Comparative analysis of Pneumonia Detection using CNN, VGG16, and ResNet Models

Roop Sagar Mangineni
Department of Computer Science
Iowa State University
Ames, Iowa
ID-948529640
roopm@iastate.edu

Sohan Salahuddin Mugdho
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa
ID-939708824
ssmugdho@iastate.edu

Sri Harsha Mudumba
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa
ID-462988584
srim@iastate.edu

***Abstract*—Pneumonia is still a major global health concern, demanding the use of modern diagnostic methods for timely and accurate detection. Several pneumonia detection methods, however, make use of Machine Learning and Deep Learning technologies. This research compares the Convolutional Neural Network (CNN), VGG16, and ResNet50 models for pneumonia identification. The implementation entails training these deep learning models on a large dataset of chest X-ray pictures in order to assess their efficacy in detecting pneumonia. A comparison of CNN, VGG16, and ResNet50 was also undertaken to determine the most effective model for identifying Pneumonia. The goal is to improve the accuracy and reliability of pneumonia identification while also expanding the field of medical image analysis.This study seeks to contribute to the creation of sophisticated deep-learning models for critical healthcare applications by combining effective architectural design, hyperparameter optimization, and interpretability.**

***Index Terms*—Pneumonia, CNN, VGG, ResNet, Image-classification.**

I. Introduction

Pneumonia, a common respiratory infection, poses considerable public health issues worldwide. It is widely regarded as the leading cause of child death worldwide. Pneumonia kills around 1.4 million children each year, accounting for 18of children under the age of five. Pneumonia affects two billion people worldwide each year. However, early and correct detection of Pneumonia is critical for timely and successful treatment, improving patient outcomes, and lowering mortality rates.

Advances in artificial intelligence (AI) and deep learning, notably Convolutional Neural Networks (CNNs), have showed remarkable possibilities in medical picture processing, including the diagnosis of Pneumonia, in recent years. CNNs are deep learning models that are meant to learn hierarchical features from input data automatically and adaptively, making them particularly effective for image-related tasks. The convergence of healthcare and machine learning presents an exciting opportunity to improve diagnostic capabilities. This project delves into pneumonia detection using Convolutional Neural Networks (CNN), specifically focusing on our proposed customized CNN, VGG16 [1], and ResNet50 [2] architectures. We also examine existing studies and research that have used CNNs to identify pneumonia, emphasizing their techniques, performance, and prospective areas for improvement. By studying progress in this sector, we hope to contribute to ongoing research efforts to improve pneumonia diagnosis and, ultimately, patient care and outcomes.

The dataset used for this challenge was taken from Kaggle and contains a large number of chest X-ray pictures that have been categorized as pneumonia-positive or pneumonia-negative. Because the dataset is fully structured for supervised learning, it's an excellent resource for training and testing machine learning models. The machine learning issue at hand is binary classification: differentiating between images of normal chest X-rays and those that reveal Pneumonia. This work is clinically significant and highlights the potential of deep learning in medical picture analysis. This study examines the various CNN model architectures used to diagnose Pneumonia from X-ray pictures of the chest. We examine CNN design and operation, focusing on their ability to extract features and patterns from medical pictures. Furthermore, we look at how to train a CNN model, including data preprocessing, model selection, and optimization strategies, to attain the best pneumonia detection performance.

The VGG16 architecture, ResNet50V2 architecture, and our proprietary CNN architecture are the key machine-learning models under consideration. Both VGG16 and ResNet50V2 have been praised for their ability to perform image classification tasks. Using these models, we hope to accurately identify Pneumonia from normal instances. The potential benefits of this research extend beyond healthcare, where early and correct diagnosis can have a huge impact on patient outcomes, to the larger field of machine learning, where interpretable models are gaining popularity.

II. Related Works

Detecting pneumonia with chest X-rays has been an unsolved research topic for years. During COVID-19, the demand for research in Pneumonia increased to aid in diagnosis. There have been numerous previous studies in which the

authors attempted to diagnose pneumonia using cutting-edge machine-learning algorithms. Pneumonia detection has been studied using deep learning approaches such as CNN with a ReLU activation function and 8M parameters, with an accuracy of 88.9%. [3]. A strategy with a high AUC of 0.99 was proposed for classifying pulmonary tuberculosis utilizing two separate DCNNs, AlexNet and GoogleNet. [4]. The use of different architectures and assembly has also resulted in a considerable boost in accuracy to 98.91 [5].

An ensemble-learning strategy was utilized to detect Pneumonia in chest X-rays, in which the model was pre-trained on ImageNet to aid detection, and the models of three base CNN models, GoogLeNet, ResNet-18, and DenseNet-121, were used for ensembling. The ensemble achieved a 98.81% accuracy rate. [6] Four different pre-trained deep Convolutional Neural Networks (CNN) were used for transfer learning for detecting pneumonia images: AlexNet, ResNet18, DenseNet201, and SqueezeNet, with DenseNet201 producing the highest accuracy for both training and testing, with an accuracy of 98% and a precision of 97%.

[7]. Marco La Salvia and Gianmarco presented a new detection technique based on Lung Ultrasound Scores. ReLU activation was employed for residual CNNs based on resNet18(11M parameters) and resNet50(23M parameters) that had already been optimized using the ImageNet dataset. The restNet-based architectures achieved nearly 98.5 percent accuracy on test data by leveraging a wealth of domain-specific insights on Pneumonia from the medical industry to better grasp the nature of the images. [8]. Pneumonia detection utilizing convolutional neural networks and deep learning was accomplished using a dropout layer for efficient calculation, resulting in an accuracy of 97% in 122ms. [9]. It introduced a Hybrid Deep Convolutional Neural Network based on two parallel Visual Geometry Group Architectures and Machine Learning Classifiers. [10]. A deep learning model for the classification of COVID and Pneumonia using DenseNet201 was developed, with a training accuracy of 98% but a test accuracy of 80%; this paper strongly emphasized HIPAA compliance requirements. [9]. A Deep Learning-based model for diagnosing Pneumonia from Chest X-ray images was developed using VGG-16 and Neural Networks. [11]. Pediatric pneumonia identification was done using a machine learning approach on chest X-rays, and it employed the quadratic SVM model to provide an accuracy of 97.58%; the prospect of employing SVM showed tremendous promise because it helps identify the site spots, which enables the medical professional to provide concrete results. diagnosis [12]. Using transfer learning on top of upgraded CNN resulted in a 92.4 percent accuracy. [13]. CheXMed, a multimodal algorithm, was created for Pneumonia Detection in the Elderly, opening up a world of possibilities and reaching an AUC of 0.768 and an accuracy of 64.5%. [14]. Pneumonia was treated with many CNN variations, including Inception V3, VGG-16, ResNet, and DenseNet, with Inception V3 achieving a high accuracy of 90.26% [15]. The majority of solutions employed convolutional neural networks and their variants, demonstrating that these models have a high potential for detecting Pneumonia.

III. Methodology

As illustrated in Fig. 1, we reviewed the methods used for pneumonia detection using deep learning models, stressing several areas such as dataset collecting, data pre-processing, feature selection, hyper-parameter choices, and assessment metrics.

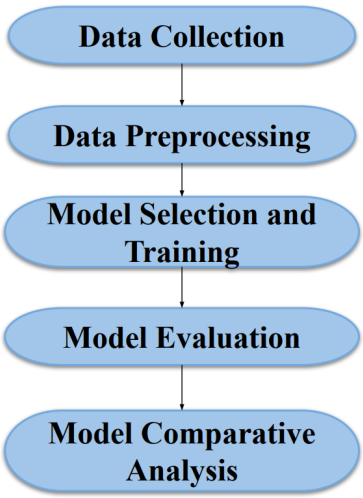

Fig. 1. Methodology

A. Dataset Collection

The study's dataset came from Kaggle and included 5,863 chest X-ray pictures. Expert radiologists analyzed and tagged these images to indicate the presence or absence of Pneumonia, as demonstrated in Fig 2 and Fig 3. The dataset contains a wide range of patient demographics, X-ray machine settings, and clinical circumstances.

B. Data Preprocessing

The chest X-ray pictures were preprocessed before training the deep-learning models. This section describes the techniques used for preprocessing, such as Gray Filtering, Image Normalization, Image Resizing, and Image Augmentation. These stages are critical for improving the model's capacity to learn meaningful features and patterns from the dataset.

1) Image Normalization: The pixel values across all photos were normalized. This step is critical for improving model convergence and performance by decreasing the impact of pixel intensity changes. Typically, normalization entails scaling pixel values to a defined range of 0 to 1.

2) Image Resizing: All photos were shrunk to a preset resolution to guarantee uniformity in model input dimensions and assist efficient computation. Resizing reduces fluctuations in image dimensions and helps the model acquire key features regardless of the original input size.

NORMAL

Fig. 2. Pneumonia Negative

PNEUMONIA

Fig. 3. Pneumonia Positive

3) Image Augmentation: **rotation, horizontal and vertical flipping, zooming, shearing and height, width, and channel shifting** were used to artificially improve the diversity of the training dataset. It prevents overfitting and improves the model's capacity to generalize to new data. The addition of variability to the training set simulates real-world conditions and improves the model's robustness.

C. Model Selection and Architecture

This section describes the selection and training of three independent deep-learning architectures - CNN, VGG16, and ResNet50 - for pneumonia identification. The models chosen cover a wide range of complexities, from a simple CNN model to more complex architectures like VGG16 and ResNet50. The training method includes optimizing the models using the preprocessed dataset to attain high accuracy and reliability in pneumonia classification.

1) CNN: CNNs are the core deep-learning models for image classification problems. The CNN architecture used in this study includes convolutional layers for feature extraction and pooling layers for downsampling. The final classification is facilitated by fully connected layers at the network's end. The architecture of the CNN model is depicted in Fig. 4

- **Input Shape**: The input images have a shape of (150, 150, 3).
- **Convolutional Layer:** The first convolutional layer*'conv2d 23'* detects basic features in the input image using 32 filters and a kernel size of (3, 3). *'conv2d 24'* is the second convolutional layer, with 64 filters and a kernel size of (3, 3). These layers can recognize more complicated patterns and features as one progresses deeper into the network. *'conv2d 25'* is the third convolutional layer, with 128 filters and a kernel size of (3,3). *'conv2d 26 '* the fourth convolutional layer with 256 filters and a kernel size of (3, 3).
- **Pooling Layers:** Max-pooling layers are *'max pooling2d23'*, *'max pooling2d24'*, *'max pooling2d25'*, and *'max pooling2d26'*. The spatial dimensions of the feature maps are reduced, while the most relevant information is retained using max-pooling layers. Because the pool size is (2, 2), the feature maps are downsampled by a factor of two in each dimension.
- **Dropout Layers:** It is a regularization strategy that prevents overfitting by randomly changing a fraction of input units to 0 during training updates. Dropout layers are *'dropout 16'* and *'dropout 17'*.
- **Flatten Layers:** *'flatten 5'* is a layer that converts 2D feature maps into 1D vectors, preparing the data for fully linked layers. Fully linked (Dense) Layers:
- **Fully Connected (Dense) Layers:** is a fully linked layer with 128 neurons. This layer is in charge of discovering complex patterns in data. *'dense 15'*, *'dense 16'*, and *'dense 17'* are the following completely connected layers, each containing 64, 32, and 1 neuron(s). For binary classification tasks, the final layer typically has a single neuron.

2) VGG16: VGG16 [1] is well-known for its deep design, which includes 16 weight layers and makes it appropriate for challenging image recognition problems. In this study, the VGG16 model uses numerous convolutional layers with modest filter sizes to catch subtle patterns in chest X-ray pictures. The CNN model's architecture is depicted in Fig. 5. For a comparative study, we have constructed the model with CNN parameters.

3) ResNet50: ResNet50 [2], a ResNet architectural version, is distinguished by its residual learning framework, which simplifies the training of exceptionally deep networks. This approach mitigates the vanishing gradient problem, allowing for effective deep neural network training. The CNN model's architecture is depicted in Fig. 6. For a comparative study, we have constructed the model with CNN parameters.

D. Training and Hyperparamters

The training approach includes fine-tuning the recommended deep learning models for pneumonia identification - CNN, VGG16, and ResNet50 - using a large dataset of preprocessed chest X-ray images. During backpropagation, the Adam optimizer, a prominent optimization algorithm, minimizes the loss functions of the models. This adaptive learning rate optimizer modifies the learning rates for each parameter separately, increasing convergence speed and training efficiency overall.

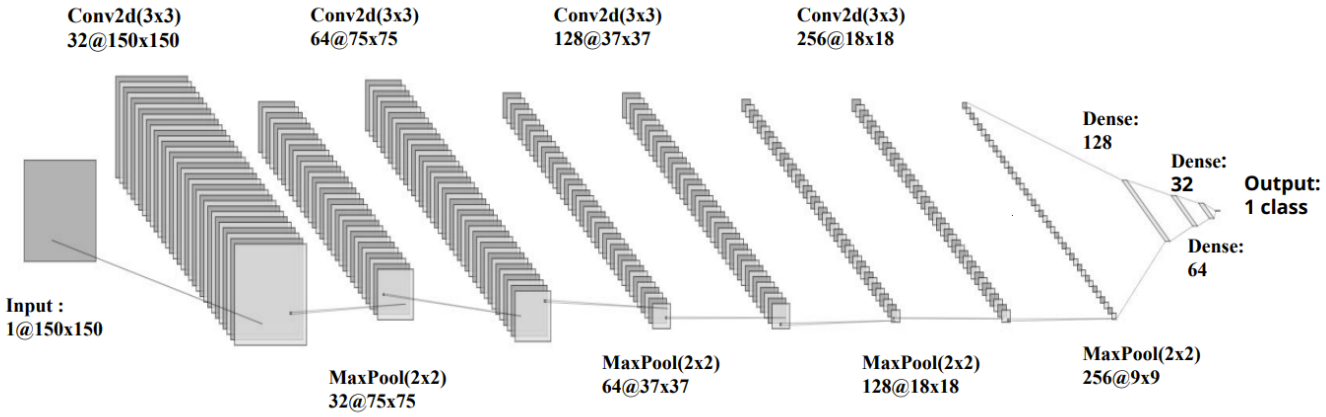

Fig. 4. CNN Architecture

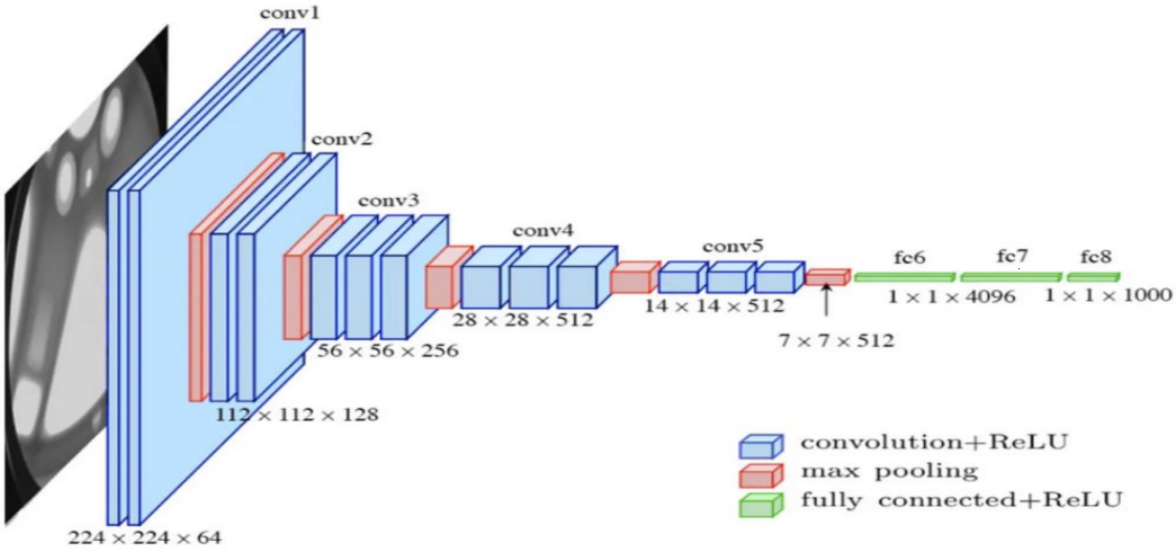

Fig. 5. VGG16 Architecture from [1]

To prevent overfitting and assure generalization capacity, an Early Stopping function was added. Early Stopping kept a close eye on the model's performance on a separate validation set during training. If the validation performance plateaued or began to deteriorate, the training process was terminated, preventing the model from learning the training set and enhancing its capacity to generalize to previously unseen data. The dataset was separated into training, validation, and test sets during training. The models were initialized with pre-trained weights from ImageNet to exploit knowledge gathered from diverse image datasets. The model parameters were fine-tuned based on the gradients generated during backpropagation on the pneumonia dataset. Gradient Descent (GD) using the Adam optimizer was used to iteratively update the model's weights in order to reduce the difference between predicted and real pneumonia labels. The training procedure iterated across a number of epochs, each representing a complete pass over the training dataset. The setting of each model's hyperparameters is shown in Table. I.

E. Evaluation Metrics

The trained CNN, VGG16, and ResNet50 models' performance in pneumonia detection was evaluated using a set of essential evaluation metrics, providing a full understanding of their usefulness in classification tasks.

1) Accuracy: Accuracy is the ratio of successfully predicted occurrences to the total number of instances and measures the overall correctness of the model's predictions. The model's high accuracy reflects its ability to appropriately categorize pneumonia-positive and pneumonia-negative cases.

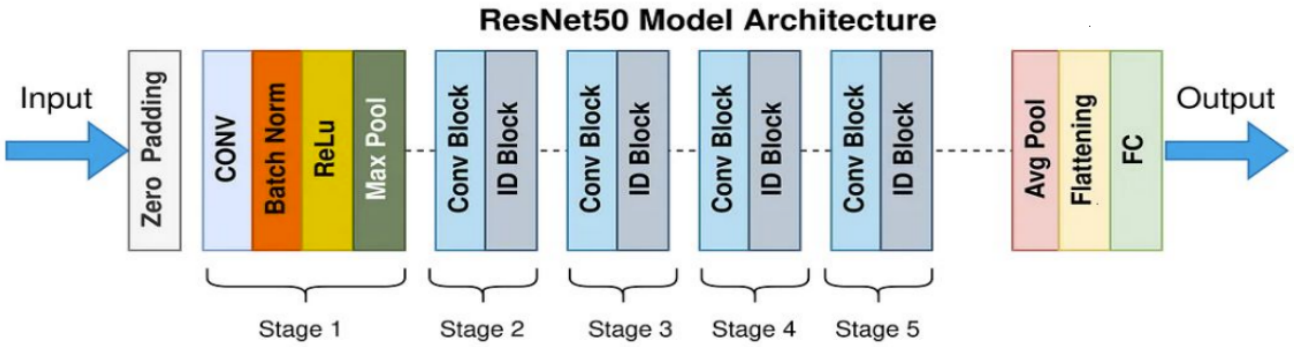

Fig. 6. ResNet50 Architecture from [2]

TABLE I
HYPERPARAMETERS DESCRIPTION

Models	CNN	VGG16 pretrained	ResNet50 pretrained	VGG16 fulltrained	ResNet50 fulltrained
Total Parameters	1.246M	14.79M	23.83M	14.79M	23.83M
Trainable Parameters	1.245M	76,033	272,641	14.79M	23.79M
Learning Rate	0.0001				
Number of Epochs	10				
Batch Size	32				
Loss Function	Binary Cross Entropy				
Optimizer	Adam				
Activation Function	ReLU, Sigmoid				
Dropout Rate	0.2				

2) Precision: Precision, also known as positive predictive value, measures the accuracy of the model's positive predictions. It is derived by dividing the number of true positive predictions by the total number of true positives and false positives. Precision is very important in medical diagnostics since it reflects the capacity to eliminate erroneous positive pneumonia identifications.

3) Recall (Sensitivity): The model's capacity to properly identify all positive cases in the dataset is measured by recall, also known as sensitivity. It is calculated by dividing the number of true positive predictions by the total number of true positives and false negatives. In medicine, high recall is critical to ensuring that pneumonia cases are not neglected.

4) F1 Score: The F1 score fairly assesses a model's performance because it represents the harmonic mean of precision and recall. It is calculated by dividing the sum of precision and recall by two times the product of precision and recall. The F1 score, which provides a combined assessment of precision and recall, is especially effective when there is an imbalance between positive and negative examples.

IV. EXPERIMENTAL RESULTS

The experimental findings of the pneumonia detection models, including the custom CNN, pre-trained and fully-trained VGG16, and ResNet50 models, are presented in this part. The preceding section's evaluation metrics were reported: accuracy, precision, recall, and F1 score. Furthermore, model loss and accuracy visualizations included in Appendix A over epochs demonstrate the training dynamics and convergence behavior.

A. Confusion Matrix

The confusion matrices detail the model's predictions, categorizing them as true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). These matrices depict the model's effectiveness in detecting pneumonia, highlighting areas of strength and potential development. The matrices were built based on the test set, allowing for a study of the classification results of the CNN, VGG16, and ResNet50 models. The confusion matrices of the VGG16 and ResNet50 models are shown in Figs 7-11 in Appendix A . Table II depicts a comparison of all models utilizing evaluation metrics.

B. Model Loss vs. Epochs

The visualization of the training and validation loss over epochs reveals information about the models' convergence and potential overfitting. The graphic depicts the development of the loss function during training, indicating whether the models learned the underlying patterns in the pneumonia dataset adequately. Figures Figs 12-16 in Appendix Ashow the model loss for CNN, VGG16, and ResNet50 models at different epochs.

C. Model Accuracy vs. Epochs

The accuracy and loss vs epochs plots represent the learning trajectories of the models. Examining how accuracy changes

TABLE II
EVALUATION RESULTS BASED ON TEST DATASET

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score
CNN	87.82	93.37	86.67	89.89
Pre-trained VGG16	85.25	81.56	98.71	89.32
Pre-trained ResNet50	85.25	81.56	98.72	89.33
Fully-trained VGG16	62.5	62.5	100.0	76.92
Fully-trained ResNet50	83.33	80.95	95.89	87.79

throughout training epochs allows us to evaluate each model's convergence and generalization capabilities, stability, and robustness. Figures Figs 12-16 in Appendix A show the accuracy of CNN, VGG16, and ResNet50 models as a function of epoch.

D. Results Analysis

On the test dataset, we compare the performance of CNN, Pre-trained, and Fully-trained VGG16 and ResNet50 models. The following section examines the experimental data, digging into the implications of the confusion matrices, summarizing key metrics, and providing insights from the loss and accuracy graphs. Patterns, trends, and potential areas for improvement are investigated, providing a more nuanced view of the performance of the CNN, VGG16, and ResNet50 models in pneumonia diagnosis. Overall, the CNN model slightly beats the other models. There could be several causes for the thoroughly comparedthe VGG16 and ResNet50 models. The VGG16 and ResNet50 models all have good recall and moderate precision. Because these models have extremely deep networks, they may have been influenced by the class imbalance and became prejudiced towards the majority class. Because the VGG16 model was a significantly deep network with no skip connections, it stopped learning after the first several stops when completely trained, probably due to the vanishing gradients problem. Because the ResNet50 model included skip connections, it was able to avoid the vanishing gradient problem and learn to some extent.

V. CONCLUSION AND FUTURE WORK

Finally, this work conducted a thorough comparison of three deep learning architectures, CNN, VGG16, and ResNet, to improve pneumonia identification from chest X-ray pictures. The incorporation of advanced picture preprocessing techniques, such as normalization, scaling, and image augmentation, helped the models' robustness. According to our findings, ResNet50 has the highest overall accuracy. While each architecture demonstrated capabilities in different areas, clinical considerations such as decreasing false negatives or positives may determine the ideal model. This research provides vital insights into the possible applications of deep learning in medical imaging as we move forward. Future research could involve the use of GAN instead of data augmentation, as well as the use of autoencoders for feature extraction and transfer learning.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] L. Račić, T. Popović, S. čakić, and S. Šandi, "Pneumonia detection using deep learning based on convolutional neural network," in *2021 25th International Conference on Information Technology (IT)*, 2021, pp. 1–4.

[4] P. Lakhani and B. Sundaram, "Deep learning at chest radiography: Automated classification of pulmonary tuberculosis by using convolutional neural networks," *Radiology*, vol. 284, no. 2, pp. 574–582, 2017, pMID: 28436741. [Online]. Available: https://doi.org/10.1148/radiol.2017162326

[5] R. Kundu, R. Das, Z. W. Geem, G.-T. Han, and R. Sarkar, "Pneumonia detection in chest x-ray images using an ensemble of deep learning models," *PLOS ONE*, vol. 16, no. 9, pp. 1–29, 09 2021. [Online]. Available: https://doi.org/10.1371/journal.pone.0256630

[6] T. Rahman, M. E. H. Chowdhury, A. Khandakar, K. R. Islam, K. F. Islam, Z. B. Mahbub, M. A. Kadir, and S. Kashem, "Transfer learning with deep convolutional neural network (cnn) for pneumonia detection using chest x-ray," *Applied Sciences*, vol. 10, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/9/3233

[7] M. La Salvia, G. Secco, E. Torti, G. Florimbi, L. Guido, P. Lago, F. Salinaro, S. Perlini, and F. Leporati, "Deep learning and lung ultrasound for covid-19 pneumonia detection and severity classification," *Computers in Biology and Medicine*, vol. 136, p. 104742, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482521005369

[8] P. Szepesi and L. Szilágyi, "Detection of pneumonia using convolutional neural networks and deep learning," *Biocybernetics and Biomedical Engineering*, vol. 42, no. 3, pp. 1012–1022, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0208521622000742

[9] H. A. Sanghvi, R. H. Patel, A. Agarwal, S. Gupta, V. Sawhney, and A. S. Pandya, "A deep learning approach for classification of covid and pneumonia using densenet-201," *Int J Imaging Syst Technol*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9537800/

[10] 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," *IEEE Access*, vol. 10, pp. 62 110–62 128, 2022.

[11] S. Sharma and K. Guleria, "A deep learning based model for the detection of pneumonia from chest x-ray images using vgg-16 and neural networks," *Procedia Computer Science*, vol. 218, pp. 357–366, 2023, international Conference on Machine Learning and Data Engineering. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050923000182

[12] N. Barakat, M. Awad, and B. A. Abu-Nabah, "A machine learning approach on chest x-rays for pediatric pneumonia detection," *Digital Health*, vol. 9, p. 20552076231180008, 2023.

[13] H. Bhatt and M. Shah, "A convolutional neural network ensemble model for pneumonia detection using chest x-ray images," *Healthcare Analytics*, vol. 3, p. 100176, 2023.

[14] H. Ren, F. Jing, Z. Chen, S. He, J. Zhou, L. Liu, R. Jing, W. Lian, J. Tian, Q. Zhang *et al.*, "Chexmed: a multimodal learning algorithm for pneumonia detection in the elderly," *Information Sciences*, p. 119854, 2023.

[15] D. Pippiri, R. Kode, and K. Ravikiran, "Pneumonia detection using cnn variants," in *AIP Conference Proceedings*, vol. 2754, no. 1. AIP Publishing, 2023.
[16] S. H. M. Sohan S. Mugdho, Roop S. Mangineni, "Ee526 project," 2023, available at Kaggle, https://www.kaggle.com/code/ssmugdho/ee526-project/notebook.
[17] P. MOONEY, "Chest x-ray images (pneumonia)," 2018, available at Kaggle, https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia.

APPENDIX A
SUPPLEMENTARY FIGURES

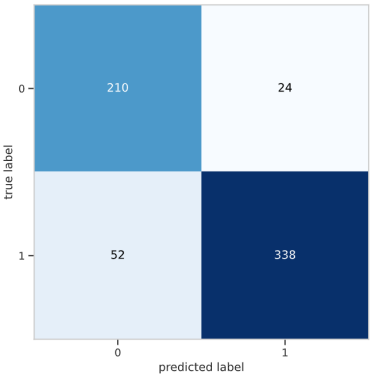

Fig. 7. Confusion Matrix of CNN model

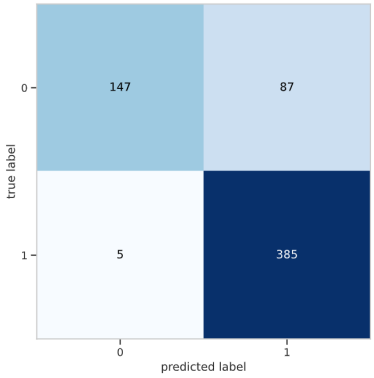

Fig. 8. Confusion Matrix of Pre-trained VGG16 model

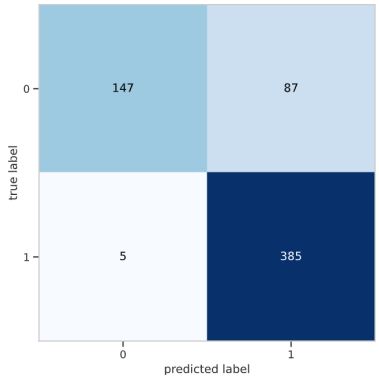

Fig. 9. Confusion Matrix of Pre-trained ResNet50 model

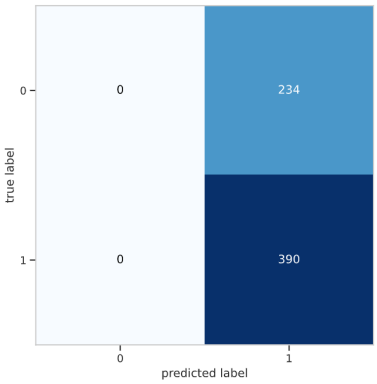

Fig. 10. Confusion Matrix of Fully-trained VGG16 model

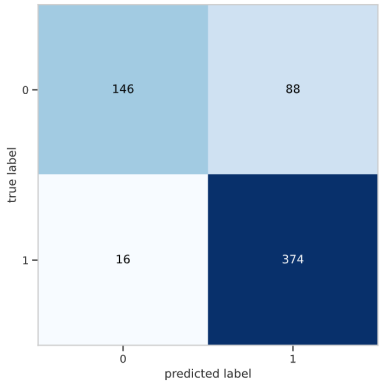

Fig. 11. Confusion Matrix of Fully-trained ResNet50 model

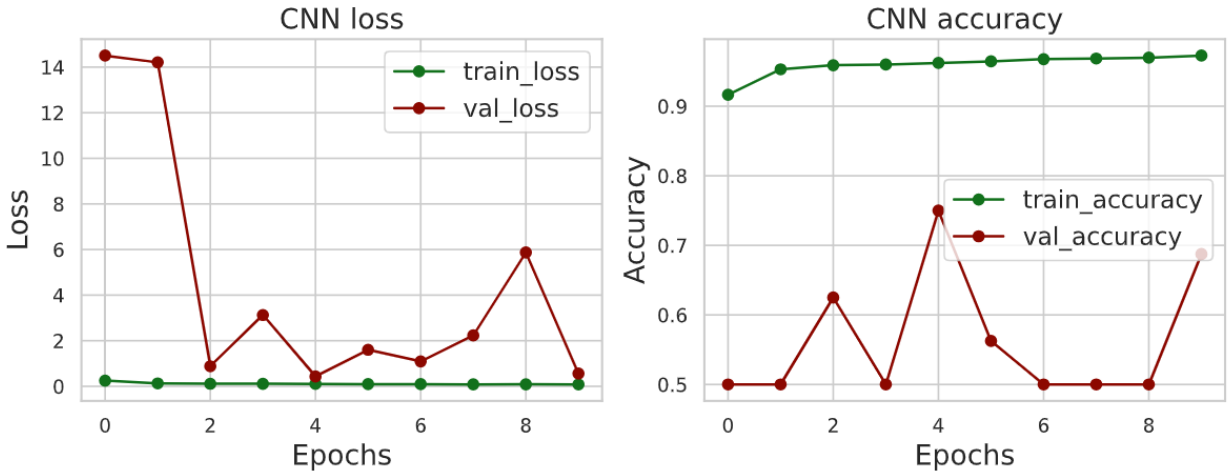

Fig. 12. Training Loss and Accuracy of CNN model

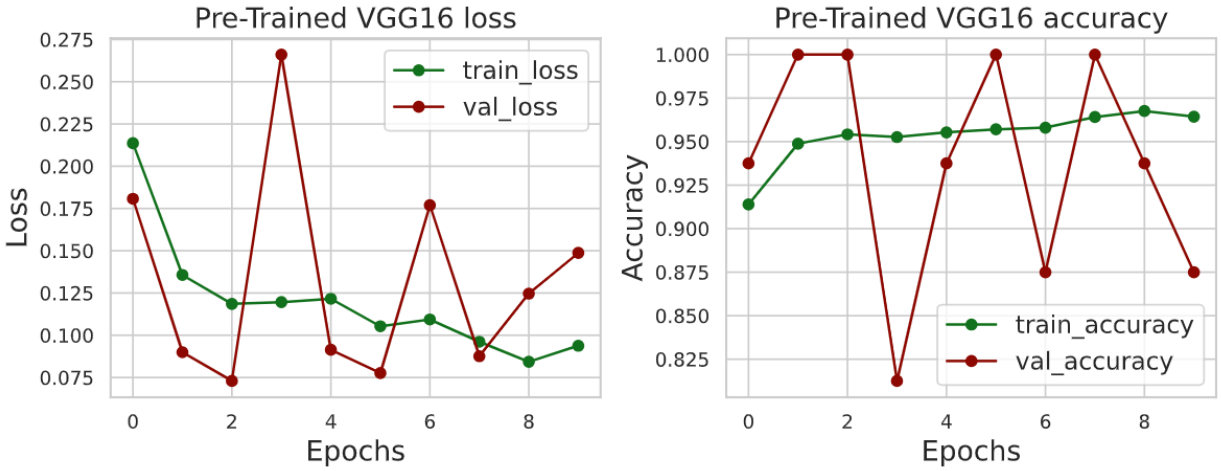

Fig. 13. Training Loss and Accuracy of Pre-trained VGG16 model

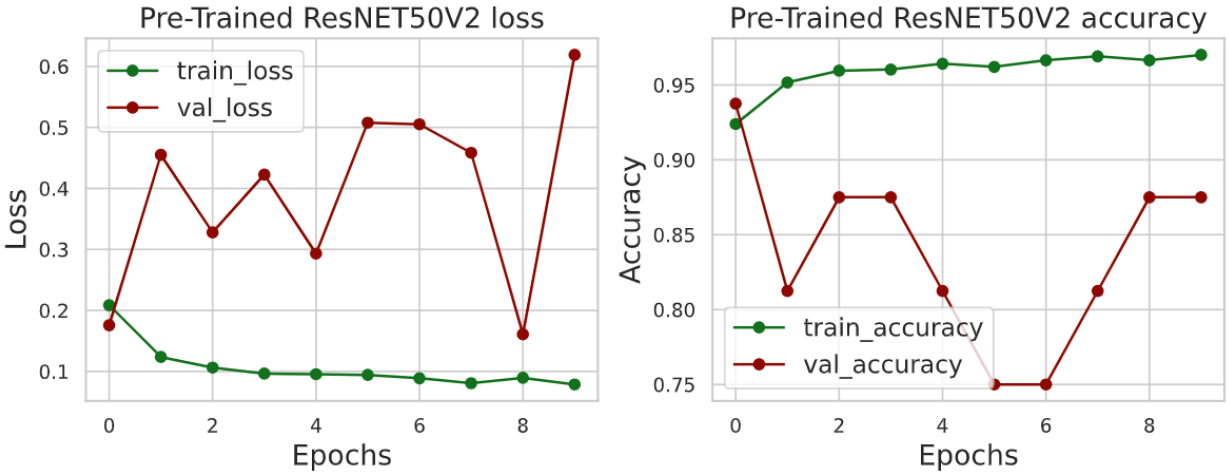

Fig. 14. Training Loss and Accuracy of Pre-trained ResNet50 model

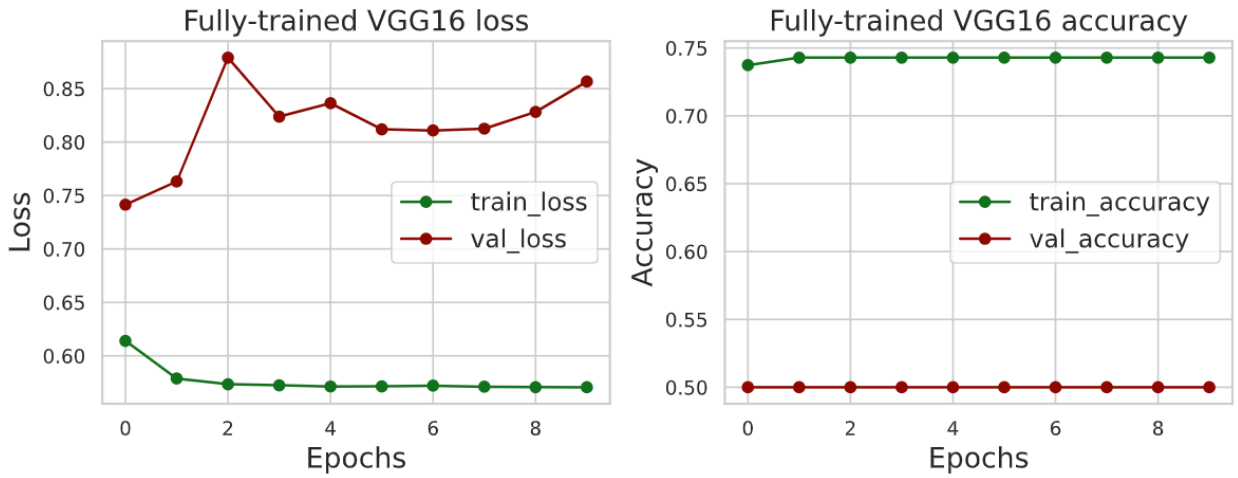

Fig. 15. Training Loss and Accuracy of Fully-trained VGG16 model

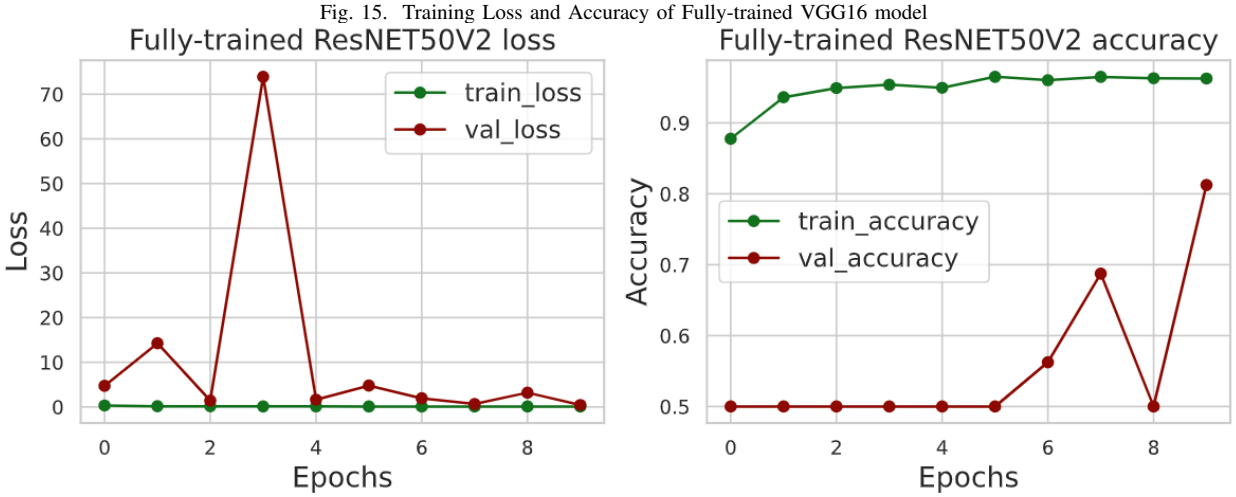

Fig. 16. Training Loss and Accuracy of Fully-trained ResNet50 model

APPENDIX B
SOURCE CODES

The source code is available at [16]. The dataset is available at [17].

A. Import Packages

```
#  importing necessary modules
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Conv2D, Flatten, MaxPool2D,
Dropout, BatchNormalization, MaxPooling2D
from keras.callbacks import ReduceLROnPlateau
from sklearn.metrics import classification_report
import pandas as pd
import numpy as np
import cv2
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
import os
```

```
import random
import zipfile
import warnings
warnings.filterwarnings("ignore")
```

B. Data Generators

```
dir_train = '/kaggle/working/chest_xray/chest_xray/train'
dir_valid = '/kaggle/working/chest_xray/chest_xray/val'
dir_test = '/kaggle/working/chest_xray/chest_xray/test'

# Augmented data generator for training
train_datagen = ImageDataGenerator(
            rescale = 1/255.,
            horizontal_flip = True,
            vertical_flip = False,
            rotation_range = 0.05,
            zca_whitening = False,
            width_shift_range = 0.05,
            height_shift_range = 0.05,
            channel_shift_range = 0.05,
            shear_range = 0.05,
            zoom_range = 0.05)

# simple data generator for validation and training
val_test_datagen = ImageDataGenerator(rescale = 1./255)

train_set = train_datagen.flow_from_directory(dir_train , class_mode = "binary",
            batch_size = 32, target_size = (150, 150))
validation_set = val_test_datagen.flow_from_directory(dir_valid ,
            class_mode = "binary", batch_size = 32, target_size = (150, 150))
test_set = val_test_datagen.flow_from_directory(dir_test ,
            class_mode = "binary", batch_size = 32, target_size = (150, 150))
```

C. Defining early stopping and learning-rate scheduling

```
# defining early stopping
early_stopping_callbacks = tf.keras.callbacks.EarlyStopping(patience = 15,
                                                    restore_best_weights = True,
                                                    verbose = 1)
# learning rate scheduling
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
patience = 2, verbose=1,factor=0.3, min_lr=0.0001)
```

D. Define and train CNN Model

```
model1 = keras.models.Sequential()
# CNN layers with batchnorm, pooling and dropout for feature extraction
model1.add(Conv2D(32, (3,3), strides = 1, padding='same', activation='relu',
input_shape=(150,150,3)))
model1.add(BatchNormalization())
model1.add(MaxPool2D((2,2), strides=2, padding='same'))
model1.add(Conv2D(64, (3,3), strides=1 ,padding='same', activation='relu'))
model1.add(Dropout(0.1))
model1.add(BatchNormalization())
```

```
model1.add(MaxPool2D((2,2), strides=2, padding='same'))
model1.add(Conv2D(64, (3,3), strides=1, padding='same', activation='relu'))
model1.add(BatchNormalization())
model1.add(MaxPool2D((2,2), strides=2, padding='same'))
model1.add(Conv2D(128, (3,3), strides=1, padding='same', activation='relu'))
model1.add(Dropout(0.2))
model1.add(BatchNormalization())

model1.add(MaxPool2D((2,2), strides=2, padding='same'))
model1.add(Conv2D(256, (3,3), strides=1, padding='same', activation='relu'))
model1.add(Dropout(0.2))
model1.add(BatchNormalization())

# fully connected layers for classification
model1.add(MaxPool2D((2,2), strides=2, padding='same'))
model1.add(Flatten())
model1.add(Dense(units=128, activation='relu'))
model1.add(Dropout(0.2))
model1.add(Dense(units = 1, activation='sigmoid'))
# compiling the model
model1.compile(optimizer='adam', loss='binary_crossentropy',
               metrics=['accuracy'])
model1.summary()

# training custom model
history = model1.fit(
        train_set, batch_size=32, shuffle=True,epochs=10,
        validation_data=validation_set,
        callbacks=[early_stopping_callbacks])
```

E. Define and train Pre-trained VGG16 Model

```
# loading pre-trained vgg16 model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import numpy as np


base_model1 = VGG16(include_top = False, weights = "imagenet",
               input_shape = (150, 150, 3), pooling = "max", classes = 2)

# Freeze the layers
for layer in base_model1.layers:
    layer.trainable = False

model2 = Sequential()
model2.add(base_model1)
model2.add(Flatten())

model2.add(Dense(128, activation="relu"))
model2.add(Dense(64, activation="relu"))
model2.add(Dense(32, activation="relu"))
model2.add(Dense(1, activation="sigmoid"))

model2.compile(optimizer="adam", loss="binary_crossentropy",
```

```
                metrics=["accuracy"])
model2.summary()

# train model
history2 = model2.fit(
    train_set, batch_size=32, shuffle=True, epochs=10,
    validation_data=validation_set, callbacks=[early_stopping_callbacks]
)
```

F. Define and train Pre-trained ResNet50 Model

```
# loading pre-trained resnet50V2 model
base_model2 = tf.keras.applications.ResNet50V2(weights = "imagenet",
                                               input_shape = (150, 150, 3),
                                               pooling = "max",
                                               include_top = False,
                                               classes = 2)

for layer in base_model2.layers:
    layer.trainable = False

# compiling resnet50v2 model
model3 = Sequential()
model3.add(base_model2)
model3.add(Flatten())

model3.add(Dense(128, activation = "relu"))
model3.add(Dense(64, activation = "relu"))
model3.add(Dense(32, activation = "relu"))
model3.add(Dense(1, activation = "sigmoid"))

model3.compile(optimizer = "adam", loss = "binary_crossentropy",
               metrics = ["accuracy"])

model3.summary()

# training resnet50v2 model
history3 = model3.fit(
    train_set, batch_size=32, shuffle=True, epochs=10,
    validation_data=validation_set, callbacks=[early_stopping_callbacks]
)
```

G. Define and train Fully-trained VGG16 Model

```
# loading untrained vgg16 model
base_model3 = VGG16(include_top = False, weights = None,
            input_shape=(150, 150, 3),
            pooling = "max",
            classes = 2)

# Freeze the layers
for layer in base_model3.layers:
    layer.trainable = True

model4 = Sequential()
model4.add(base_model3)
```

```
model4.add(Flatten())

model4.add(Dense(128, activation="relu"))
model4.add(Dense(64, activation="relu"))
model4.add(Dense(32, activation="relu"))
model4.add(Dense(1, activation="sigmoid"))

model4.compile(optimizer="adam", loss="binary_crossentropy",
               metrics=["accuracy"])
model4.summary()

# training fully-trained vgg16 model
history4 = model4.fit(
    train_set, batch_size=32, shuffle=True, epochs=10,
    validation_data=validation_set, callbacks=[early_stopping_callbacks]
)
```

H. Define and train Fully-trained ResNet50 Model

```
# loading pre-trained resnet50V2 model
base_model4 = tf.keras.applications.ResNet50V2(weights = None,
                                               input_shape = (150, 150, 3),
                                               pooling = "max",
                                               include_top = False,
                                               classes = 2)

for layer in base_model4.layers:
    layer.trainable = True

# compiling resnet50v2 model
model5 = Sequential()
model5.add(base_model4)
model5.add(Flatten())

model5.add(Dense(128, activation = "relu"))
model5.add(Dense(64, activation = "relu"))
model5.add(Dense(32, activation = "relu"))
model5.add(Dense(1, activation = "sigmoid"))

model5.compile(optimizer = "adam", loss = "binary_crossentropy",
               metrics=["accuracy"])

model5.summary()

# training resnet50v2 model
history5 = model5.fit(
    train_set, batch_size=32, shuffle=True, epochs=10,
    validation_data=validation_set, callbacks=[early_stopping_callbacks]
)
```

I. Sample code for test set evaluation

```
# print test scores
scores = model1.evaluate(x_test, y_test, verbose=0)
print("Score : %.2f%%" % (scores[1]*100))
```

J. Sample code for generating confusion matrix and evaluation metrics

```
from sklearn.metrics import accuracy_score, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt

# creating confusion matrix for CNN
print("CNN Confusion Matrix : \n")

predictions = model1.predict(x_test)
conf_m = confusion_matrix(y_test, np.round(predictions))
acc = accuracy_score(y_test, np.round(predictions)) * 100
tn, fp, fn, tp = conf_m.ravel()

fig, ax = plot_confusion_matrix(conf_mat = conf_m, figsize = (6, 6),
        cmap = plt.cm.Blues)
plt.show()

# generating evaluation metrics for CNN
precision = tp / (tp + fp) * 100
recall = tp / (tp + fn) * 100
print("Accuracy: {}%".format(acc))
print("Precision: {}%".format(precision))
print("Recall: {}%".format(recall))
print("F1-score: {}".format(2 * precision * recall / (precision + recall)))
```