

# Pneumonia Detection using Deep Convolutional Neural Networks

Group 6: Adish Pathare, Aditya Tirakannavar, Disha Revandkar, Rishi Sharma

## I. Task Definition, Evaluation Protocol, and Data

Pneumonia is a severe respiratory infection that affects millions of people around the world and is a leading cause of death, especially among young children and the elderly. Early diagnosis and treatment of pneumonia are critical for avoiding complications and lowering mortality rates. The analysis of chest X-ray images, which are commonly used in clinical practice to diagnose pneumonia, is one approach to early detection. Machine learning techniques, such as convolutional neural networks (CNNs), have shown great success in image recognition tasks, such as pneumonia detection from chest X-ray images, in recent years. The goal of this project is to accurately classify chest X-ray images as normal or positive for pneumonia, which could aid in the early detection and treatment of pneumonia [1].

The Chest X-Ray Images (Pneumonia) dataset [2] will be used in this project. The dataset contains a total of 5,856 chest X-ray images divided into training, testing, and validation sets. The image files for the posterior-anterior chest X-rays are in PNG format. The dataset is divided into two classes: normal and pneumonia-positive. The normal class contains chest X-ray images of patients who do not have pneumonia, while the pneumonia-positive class contains chest X-ray images of patients who do have pneumonia.

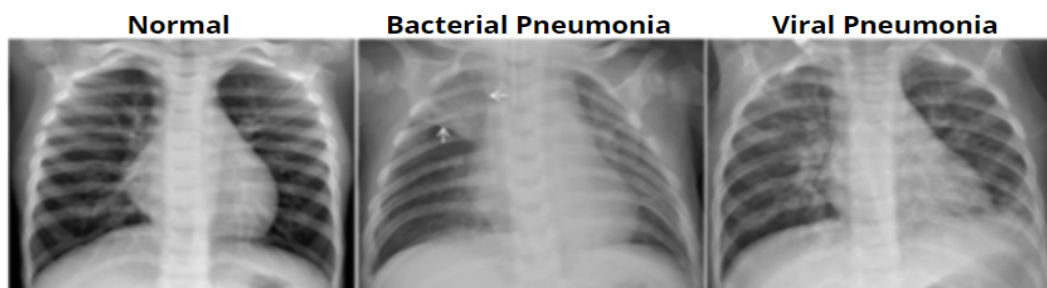


Figure 1: Sample Image from Chest X-Ray Images Dataset

A standard train-test split evaluation protocol is used. To assess its performance in classifying chest X-ray images as normal or pneumonia-positive, the CNN model is trained on the training set and tested on the testing set. During training, the validation set is used for hyperparameter tuning and model selection.

Several metrics are used to assess the performance of the CNN model. The percentage of correctly classified images out of all images in the testing set is the primary performance metric. Precision and recall are also calculated, which provide information about the proportion of true positive classifications out of all positive classifications and the proportion of true positive classifications out of all actual positive cases. Finally, the results are summarized using a

confusion matrix. Figure 2 illustrates the proposed flow diagram of the pneumonia detection model.

Using a deep convolutional neural network to predict pneumonia has the potential to improve lung disease diagnosis and treatment accuracy, resulting in better outcomes. By accurately predicting pneumonia conditions in patients, we can reduce the burden on healthcare systems. Finally, the goal of this project is to demonstrate the potential of deep learning techniques in healthcare and to contribute to ongoing efforts to improve the accuracy and efficiency of medical diagnosis.

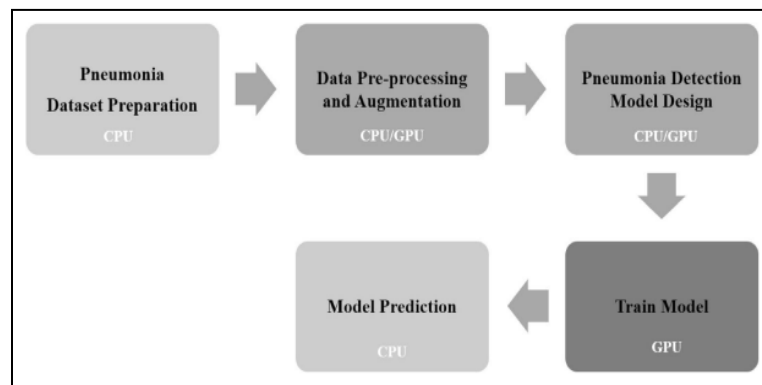


Figure 2: Flow diagram of proposed Pneumonia Detection Model

## II. Neural Network / Machine Learning Model

The neural network model used in the paper we referred to is a VGG19 model.[11] The model is trained by learning a set of filters that convolve over the input image to extract features at different scales. VGG19 architecture consists of 19 layers, including 16 convolutional layers and 3 fully connected layers[12]. The convolutional layers[13] use a small 3x3 filter size with a stride of 1, and the max pooling layers use a 2x2 filter with a stride of 2. The architecture uses a rectified linear unit (ReLU) activation function after each convolutional layer to introduce non-linearity. Max pooling layers are used to reduce the spatial dimensions of the feature maps while preserving their depth. The output of the last max pooling layer is flattened, which is then fed to 3 fully connected layers. Each fully connected layer is followed by a ReLU activation function and a dropout layer. Dropout is used to prevent overfitting and improve the generalization performance of the model. The final fully connected layer produces the class probability value between 0 and 1 using a softmax activation function representing the likelihood of the input image being pneumonia-positive or normal.

In our implementation we will be modifying the VGG19 model. The authors of the paper we referred to have originally used 16 layers, 13 convolutional layers and 3 fully connected layers. The model the authors used was more suitable for datasets with multiple classes. However, in the current dataset that we have there are only two classes, pneumonia-positive and normal. Therefore, the architecture was modified. Our model will have 9 convolutional layers, and the remaining layers will be left untrained to reduce computational complexity and improve training efficiency without sacrificing performance. Our model will have 4 custom dense layers, and a

sigmoid activation function layer used for binary classification. The model will be trained using binary cross-entropy loss, with the backpropagation algorithm used to update the weights. Figure 3 [14] in detail illustrates the working of our VGG19 architecture.

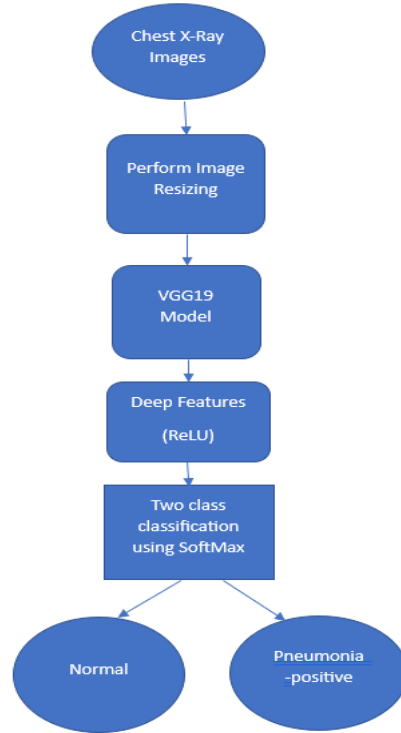


Figure 3: VGG19 Algorithm.

The loss metric used in training the model is binary cross-entropy loss. This loss function is commonly used for binary classification problems, where the goal is to predict a binary outcome. In our case it was predicting if the image is pneumonia-positive or normal. Binary cross-entropy loss measures the difference between the predicted probability and the true label, and the goal of the training process is to minimize this difference. The equation for binary cross-entropy loss is as follows:

$$L(y, \hat{y}) = -(y \log(\hat{y}) + (1-y) \log(1-\hat{y}))$$

where  $y$  is the true label (0 or 1),  $\hat{y}$  is the predicted probability, and  $\log$  is the natural logarithm.

During training, the model was fed batches of chest X-ray images along with the corresponding labels. The model then makes predictions on the images in the batch, and the binary cross-entropy loss is calculated based on the predictions and true labels. The weights of the model are then adjusted using backpropagation to minimize the loss. This process is repeated for multiple epochs until the model achieves convergence. The equations for the gradient of the binary cross-entropy loss with respect to the weights of the network are as follows:

$$dL/dw = (\hat{y} - y) * x$$

for weights between input and hidden layers

$$dL/dw = (\hat{y} - y) * \hat{y} * (1 - \hat{y}) * y_i$$

for weights between hidden and output layers

where  $w$  is the weight,  $x$  is the input,  $y_i$  is the output from the  $i$ -th neuron in the previous layer,  $\hat{y}$  is the predicted probability, and  $y$  is the true label.

We have made the following key modifications to the original model:

- **Data Augmentation:** We added data augmentation techniques such as rotation, flipping, and zooming to increase the diversity of the training dataset. This helped the model learn more robust features and generalize better to new data.
- **Model Architecture:** We have made modifications to the original model architecture to improve its performance by adding custom dense layers with a higher number of neurons to capture more complex features in the input images. We have also introduced dropout regularization to prevent overfitting and improve generalization performance. Dropout layers were added between the dense layers to reduce the likelihood of overfitting.
- **Transfer Learning:** We have used transfer learning to leverage the pre-trained weights of the VGG16 model and fine-tune them on our dataset. This helped reduce the training time and improve the overall performance of the model.
- **Hyperparameter Tuning:** We experimented with different hyperparameters such as learning rate, batch size, and optimizer to optimize the performance of the model.

### III. Experiment

Design:

Research Question	Can we find a model that can do better to detect Pneumonia using Chest X-rays images than previously used models?
Hypothesis	The VGG19 model with few modifications will perform better for detecting Pneumonia using Chest X-rays (Binary classification) than previously used algorithms.

Independent Variables	Hyperparameters - 16 convolutional layers, 3 fully connected layers, 5 MaxPool layers, dim(filter) = (height, Width, number of Channels). Pooling - Max pooling with 2x2 window and stride of 2. Model form - 16 convolutional layers followed by 3 fully connected layers, ReLU activation function in hidden layers, sigmoid activation function to classify model output to class. Total Parameters - 25,342,241 Trainable Params - 5,317,857 Non-Trainable Params - 20,024,384 Batch Size - 128 (For train-validation-test set)
Control Variables	Modeling Assumption - Nearby pixels are likely related, General features learned in lower layers.
Dependent Variables	Performance metric - Accuracy, Loss. Convergence intervals in the number of epochs - 13.

Table1 - Research Question, Hypothesis and Variables

#### Methodology:

Our code base is the pre-trained VGG19 module in tensorflow. The dataset contains 5,856 grayscale chest X-ray images in PNG format. There are 1,469 images in the "normal" class and 4,387 images in the "pneumonia-positive" class. The images are split into training, validation, and testing sets, with 5,216 images in the training set, 16 in the validation set, and 624 in the testing set.

We pre-processed the data using the ImageDataGenerator module in Keras. This was achieved by defining an ImageDataGenerator object for generating augmented images from the training data. These augmentations are applied to the original images to diversify the training set, thereby improving the model's ability to generalize new data. These augmented images are generated in batches. Similarly, ImageDataGenerator objects are designed for generating augmented images of the validation and test data.

Modifications include adding 3 custom Dense layers with ReLU activation function and a dense layer with sigmoid activation function. Another modification involves setting the last 7 convolutional layers of the architecture as frozen and not trainable. Further, we also added a dropout layer as a regularization technique to randomly set 40% of the input to the next layer as zero. This will be helpful to avoid overfitting.

We are not using the predefined dense layers of VGG19. We used the custom dense layers defined earlier. Moreover, we are not training our model with any new weights, instead we used the pre-trained VGG19 model weights.

## IV. Experimental Results and Discussion

### Running VGG19 on Training And Validation Set:

```
CSCI-635_Project_Code.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[ ] Total params: 25,342,241
Trainable params: 5,317,857
Non-trainable params: 20,024,384

[ ] model_1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
(variable) train_generator: DirectoryIterator
history = model_1.fit(train_generator, epochs = 13, validation_data = validation_generator)

Epoch 1/13
41/41 [=====] - 204s 5s/step - loss: 0.5884 - accuracy: 0.8309 - val_loss: 0.3097 - val_accuracy: 0.7500
Epoch 2/13
41/41 [=====] - 180s 4s/step - loss: 0.1836 - accuracy: 0.9222 - val_loss: 0.5756 - val_accuracy: 0.6875
Epoch 3/13
41/41 [=====] - 180s 4s/step - loss: 0.1450 - accuracy: 0.9444 - val_loss: 0.4728 - val_accuracy: 0.8125
Epoch 4/13
41/41 [=====] - 181s 4s/step - loss: 0.1230 - accuracy: 0.9521 - val_loss: 0.3885 - val_accuracy: 0.8125
Epoch 5/13
41/41 [=====] - 179s 4s/step - loss: 0.1172 - accuracy: 0.9540 - val_loss: 0.2415 - val_accuracy: 0.8750
Epoch 6/13
41/41 [=====] - 177s 4s/step - loss: 0.1172 - accuracy: 0.9555 - val_loss: 0.5747 - val_accuracy: 0.7500
Epoch 7/13
41/41 [=====] - 179s 4s/step - loss: 0.0949 - accuracy: 0.9636 - val_loss: 0.1938 - val_accuracy: 0.9375
Epoch 8/13
41/41 [=====] - 181s 4s/step - loss: 0.0850 - accuracy: 0.9695 - val_loss: 0.1999 - val_accuracy: 1.0000
Epoch 9/13
41/41 [=====] - 179s 4s/step - loss: 0.0942 - accuracy: 0.9643 - val_loss: 0.3017 - val_accuracy: 0.7500
Epoch 10/13
41/41 [=====] - 179s 4s/step - loss: 0.0983 - accuracy: 0.9611 - val_loss: 0.2924 - val_accuracy: 0.7500
Epoch 11/13
41/41 [=====] - 179s 4s/step - loss: 0.0754 - accuracy: 0.9739 - val_loss: 0.5275 - val_accuracy: 0.7500
Epoch 12/13
41/41 [=====] - 181s 4s/step - loss: 0.0780 - accuracy: 0.9693 - val_loss: 0.1903 - val_accuracy: 0.8750
Epoch 13/13
41/41 [=====] - 180s 4s/step - loss: 0.0883 - accuracy: 0.9651 - val_loss: 0.2177 - val_accuracy: 0.8750

[ ] # Saving the trained model
model_1.save("VGG19-7_trained.h5", overwrite=True)

0s completed at 9:09 PM
```

Figure 4 - Running VGG19 on training and validation set

Achieved training and maximum validation accuracy of 96.25%(approx) and 100% respectively, after running the model for 13 epochs.

We plotted the below lines plots based on the accuracy and loss values obtained from training.

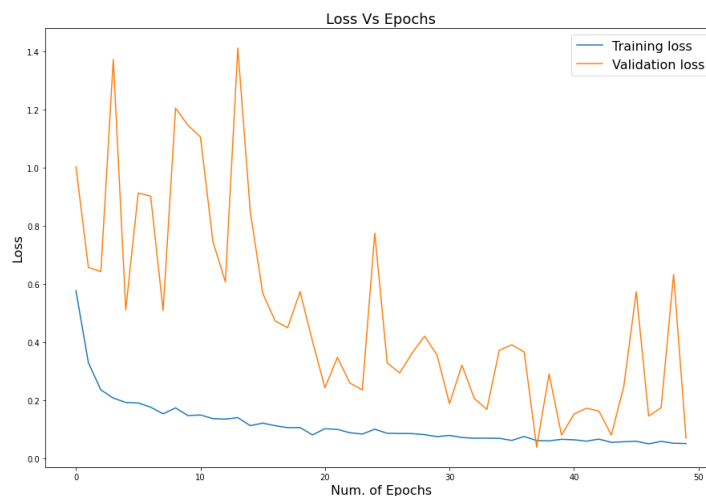


Figure 5 - Loss Vs Epochs plot

Figure 5 shows the trend of loss with respect to the number of epochs. We can observe that the loss value rapidly decreases with increase in the number of epochs initially but the change in loss value becomes minimal after a certain number of epochs. Training the model for more epochs after this limit may lead to overfitting.

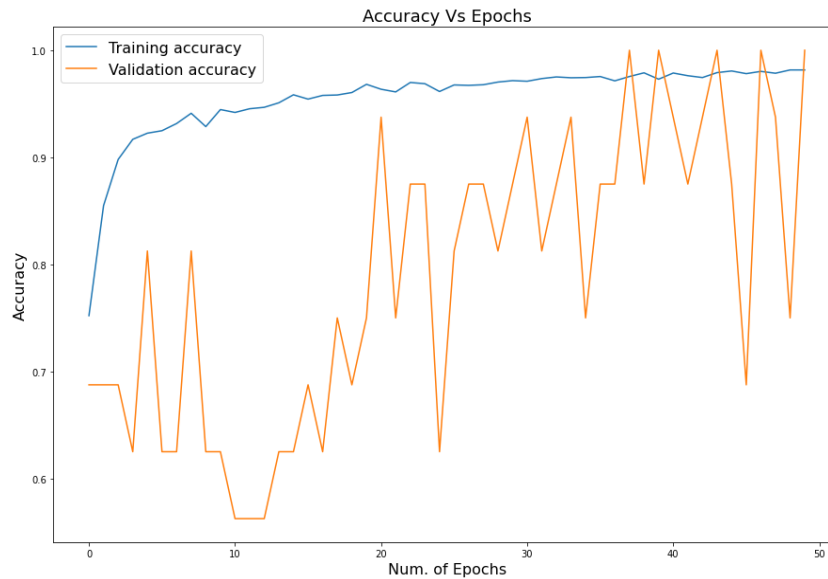


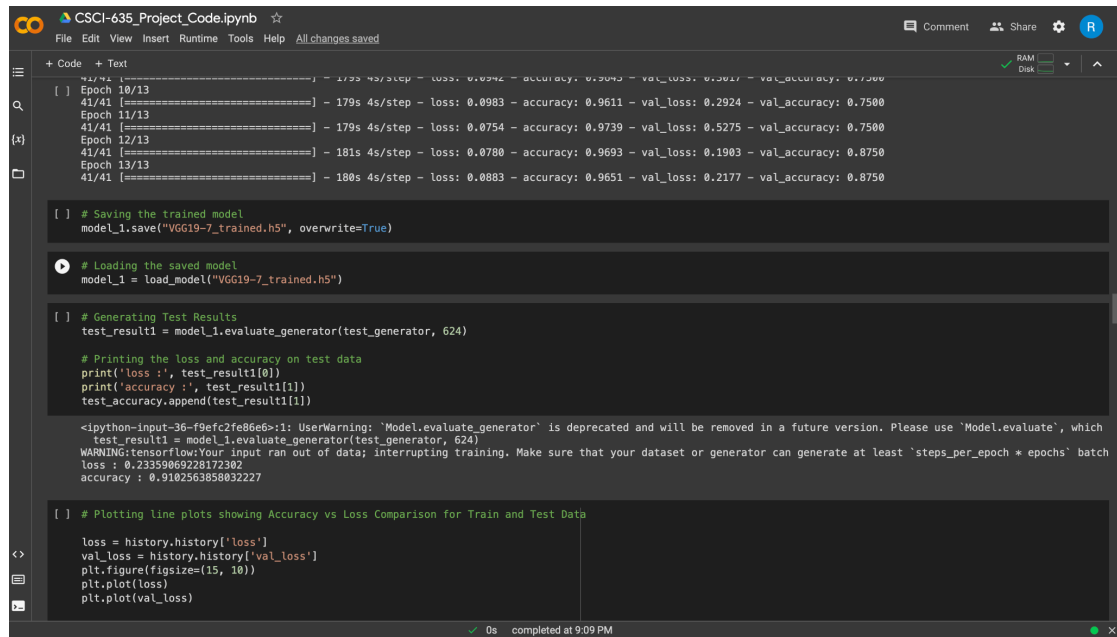
Figure 6 - Accuracy Vs Epochs plot

Figure 6 shows the trend of accuracy with respect to the number of epochs. We can similarly observe that the accuracy value rapidly increases with increase in the number of epochs initially, but the change in accuracy value becomes minimal after a certain number of epochs. More epochs may lead to overfitting.

We can also observe a zig-zag pattern in Validation loss and accuracy for both Fig 5 and Fig 6. That's because the Validation set consists of only 16 images (8 +ve and 8 -ve Pneumonia cases), and is primarily used for hyperparameter tuning.

The variation in Training loss and accuracy is less, and a trend can be observed, because of the large size of the train set, and the fact that our model learns the data well with every increasing epoch.

## Predicting Test Results:

The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar indicates the file is 'CSCI-635\_Project\_Code.ipynb'. The notebook contains several code cells. The first cell shows training progress for epochs 10, 11, 12, and 13, with metrics like loss, accuracy, val\_loss, and val\_accuracy. The second cell saves the trained model as 'VGG19-7\_trained.h5'. The third cell loads the saved model. The fourth cell generates test results using 'model.evaluate\_generator', showing a test loss of approximately 0.23 and a test accuracy of approximately 0.91. The fifth cell plots the training and validation loss and accuracy over time. The bottom status bar shows '0s completed at 9:09 PM'.

```
CSCI-635_Project_Code.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
[ ] Epoch 10/13
41/41 [=====] - 179s 4s/step - loss: 0.0983 - accuracy: 0.9611 - val_loss: 0.2924 - val_accuracy: 0.7500
Epoch 11/13
41/41 [=====] - 179s 4s/step - loss: 0.0754 - accuracy: 0.9739 - val_loss: 0.5275 - val_accuracy: 0.7500
Epoch 12/13
41/41 [=====] - 181s 4s/step - loss: 0.0780 - accuracy: 0.9693 - val_loss: 0.1903 - val_accuracy: 0.8750
Epoch 13/13
41/41 [=====] - 180s 4s/step - loss: 0.0883 - accuracy: 0.9651 - val_loss: 0.2177 - val_accuracy: 0.8750

[ ] # Saving the trained model
model_1.save("VGG19-7_trained.h5", overwrite=True)

[ ] # Loading the saved model
model_1 = load_model("VGG19-7_trained.h5")

[ ] # Generating Test Results
test_result1 = model_1.evaluate_generator(test_generator, 624)

# Printing the loss and accuracy on test data
print('loss :', test_result1[0])
print('accuracy :', test_result1[1])
test_accuracy.append(test_result1[1])

<ipython-input-36-f9efc2fe86e6>:1: UserWarning: 'Model.evaluate_generator' is deprecated and will be removed in a future version. Please use 'Model.evaluate', which
test_result1 = model_1.evaluate_generator(test_generator, 624)
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least 'steps_per_epoch * epochs' batch
loss : 0.23359069228172302
accuracy : 0.9102563858032227

[ ] # Plotting line plots showing Accuracy vs Loss Comparison for Train and Test Data

loss = history.history['loss']
val_loss = history.history['val_loss']
plt.figure(figsize=(15, 10))
plt.plot(loss)
plt.plot(val_loss)
```

Figure 7 - Predicting Test results

Our model achieved a prediction accuracy of 91%, with a loss value of 0.23 (approx) on the test data.

Below table shows the performance of other models we used for comparison:

Model	Training Accuracy	Validation Accuracy	Test Accuracy
ResNet50	86.08%	81.25%	77.24%
InceptionV3	97.97%	93.75%	89.26%

Table 2 - Performance comparison with ResNet50 and InceptionV3

## Model Comparisons:

We have selected accuracy as the base metric to decide which model performs better.



Figure 8 depicts the comparison between all the models used, based on achieved Test Accuracy:

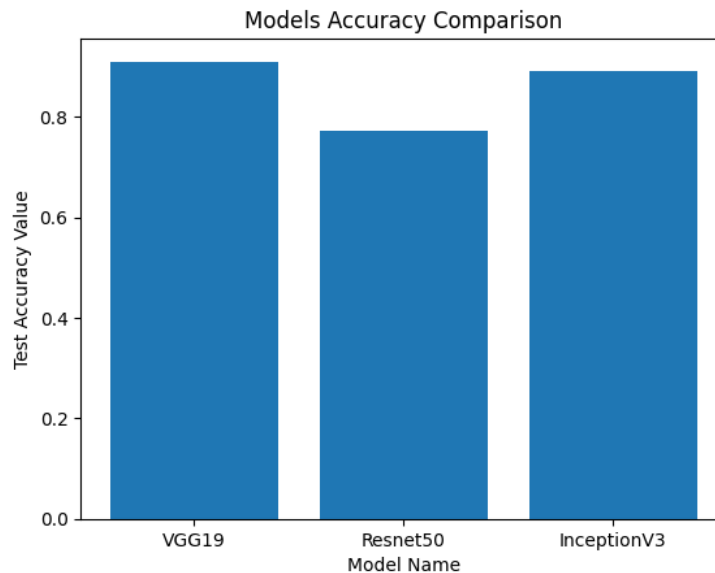


Figure 8 - Models accuracy comparison

We can observe that the VGG19 model performed better than the other models, which was also confirmed when we trained and tested our models multiple times, with VGG19 achieving around 93% accuracy (at max), which outperforms both the other models.

Hence, it confirms our Hypothesis that “VGG19 model with few modifications will perform better for detecting Pneumonia using Chest X-rays (Binary classification) than previously used algorithms.”, based on the comparison results.

Given that our Hypothesis is confirmed, we can now answer our research question that “Yes, we were able to find a model that performs better (based on accuracy) than other previously used state of the art algorithms for Pneumonia Detection using Chest X-rays Images”

#### Key Points to Note:

Our comparison and finding are based on the model configurations we implemented. For now we have used

- Batch Size = 128 (For the the train-validation-test sets)
- Number of epochs = 13 (For all the models)
- And only a limited number of layers, along with freezing (not training) some layers in pre-trained models

All this was done because of the Resource Constraints in Google Colab. Increasing the batch size, adding model complexity or increasing epochs further than what we did, was resulting in Resource Exhausted Error on Colab. Maybe we could have achieved even better results using the changes mentioned, but that can be our future research question (i.e. if we can improve this model of ours).

## References

1. Rajasenbagam, T., Jeyanthi, S. & Pandian, J.A. Detection of pneumonia infection in lungs from chest X-ray images using deep convolutional neural network and content-based image retrieval techniques. *J Ambient Intell Human Comput* (2021). <https://doi.org/10.1007/s12652-021-03075-2>
2. <https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia>
3. S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.
4. P. Naveen and B. Diwan, "Pre-trained VGG-16 with CNN Architecture to classify X-Rays images into Normal or Pneumonia," 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2021, pp. 102-105, doi: 10.1109/ESCI50559.2021.9396997.
5. S. Singh, "Pneumonia Detection using Deep Learning," 2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), NaviMumbai, India, 2021, pp. 1-6, doi: 10.1109/ICNTE51185.2021.9487731.
6. L. Račić, T. Popović, S. Ćakić and S. Šandi, "Pneumonia Detection Using Deep Learning Based on Convolutional Neural Network," 2021 25th International Conference on Information Technology (IT), Zabljak, Montenegro, 2021, pp. 1-4, doi: 10.1109/IT51528.2021.9390137.
7. D. Saikrishna et al., "Pneumonia Detection Using Deep Learning Algorithms," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 282-287, doi: 10.1109/ICIEM51511.2021.9445310.
8. T. Boyadzhiev, S. Tsvetanov and S. Dimitrova, "Deep Learning Image Classification for Pneumonia Detection," 2022 29th International Conference on Systems, Signals and Image Processing (IWSSIP), Sofia, Bulgaria, 2022, pp. 1-3, doi: 10.1109/IWSSIP55020.2022.9854442.
9. M. Yaseliani, A. Z. Hamadani, A. I. Maghsoodi and A. Mosavi, "Pneumonia Detection Proposing a Hybrid Deep Convolutional Neural Network Based on Two Parallel Visual Geometry Group Architectures and Machine Learning Classifiers," in *IEEE Access*, vol. 10, pp. 62110-62128, 2022, doi: 10.1109/ACCESS.2022.3182498.
10. Lamia A, Fawaz A. Detection of Pneumonia Infection by Using Deep Learning on a Mobile Platform. *Comput Intell Neurosci*. 2022 Jul 30;2022:7925668. doi: 10.1155/2022/7925668. PMID: 35942467; PMCID: PMC9356824.
11. Rajpurkar, P., Irvin, J., Bagul, A., Ding, D., Duan, T., Mehta, H., ... & Ng, A. (2018). Mura dataset: Towards radiologist-level abnormality detection in musculoskeletal radiographs. *arXiv preprint arXiv:1712.06957*.<https://arxiv.org/abs/1712.06957>
12. Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3462-3471). <https://arxiv.org/abs/1705.02315>
13. Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H., Baxter, S. L., ... & Zhang, K. (2018). Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5), 1122-1131. <https://doi.org/10.1016/j.cell.2018.02.010>
14. Nilanjan Dey, Yu-Dong Zhang, V. Rajinikanth, R. Pugalenth, N. Sri Madhava Raja, Customized VGG19 Architecture for Pneumonia Detection in Chest X-Rays, *Pattern Recognition Letters*, Volume 143, 2021, Pages 67-74, ISSN 0167-8655, <https://doi.org/10.1016/j.patrec.2020.12.010>.