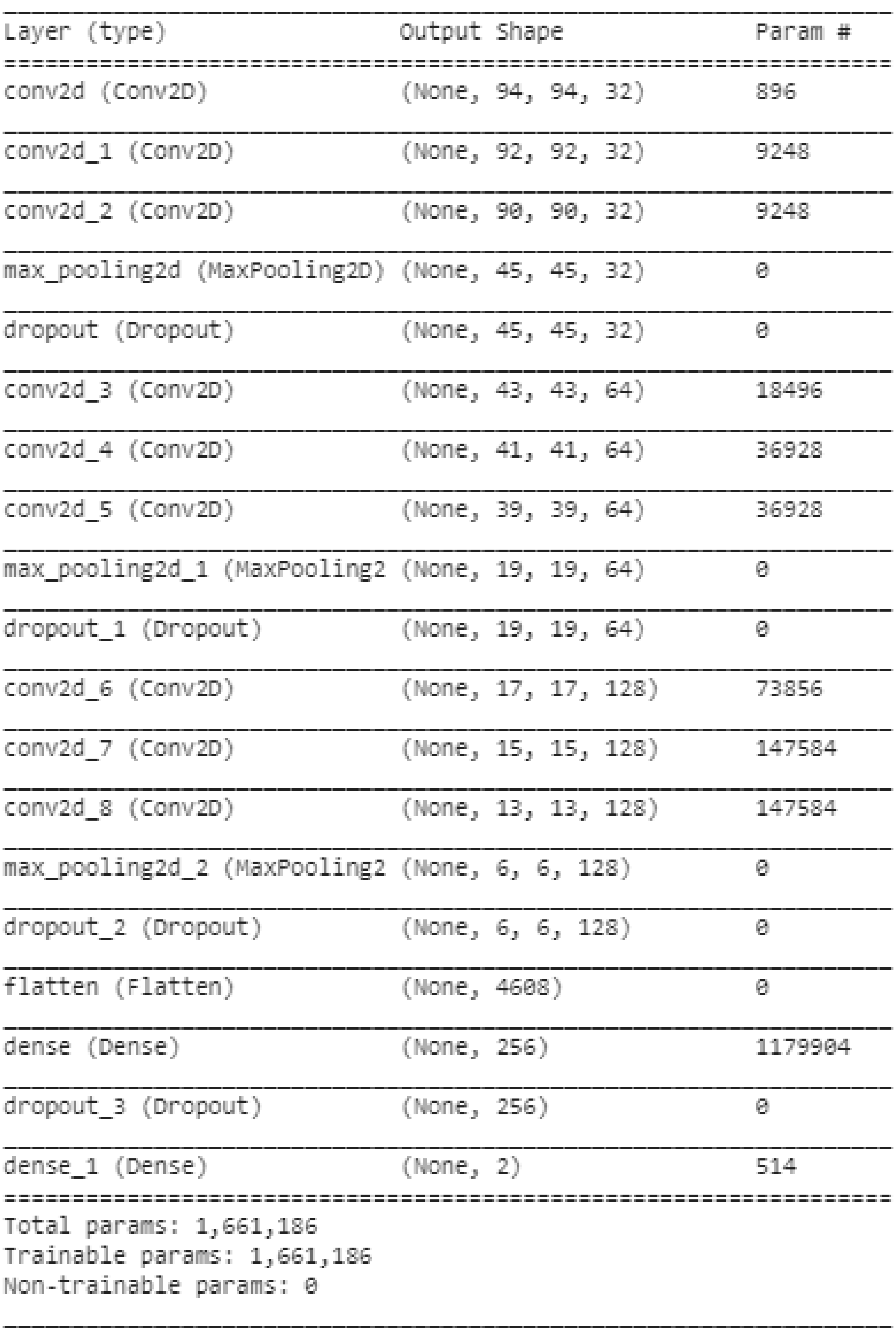
```
datagen = ImageDataGenerator(rescale=1.0/255)

train_gen = datagen.flow_from_directory(train_path,
                                        target_size=(IMAGE_HEIGHT,IMAGE_WIDTH),
                                        batch_size=train_batch_size,
                                        class_mode='categorical')

val_gen = datagen.flow_from_directory(valid_path,
                                        target_size=(IMAGE_HEIGHT,IMAGE_WIDTH),
                                        batch_size=val_batch_size,
                                        class_mode='categorical')

# Note: shuffle=False causes the test dataset to not be shuffled
test_gen = datagen.flow_from_directory(valid_path,
                                        target_size=(IMAGE_HEIGHT,IMAGE_WIDTH),
                                        batch_size=val_batch_size,
                                        class_mode='categorical',
                                        shuffle=False)
```

```
Found 2040 images belonging to 2 classes.
Found 120 images belonging to 2 classes.
Found 120 images belonging to 2 classes.
```

Create the model Architecture:

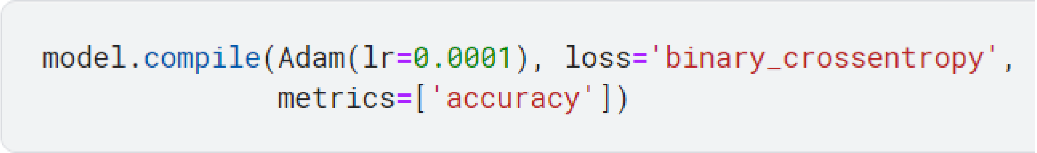


```
# Source: https://www.kaggle.com/fmarazzi/baseline-keras-cnn-roc-fast-5mi

kernel_size = (3,3)
pool_size= (2,2)
first_filters = 32
second_filters = 64
third_filters = 128

dropout_conv = 0.3
dropout_dense = 0.3


model = Sequential()
model.add(Conv2D(first_filters, kernel_size, activation = 'relu',
                 input_shape = (IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(dropout_dense))
model.add(Dense(2, activation = "softmax"))

model.summary()
```

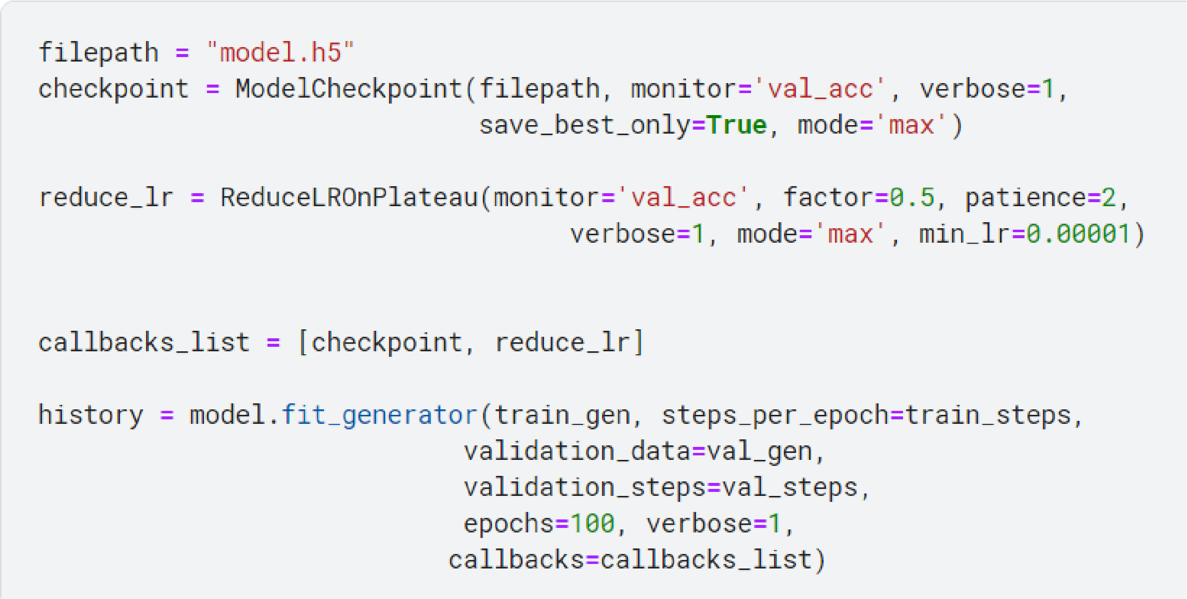


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 94, 94, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 92, 92, 32)        9248
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 90, 90, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 45, 45, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 43, 43, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 41, 41, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 39, 39, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 17, 17, 128)       73856
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1179904
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 1,661,186
Trainable params: 1,661,186
Non-trainable params: 0
_________________________________________________________________
```

Train the model:



```python
model.compile(Adam(lr=0.0001), loss='binary_crossentropy',
              metrics=['accuracy'])
```



```python
filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=2,
                                   verbose=1, mode='max', min_lr=0.00001)


callbacks_list = [checkpoint, reduce_lr]

history = model.fit_generator(train_gen, steps_per_epoch=train_steps,
                            validation_data=val_gen,
                            validation_steps=val_steps,
                            epochs=100, verbose=1,
                           callbacks=callbacks_list)
```

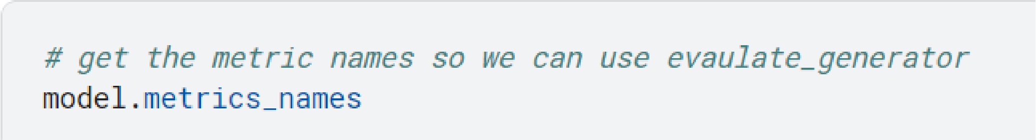
```
Epoch 1/100
67/68 [============================>.] - ETA: 0s - loss: 0.6946 - acc: 0.4851
Epoch 00001: val_acc improved from -inf to 0.49167, saving model to model.h5
68/68 [==============================] - 57s 834ms/step - loss: 0.6946 - acc: 0.4868 - val_loss: 0.
6932 - val_acc: 0.4917
Epoch 2/100
67/68 [============================>.] - ETA: 0s - loss: 0.6919 - acc: 0.5313
Epoch 00002: val_acc improved from 0.49167 to 0.50833, saving model to model.h5
68/68 [==============================] - 52s 771ms/step - loss: 0.6923 - acc: 0.5294 - val_loss: 0.
6926 - val_acc: 0.5083
Epoch 3/100
67/68 [============================>.] - ETA: 0s - loss: 0.6919 - acc: 0.5284
Epoch 00003: val_acc did not improve from 0.50833
68/68 [==============================] - 52s 771ms/step - loss: 0.6917 - acc: 0.5324 - val_loss: 0.
6907 - val_acc: 0.5000
Epoch 4/100
67/68 [============================>.] - ETA: 0s - loss: 0.6875 - acc: 0.5522
Epoch 00004: val_acc improved from 0.50833 to 0.61667, saving model to model.h5
68/68 [==============================] - 52s 771ms/step - loss: 0.6863 - acc: 0.5544 - val_loss: 0.
6692 - val_acc: 0.6167
Epoch 5/100
67/68 [============================>.] - ETA: 0s - loss: 0.6590 - acc: 0.6239
Epoch 00005: val_acc did not improve from 0.61667
68/68 [==============================] - 52s 770ms/step - loss: 0.6577 - acc: 0.6250 - val_loss: 0.
6469 - val_acc: 0.6083
Epoch 6/100
67/68 [============================>.] - ETA: 0s - loss: 0.6524 - acc: 0.6448
Epoch 00006: val_acc improved from 0.61667 to 0.62500, saving model to model.h5
```

```
68/68 [==============================] - 52s 769ms/step - loss: 0.6532 - acc: 0.6441 - val_loss: 0.
6576 - val_acc: 0.6250
Epoch 7/100
67/68 [============================>.] - ETA: 0s - loss: 0.6273 - acc: 0.6642
Epoch 00007: val_acc improved from 0.62500 to 0.70833, saving model to model.h5
68/68 [==============================] - 71s 1s/step - loss: 0.6285 - acc: 0.6632 - val_loss: 0.6170
- val_acc: 0.7083
Epoch 8/100
67/68 [============================>.] - ETA: 0s - loss: 0.5852 - acc: 0.7119
Epoch 00008: val_acc improved from 0.70833 to 0.74167, saving model to model.h5
68/68 [==============================] - 55s 805ms/step - loss: 0.5850 - acc: 0.7103 - val_loss: 0.
5615 - val_acc: 0.7417
Epoch 9/100
67/68 [============================>.] - ETA: 0s - loss: 0.5881 - acc: 0.6925
Epoch 00009: val_acc did not improve from 0.74167
68/68 [==============================] - 54s 798ms/step - loss: 0.5871 - acc: 0.6926 - val_loss: 0.
5853 - val_acc: 0.6500
Epoch 10/100
67/68 [============================>.] - ETA: 0s - loss: 0.5687 - acc: 0.7030
Epoch 00010: val_acc improved from 0.74167 to 0.79167, saving model to model.h5
68/68 [==============================] - 54s 791ms/step - loss: 0.5691 - acc: 0.7029 - val_loss: 0.
5418 - val_acc: 0.7917
Epoch 11/100
67/68 [============================>.] - ETA: 0s - loss: 0.5320 - acc: 0.7388
Epoch 00011: val_acc did not improve from 0.79167
68/68 [==============================] - 54s 787ms/step - loss: 0.5352 - acc: 0.7382 - val_loss: 0.
5697 - val_acc: 0.7167
Epoch 12/100
67/68 [============================>.] - ETA: 0s - loss: 0.5748 - acc: 0.7194
Epoch 00012: val_acc did not improve from 0.79167

Epoch 00012: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
68/68 [==============================] - 54s 789ms/step - loss: 0.5721 - acc: 0.7206 - val_loss: 0.
5341 - val_acc: 0.7750
Epoch 13/100
67/68 [============================>.] - ETA: 0s - loss: 0.5200 - acc: 0.7418
Epoch 00013: val_acc improved from 0.79167 to 0.79167, saving model to model.h5
68/68 [==============================] - 54s 792ms/step - loss: 0.5143 - acc: 0.7456 - val_loss: 0.
4933 - val_acc: 0.7917
Epoch 14/100
67/68 [============================>.] - ETA: 0s - loss: 0.5420 - acc: 0.7254
Epoch 00014: val_acc did not improve from 0.79167

Epoch 00014: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
68/68 [==============================] - 53s 785ms/step - loss: 0.5380 - acc: 0.7279 - val_loss: 0.
5031 - val_acc: 0.7583
Epoch 15/100
67/68 [============================>.] - ETA: 0s - loss: 0.5168 - acc: 0.7463
Epoch 00015: val_acc did not improve from 0.79167
68/68 [==============================] - 53s 783ms/step - loss: 0.5213 - acc: 0.7412 - val_loss: 0.
4919 - val_acc: 0.7917
Epoch 16/100
67/68 [============================>.] - ETA: 0s - loss: 0.5209 - acc: 0.7403
Epoch 00016: val_acc did not improve from 0.79167

Epoch 00016: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.
68/68 [==============================] - 53s 781ms/step - loss: 0.5223 - acc: 0.7397 - val_loss: 0.
5409 - val_acc: 0.7250
Epoch 17/100
```

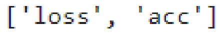
```
67/68 [============================>.] - ETA: 0s - loss: 0.4765 - acc: 0.7761
Epoch 00017: val_acc improved from 0.79167 to 0.80833, saving model to model.h5
68/68 [==============================] - 54s 789ms/step - loss: 0.4735 - acc: 0.7794 - val_loss: 0.
4812 - val_acc: 0.8083
Epoch 18/100
67/68 [============================>.] - ETA: 0s - loss: 0.5169 - acc: 0.7493
Epoch 00018: val_acc did not improve from 0.80833
68/68 [==============================] - 59s 874ms/step - loss: 0.5171 - acc: 0.7500 - val_loss: 0.
4991 - val_acc: 0.7917
Epoch 19/100
67/68 [============================>.] - ETA: 0s - loss: 0.5141 - acc: 0.7522
Epoch 00019: val_acc did not improve from 0.80833

Epoch 00019: ReduceLROnPlateau reducing learning rate to 1e-05.
68/68 [==============================] - 71s 1s/step - loss: 0.5152 - acc: 0.7500 - val_loss: 0.5069
- val_acc: 0.7917
Epoch 20/100
67/68 [============================>.] - ETA: 0s - loss: 0.4841 - acc: 0.7925
Epoch 00020: val_acc did not improve from 0.80833
68/68 [==============================] - 55s 806ms/step - loss: 0.4877 - acc: 0.7897 - val_loss: 0.
4903 - val_acc: 0.7917
Epoch 21/100
67/68 [============================>.] - ETA: 0s - loss: 0.4993 - acc: 0.7552
Epoch 00021: val_acc did not improve from 0.80833
68/68 [==============================] - 54s 793ms/step - loss: 0.5014 - acc: 0.7544 - val_loss: 0.
4960 - val_acc: 0.8000
Epoch 22/100
67/68 [============================>.] - ETA: 0s - loss: 0.5277 - acc: 0.7254
Epoch 00022: val_acc did not improve from 0.80833
68/68 [==============================] - 54s 797ms/step - loss: 0.5273 - acc: 0.7265 - val_loss: 0.
5029 - val_acc: 0.7917
Epoch 23/100
67/68 [============================>.] - ETA: 0s - loss: 0.4705 - acc: 0.7836
Epoch 00023: val_acc did not improve from 0.80833
68/68 [==============================] - 53s 785ms/step - loss: 0.4756 - acc: 0.7824 - val_loss: 0.
4783 - val_acc: 0.7917
Epoch 24/100
67/68 [============================>.] - ETA: 0s - loss: 0.4896 - acc: 0.7806
Epoch 00024: val_acc did not improve from 0.80833
68/68 [==============================] - 54s 791ms/step - loss: 0.4939 - acc: 0.7779 - val_loss: 0.
4800 - val_acc: 0.7917
Epoch 25/100
67/68 [============================>.] - ETA: 0s - loss: 0.4857 - acc: 0.7597
Epoch 00025: val_acc did not improve from 0.80833
68/68 [==============================] - 54s 793ms/step - loss: 0.4845 - acc: 0.7588 - val_loss: 0.
4887 - val_acc: 0.8000
Epoch 26/100
67/68 [============================>.] - ETA: 0s - loss: 0.4923 - acc: 0.7597
Epoch 00026: val_acc did not improve from 0.80833
68/68 [==============================] - 53s 786ms/step - loss: 0.4897 - acc: 0.7618 - val_loss: 0.
4981 - val_acc: 0.7833
Epoch 27/100
67/68 [============================>.] - ETA: 0s - loss: 0.5211 - acc: 0.7478
Epoch 00027: val_acc did not improve from 0.80833
68/68 [==============================] - 53s 785ms/step - loss: 0.5227 - acc: 0.7471 - val_loss: 0.
4884 - val_acc: 0.8000
Epoch 28/100
67/68 [============================>.] - ETA: 0s - loss: 0.4609 - acc: 0.7955
Epoch 00028: val_acc did not improve from 0.80833
```

```
68/68 [==============================] - 54s 791ms/step - loss: 0.4580 - acc: 0.7985 - val_loss: 0.
4796 - val_acc: 0.8000
Epoch 29/100
67/68 [============================>.] - ETA: 0s - loss: 0.5041 - acc: 0.7493
Epoch 00029: val_acc did not improve from 0.80833
68/68 [==============================] - 53s 784ms/step - loss: 0.5027 - acc: 0.7500 - val_loss: 0.
4966 - val_acc: 0.7833
Epoch 30/100
67/68 [============================>.] - ETA: 1s - loss: 0.5278 - acc: 0.7224
Epoch 00030: val_acc did not improve from 0.80833
68/68 [==============================] - 75s 1s/step - loss: 0.5312 - acc: 0.7206 - val_loss: 0.4859
- val_acc: 0.7833
Epoch 31/100
67/68 [============================>.] - ETA: 0s - loss: 0.4720 - acc: 0.7731
Epoch 00031: val_acc did not improve from 0.80833
68/68 [==============================] - 54s 795ms/step - loss: 0.4724 - acc: 0.7735 - val_loss: 0.
4912 - val_acc: 0.8000
Epoch 32/100
67/68 [============================>.] - ETA: 0s - loss: 0.4965 - acc: 0.7552
Epoch 00032: val_acc did not improve from 0.80833
68/68 [==============================] - 54s 793ms/step - loss: 0.4985 - acc: 0.7544 - val_loss: 0.
4949 - val_acc: 0.8000
Epoch 33/100
18/68 [======>.......................] - ETA: 36s - loss: 0.4598 - acc: 0.8000
```

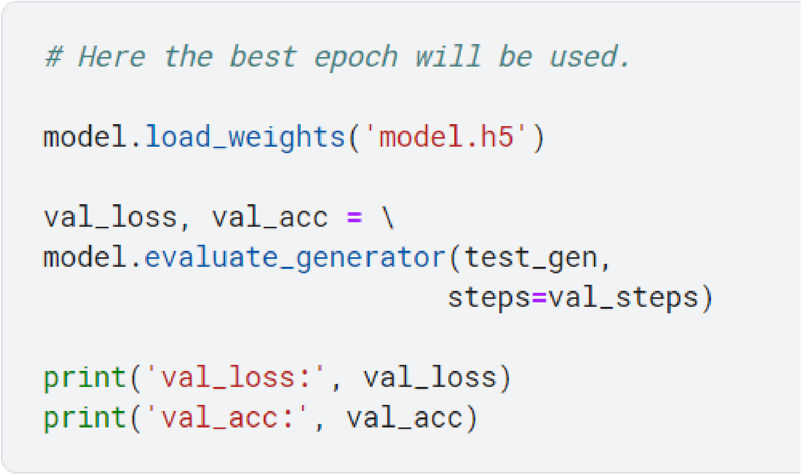
Evaluate the model set using the val set:



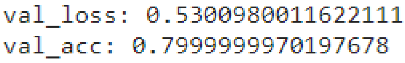
```
# get the metric names so we can use evaulate_generator
model.metrics_names
```



```
['loss', 'acc']
```



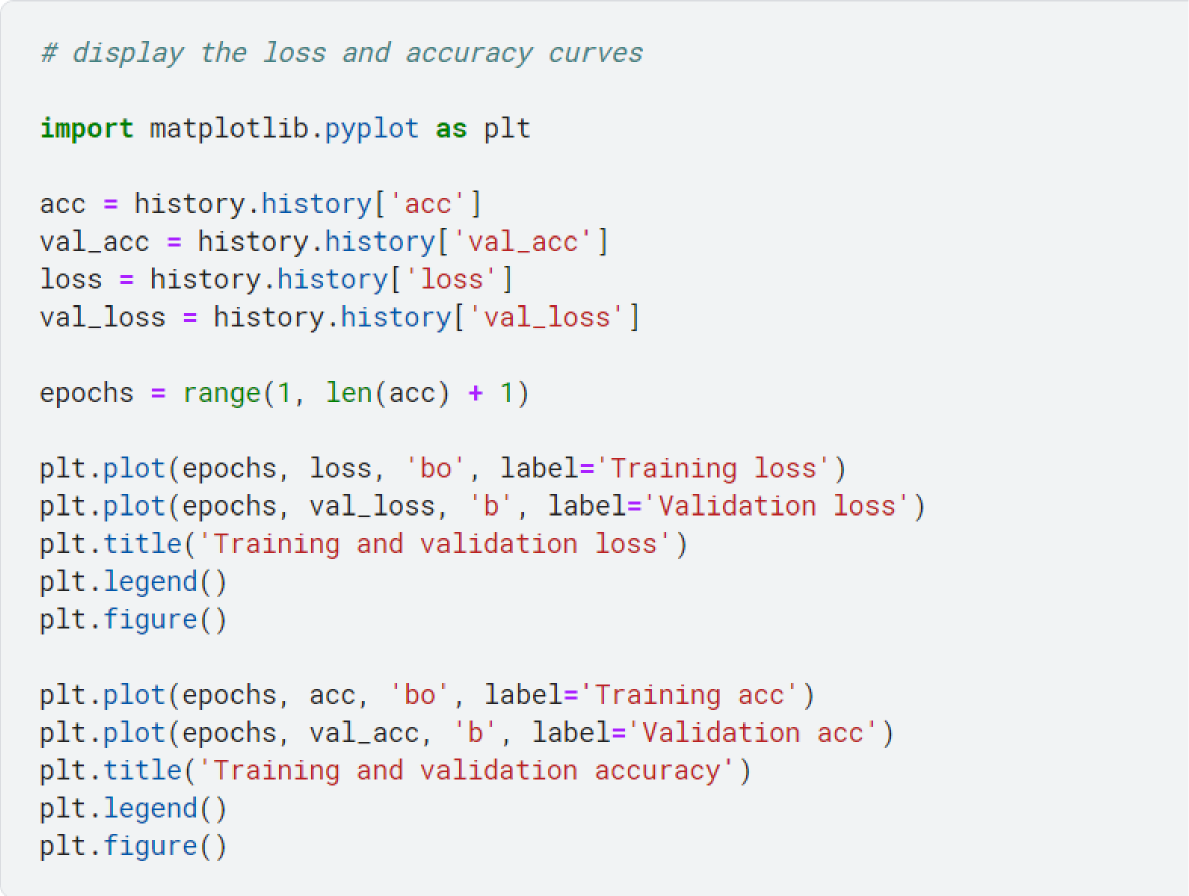
```
# Here the best epoch will be used.

model.load_weights('model.h5')

val_loss, val_acc = \
model.evaluate_generator(test_gen,
                        steps=val_steps)

print('val_loss:', val_loss)
print('val_acc:', val_acc)
```



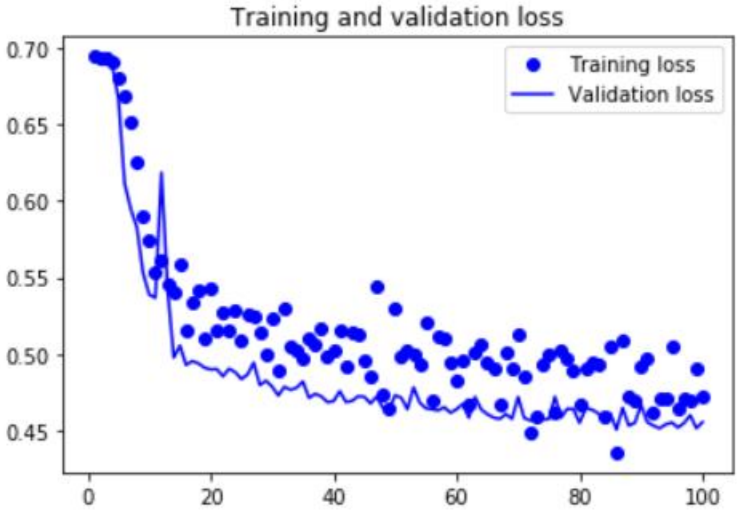
```
val_loss: 0.5300980011622111
val_acc: 0.7999999970197678
```

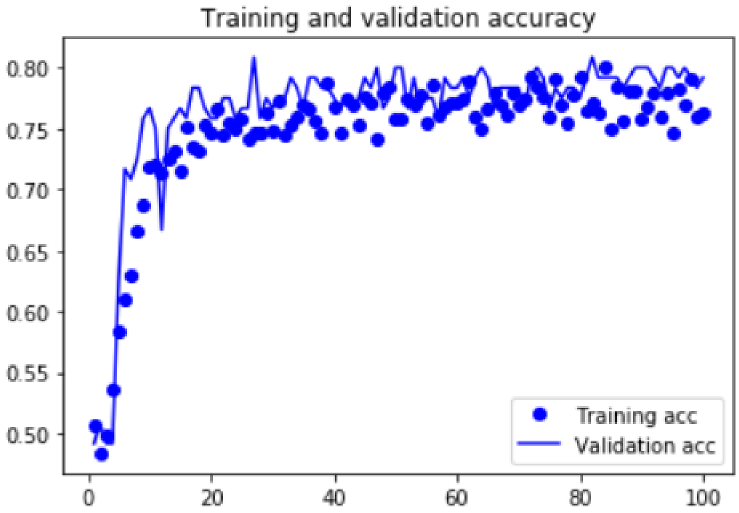
Loss: 53%
Accuracy: 79%

Plot the Training curve:

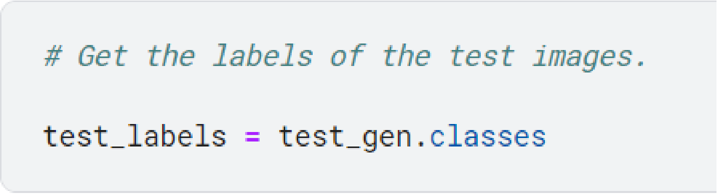


```
# display the loss and accuracy curves

import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
```

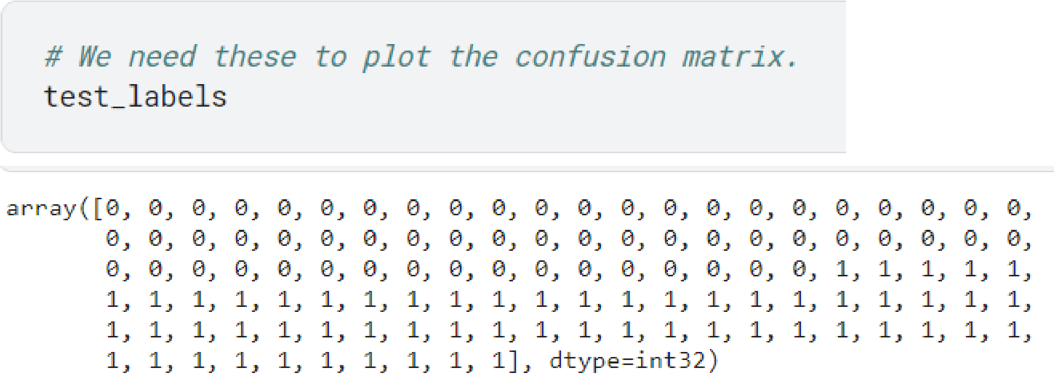




Create a confusion matrix:

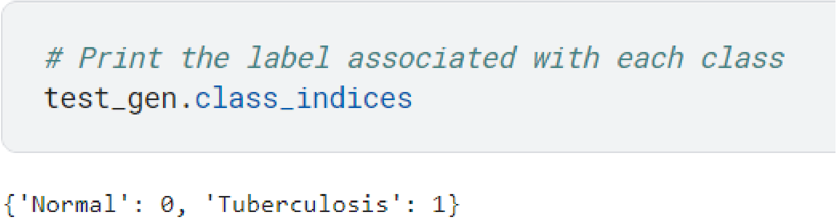


```
# Get the labels of the test images.

test_labels = test_gen.classes
```



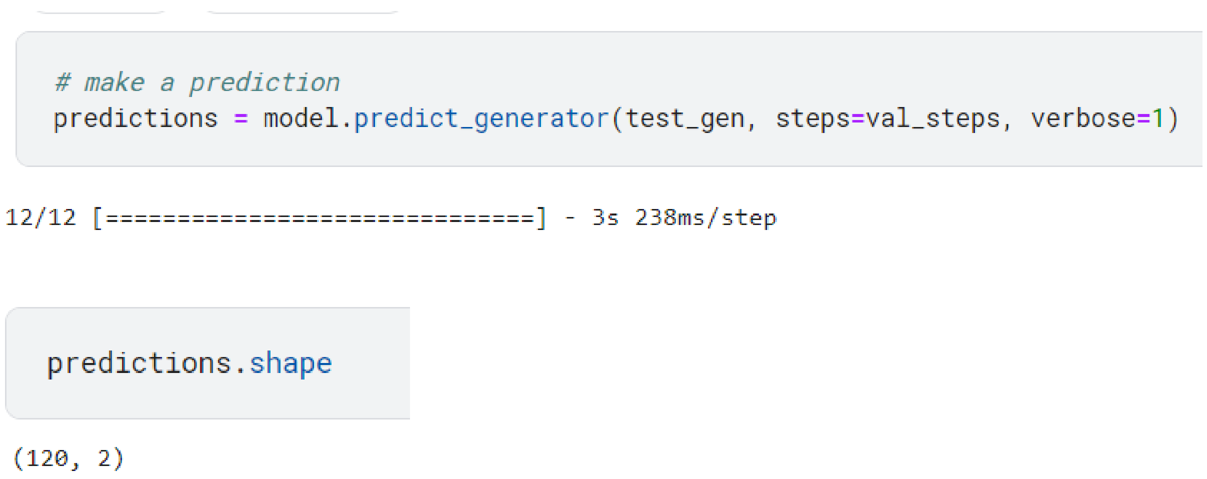
```
# We need these to plot the confusion matrix.
test_labels
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```



```
# Print the label associated with each class
test_gen.class_indices
```

```
{'Normal': 0, 'Tuberculosis': 1}
```

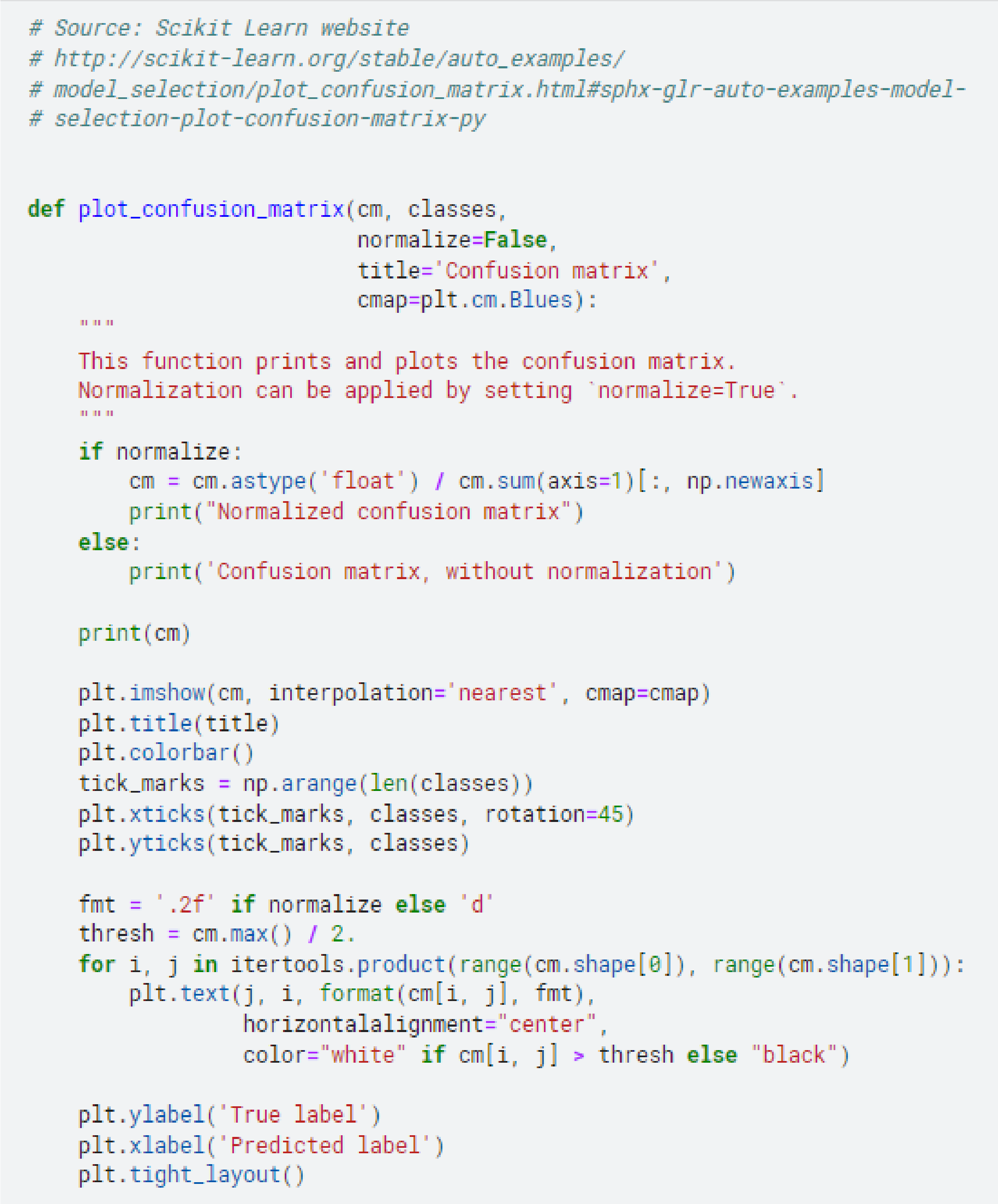


```
# make a prediction
predictions = model.predict_generator(test_gen, steps=val_steps, verbose=1)
```

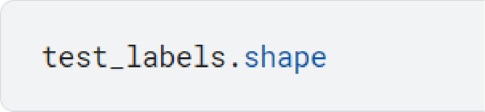
```
12/12 [==============================] - 3s 238ms/step
```

```
predictions.shape
```

```
(120, 2)
```

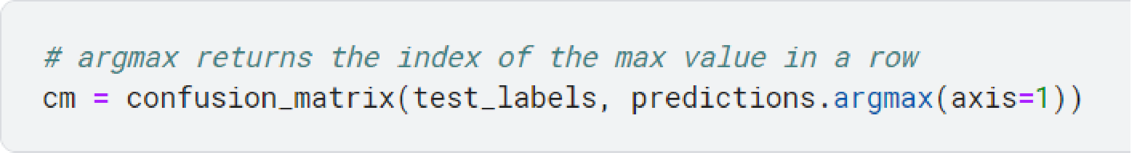


```python
# Source: Scikit Learn website
# http://scikit-learn.org/stable/auto_examples/
# model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-
# selection-plot-confusion-matrix-py


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

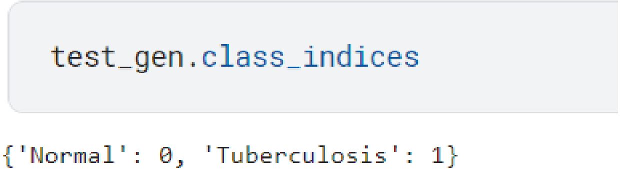


```python
test_labels.shape
```

```
(120,)
```

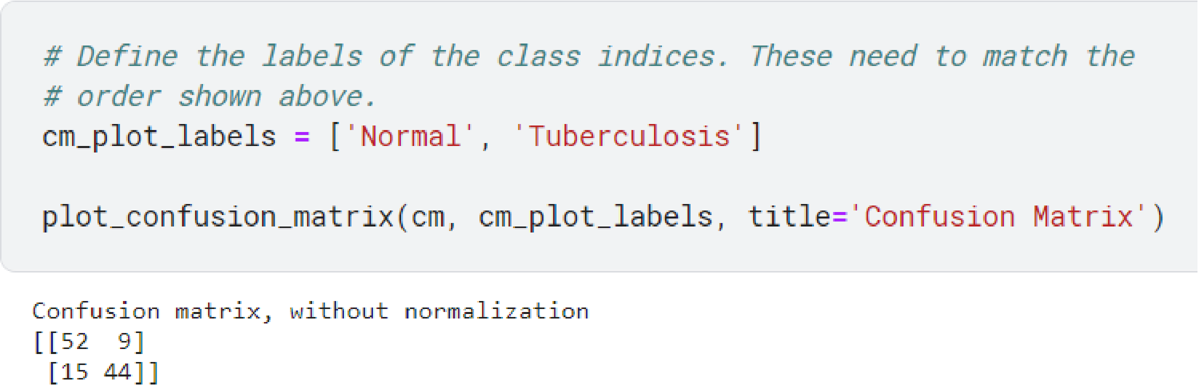


```
# argmax returns the index of the max value in a row
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))
```



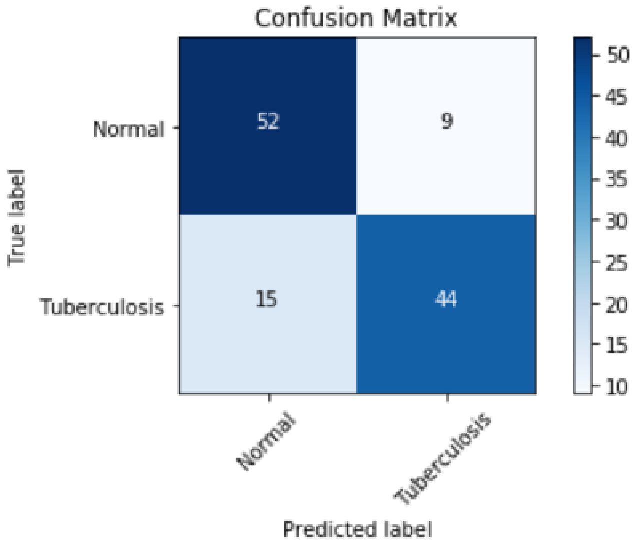
```
test_gen.class_indices
```

```
{'Normal': 0, 'Tuberculosis': 1}
```

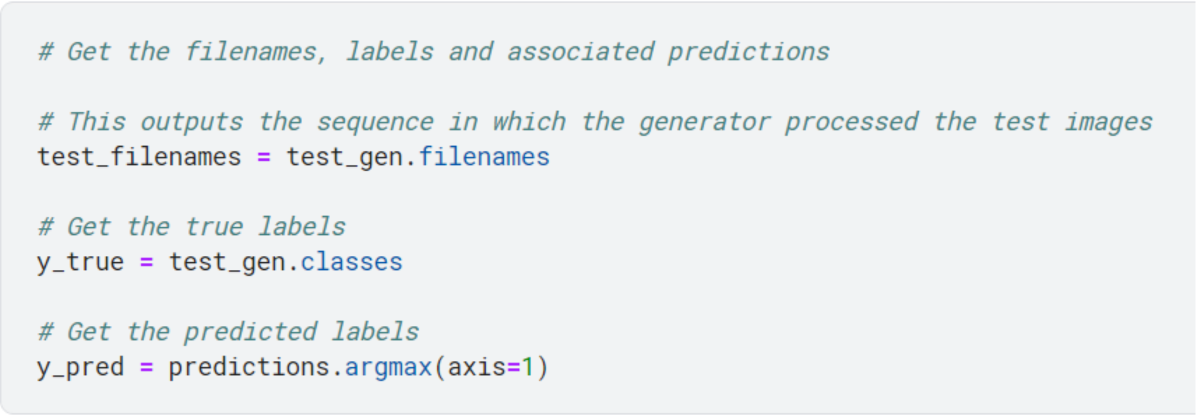


```
# Define the labels of the class indices. These need to match the
# order shown above.
cm_plot_labels = ['Normal', 'Tuberculosis']

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

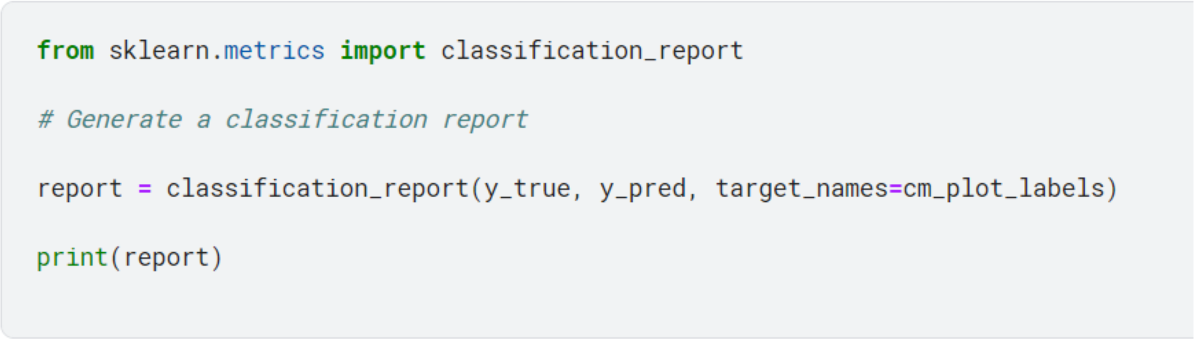
```
Confusion matrix, without normalization
[[52  9]
 [15 44]]
```



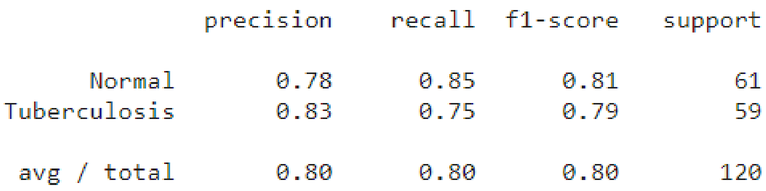
Create a classification report:



```
# Get the filenames, labels and associated predictions

# This outputs the sequence in which the generator processed the test images
test_filenames = test_gen.filenames

# Get the true labels
y_true = test_gen.classes

# Get the predicted labels
y_pred = predictions.argmax(axis=1)
```



```
from sklearn.metrics import classification_report

# Generate a classification report

report = classification_report(y_true, y_pred, target_names=cm_plot_labels)

print(report)
```



```
              precision    recall  f1-score   support

      Normal       0.78      0.85      0.81        61
Tuberculosis       0.83      0.75      0.79        59

 avg / total       0.80      0.80      0.80       120
```

ACHIEVED PRECISION: 80%

ACHIEVED RECALL: 80%

F1-SCORE: 80%

SUPPORT: 120

The dataset is quite small but by using a CNN and data augmentation The F1 score is
greater than 80%. From the confusion matrix we see that our model has a tendency to
classify TB images as Normal, more so than to classify Normal images as TB.

Reference link:

The full code of Kaggle can be accessed in this website.