

Project Proposal: CNN Architecture to Detect PCOS from Ovarian Ultrasound Images

Group Members:

- **Mahima Agarwal || 2nd Year**
- **Shweta Meena || 2nd Year**
- **Charvi Mehra || 2nd Year**
- **Divya Bandaru || 2nd Year**
- **Tushitaa Narayan Ojha || 2nd Year**
- **Swarnim Sharma || 2nd Year**
- **Kritika Verma || 3rd Year**
- **Arpita Keshwarni || 3rd Year**

Abstract

Detection of PCOS using Ultrasound Images and Statistical Data

Polycystic ovary syndrome (PCOS) has been classified as a severe health problem common among women globally. Polycystic ovary syndrome (PCOS) can affect a female's reproductive system and comes with associated complications to the endocrine system of the affected individual. The diagnosis success of the condition varies depending on the stage of the disease. Early detection and treatment of PCOS reduce the possibility of long-term complications, such as increasing the chances of developing type 2 diabetes and gestational diabetes. Thus, there is a need for investment in additional technologies that can help bolster the overall diagnosis success of the condition and cue in prompt care strategies. A substantial amount of work has been done on this, where artificial intelligence technology has been investigated around the exploitation of patient medical health records towards predicting whether a patient is carrying the PCOS condition. Therefore, effective and early PCOS diagnosis will help the healthcare systems to reduce the disease's problems and complications. Machine learning (ML) has recently shown promising results in medical diagnostics. The main goal of our project was to provide model explanations to ensure efficiency, effectiveness, and trust in the developed model. Different types of ML models (logistic regression (LR), random forest (RF), decision tree (DT), naive Bayes (NB), support vector machine (SVM), k-nearest neighbor (KNN) algorithm have been implemented to find the best model. The shortcomings associated with this related literature are based on the use of an unbalanced dataset towards the training of the candidate models, which can induce a form of model bias and approach the problem as a binary-based prediction exercise. This study aims to provide a solution to this apparent gap in knowledge by designing a prediction model using the Kaggle PCOS dataset, which is initially balanced using a synthetic sample generation algorithm. Next, a probability-based inference system is designed to estimate and stage the degree of the PCOS condition in the patient.

1. INTRODUCTION

1.1 Background and Context

PCOS affects the health of women by causing hormonal imbalances and metabolism problems. It is a disease that primarily affects women's fertility, as 5 to 10% of females suffer from this disease in their childbearing years (15–45). It is a hormonal disorder that causes problems with the ovaries. In the normal state, the ovaries produce hormones (chemicals that control the functions of the body), namely estrogen (female hormone) and androgens (male hormones), for normal health. In affected women, the hormones are imbalanced, with higher androgens or less estrogen than normal. This causes lumps (fluid-filled sacs) to grow on the ovaries. These lumps gradually enlarge and then obstruct the ovulation process. This disruption of ovulation in women with PCOS reduces their chances of becoming pregnant. Women with PCOS are more likely to develop diabetes, heart disease, high blood pressure, endometrial thickness, sleep apnea, depression, anxiety, eating disorders, and endometrial cancer. In addition to genetic factors, environmental factors may also contribute to PCOS development. In addition to early diagnosis, treatment, and weight loss, long-term complications can be reduced. Pioneering work in the area of polycystic ovary syndrome (PCOS) is attributed to the likes of Stein and Leventhal back in 1935, who identified traits in women such as amenorrhea, hirsutism, and large ovaries that had a number of cysts. Subsequently, the medical literature now robustly describes PCOS as a heritable disease with symptoms that include hormonal imbalances and ovulatory dysfunction, as well as the presence of polycystic ovaries. The prevalence rate of PCOS is high among Indian women. The pooled prevalence of PCOS was close to 10% using Rotterdam's criteria and AES criteria, while it was 5.8% using NIH criteria. Table 1 shows the manifestation of PCOS around various stages of development. Artificial intelligence (AI) has gained steady momentum in its application across clinical medicine, where the concept of machine learning—a subset of AI—is themed around the use of data-driven means towards optimising prediction machines to learn from carefully selected data for its design, tuning and calibration. These machine learning models have been mostly investigated in a research capacity thus far, and have created avenues which have facilitated enhanced functionality in the area of rehabilitation medicine in upper limb prosthesis control. Furthermore, these models have aided early and proactive disease predictions in the public health sectors in areas such as various kinds of cancers, obstetric cases for proactive prediction of premature births, and in psychiatry for the firm diagnosis of variants of psychosis that are hard to diagnose. Artificial intelligence (AI) has revolutionized the detection and treatment of diseases, specifically PCOS. AI-based technologies such as machine learning (ML) algorithms and deep learning networks (DL)

have enabled the development of automated systems for the accurate and reliable detection of heart disease. AI-based methods can identify patterns in medical data, such as hormone levels, to distinguish PCOS patients from those without the disorder. This improved accuracy could lead to earlier, more accurate diagnoses and better overall outcomes for PCOS patients. Furthermore, AI-based systems can be used, to monitor patients over time, providing clinicians with valuable insights into potential treatments and enabling more precise interventions. In short, AI-driven technology has the potential to revolutionize PCOS detection and treatment, providing more effective and efficient care for those suffering from the condition.

The detection of multiple cysts using ovary ultrasonography (USG) scans is one of the most reliable approach for making an accurate diagnosis of PCOS and creating an appropriate treatment plan to heal the patients with this syndrome. In this research, an extended machine learning classification technique for PCOS prediction has been proposed, trained and tested over 1700 ovary USG images; where the Convolutional Neural Network (CNN) incorporating different state-of-the-art techniques and transfer learning has been employed for feature extraction from the images; and then stacking ensemble machine learning technique using conventional models as base learners and bagging or boosting ensemble model as meta-learner have been used on that reduced feature set to classify between PCOS and non-PCOS ovaries. The proposed technique significantly enhances the accuracy while also reducing training execution time comparing with the other existing ML based techniques.

1.2 Scope and Objectives

Scope -

The detection of Polycystic Ovary Syndrome (PCOS) using ultrasound and statistical data is a relevant and impactful project in today's world. PCOS is a common endocrine disorder among women of reproductive age, and its early detection is crucial for timely management and prevention of complications. Here are some aspects of the scope for such a project:

1. **Prevalence of PCOS:** PCOS is a prevalent condition, affecting a significant percentage of women worldwide. A statistical analysis can help provide insights into the prevalence of PCOS in different populations, ethnic groups, and age ranges.

- 2. Patient Education and Awareness:** Your project can be a basis for educational initiatives aimed at increasing awareness about PCOS. Providing information to the public, healthcare professionals, and policymakers can contribute to better understanding, early detection, and management of PCOS.
- 3. Technological Advancements:** As technology continues to advance, your project may contribute to the development of more sophisticated and accurate diagnostic tools for PCOS. This could include the integration of artificial intelligence and machine learning algorithms to enhance the accuracy of ultrasound-based diagnosis
- 4. Global Health Impact:** PCOS affects women globally, and your project could have a positive impact on global women's health. Sharing findings and collaborating with international researchers can contribute to a better understanding of PCOS on a global scale.

Objectives -

1. Data Collection and Compilation:

- Gather a diverse dataset of ultrasound images and associated statistical data from individuals with and without PCOS.
- Ensure that the dataset is representative of different age groups, ethnicities, and PCOS severity levels.

2. Data Preprocessing:

- Implement image preprocessing techniques to enhance the quality and standardize the ultrasound images.
- Clean and preprocess statistical data to remove outliers and ensure consistency.

3. Feature Extraction:

- Identify relevant features from ultrasound images, such as follicle count, ovarian volume, and texture patterns.
- Extract key statistical parameters from the dataset, including hormone levels, BMI, and menstrual cycle characteristics.

4. Algorithm Development:

- Design and implement machine learning or deep learning algorithms for PCOS detection based on the extracted features.
- Evaluate different models (e.g., convolutional neural networks) to identify the most effective approach.

5. Validation and Performance Assessment:

- Split the dataset into training and testing sets to assess the model's generalization ability.
- Employ cross-validation techniques to ensure robustness and reliability of the developed algorithms.
- Evaluate the model's performance using metrics such as sensitivity, specificity and accuracy.

6. Interpretability and Explainability:

- Implement methods to enhance the interpretability and explainability of the model's predictions, ensuring that healthcare professionals can trust and understand the results.

7. Ethical Considerations:

- Address privacy concerns related to patient data, ensuring compliance with relevant regulations (e.g., HIPAA).
- Consider the potential impact of false positives/negatives and implement measures to minimize risks.

8. Integration with Healthcare Systems:

- Explore opportunities for integrating the developed model into existing healthcare systems to facilitate seamless adoption by medical professionals.

9. Continued Improvement:

- Establish a framework for continuous improvement, allowing the model to adapt to emerging research findings and updates in diagnostic criteria.

2. FORMULATION OF THE PROBLEM

2.1 Software Used

Python

Python is a programming language that lets you work more quickly and integrate your systems more effectively. Python is used successfully in thousands of real-world business applications around the world, including many large and mission critical systems.



Colab

Colab is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It supports multiple programming languages including Python, which is one of the most widely used programming languages in the data science and scientific computing communities. It is often used for data analysis, numerical simulation, machine learning, and more.



Kaggle

Kaggle is a data science competition platform and online community of data scientists and machine learning practitioners under Google LLC. Kaggle enables users to find and publish datasets, explore and build models in a web-based data science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.



2.2 Methodology

2.2.1 Models Explored

a) For statistical data

- 1) **Logistic Regression (LR):** this is another statistically driven model, which uses a threshold alongside a nonlinear activation function in the form of a sigmoid towards the determination of the data class of input samples using a binary-like format.
- 2) **K-Nearest Neighbour (KNN):** this model utilizes a voting strategy based on a designated number of nearest neighbours towards the assigning of samples to various data classes. In this work, the parameter of k was selected to be 1, while the distance metric of choice was selected to be the Euclidean distance.
- 3) **Support Vector Machine (SVM):** this model iteratively solves an optimisation problem based around the projection of a feature matrix into a higher dimensional space on the basis that data classes have a greater separability in a higher dimensional space. This is done using a small subset of the data which is referred to as the support vectors. Once the class partitioning is completed in a higher dimensional space, the coordinates for the instilled class boundaries are retained as the data is projected downwards to a lower dimensional space in an approach which is known as a kernel trick.
- 4) **Naive Bayes (NB):** this model is a probabilistic machine learning algorithm based on Bayes' theorem with a "naive" assumption that input features are conditionally independent given the class label. Despite its simplicity, Naive Bayes is widely used for classification tasks, particularly in natural language processing applications such as spam filtering and sentiment analysis. The model calculates the probability of each class given the observed features, and the class with the highest probability is assigned as the prediction.
- 5) **Random Forest Classifier (RFC):** It is a machine learning algorithm that operates by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each decision tree is constructed based on a random subset of the training data and a random subset of the features. This randomness helps mitigate overfitting and increases the model's robustness. The final prediction is determined by aggregating the predictions of all the trees, making Random Forests a powerful and widely used algorithm in practice for tasks such as classification, regression, and feature selection.

b) FOR ULTRASOUND IMAGES

Convolutional Neural Network (CNN)

It is a class of deep learning neural networks designed for processing and analyzing visual data. Inspired by the visual processing in the human brain, CNNs are particularly effective in tasks related to image recognition, classification, and feature extraction. The key innovation of CNNs lies in their ability to automatically learn hierarchical representations of features from input images through a series of convolutional and pooling layers. Convolutional layers apply filters to input images, capturing local patterns and features. Pooling layers reduce spatial dimensions, focusing on the most important information. Fully connected layers combine learned features to make predictions. CNNs have demonstrated remarkable success in various domains, including computer vision, medical image analysis, and autonomous vehicles, making them a foundational technology in modern artificial intelligence applications.

2.2.2 Dataset

Ultrasound images

- The dataset "PCOS detection using ultrasound images" **available on Kaggle.**
- **Size:** The dataset contains a total of 2,000 ovarian ultrasound images, with 787 infected and 1145 normal images.
- Data folder consist of 'train' and 'test' subfolders containing 2 categories of data 'infected' and 'notinfected'
 - **infected** : Images of ovaries having PCOS
 - **notinfected** : Images of healthy ovaries

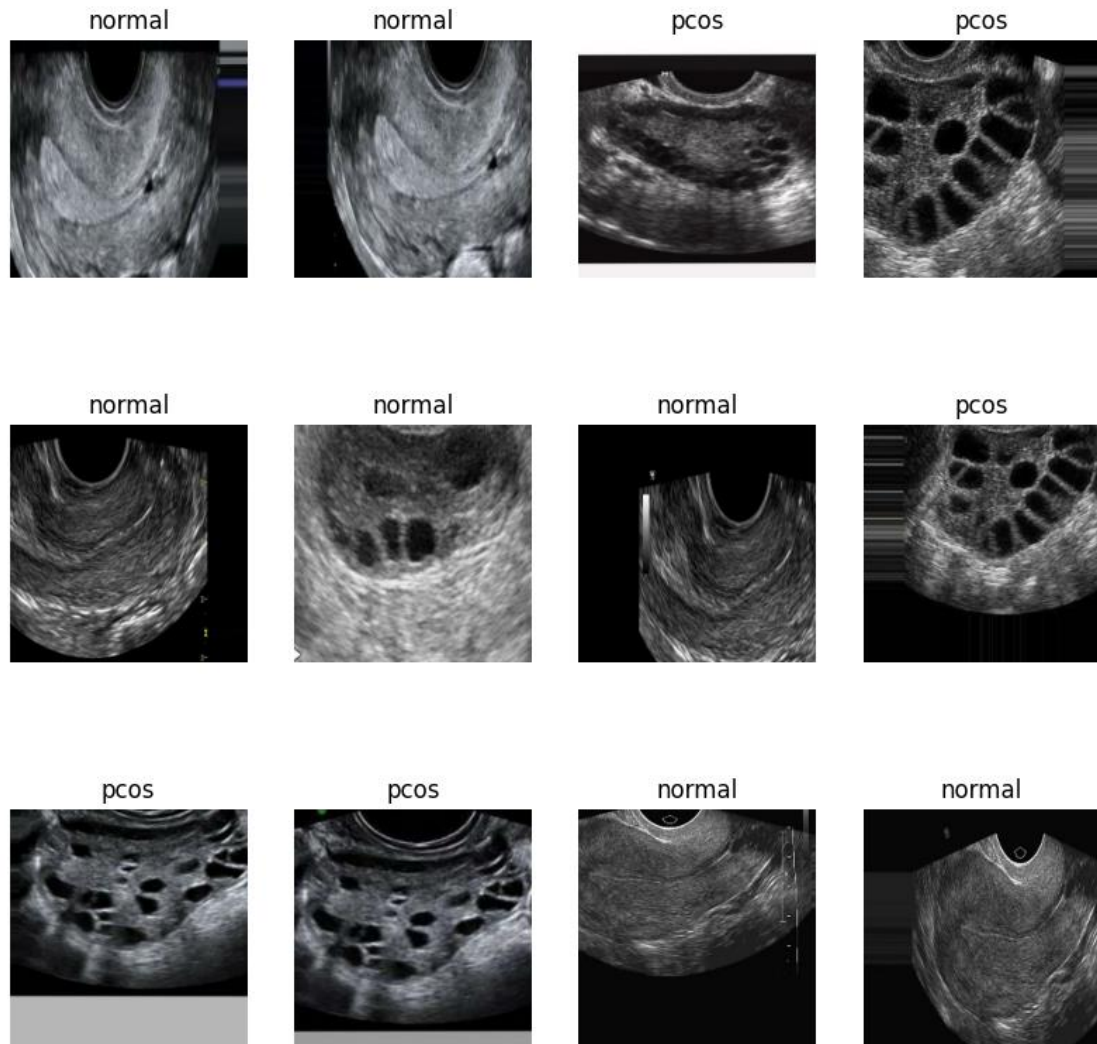


Figure 1 : Sample Images from dataset

Statistical data

- The dataset "Polycystic ovary syndrome (PCOS)" available on Kaggle contains physical and clinical parameters to determine PCOS and infertility-related issues.
- The data is collected from **10 different hospitals** and includes information on various aspects of PCOS, **such as demographics, clinical symptoms, and laboratory findings.**

Feature Name	Abb	Description
Patient File No.		Patient file number (unique identifier)
Polycystic Ovary Syndrome	PCOS	Class label (determine if the patient has this syndrome or not)
Age	AGE	Patient's age in years
Weight	WEIGHT	Patient's weight in KG
Height	HEIGHT	Patient's height in CM
Body Mass Index	BMI	Body mass index of the patient (height/weight)
Blood Group	BG	Patients belong to which blood group (A+, A-, B+, B-, O+, O-, AB+, AB-)
Pulse Rate	PR	Heartbeat per minute
Respiration Rates	RR	Respiration rates per minute
Hemoglobin	HB	Number of red blood cells in patient's body
Cycle	CYCLE	Length of the menstrual cycle
Cycle Length	CL	Number of days of a cycle
Marriage Status	MS	Number of years since marriage
Pregnant	P	Pregnant status
No. of Abortions	AB	No. of abortions
I Beta-HCG	BETA_I	Amount of human chorionic gonadotropin
Beta Healthy Singleton Pregnancy	BETA_II	Beta HCG level is indication of 100 mIU/ml about 16 days after ovulation,
Follicle-Stimulating Hormone	FSH	Attributes ranging from 0.3 to 10.0 mIU/mL indicate if are still menstruating or have undergone menopause
Luteinizing Hormone	LH	Chemical agitator that stimulates the reproductive system
Follicle-Stimulating Hormone/ Luteinizing Hormone	FSH/LH	Ratio of FSH and LH
Hip Size	HIP	Size of hip in inches
Waist Size	WAIST	Size of waist in inches
Waist-Hip Ratio	HIP_RATIO	Waist size proportion to hip
Thyroid-Stimulating Hormone	TSH	Amount of TSH in the blood
Anti-Mullerian Hormone	AMH	Plays a key role in developing a baby's sex organs while in the womb
Prolactin levels	PRL	Prolactin levels in women's bodies
Vitamin D	VIT_D3	Vitamin D levels
Progesterone Levels	PRG	Progesterone levels

Random Blood Sugar	RBS	Value of random blood sugar (RBS) test
Weight Gain	WG	Test to check if the patient gains weight
Hair Growth	HG	Test to check if a patient has hair growth
Hair Loss	HL	Test to check hair loss
Pimples	PIMPLES	Pimple issues
Fast Food	FF	Check if fast food part of the diet
Reg.Exercise	RE	Check if patient exercises on a regular basis
Blood Pressure Systolic	BP_ SYSTOLIC:	Amount of pressure in the arteries when the heart is contracting
Blood Pressure Diastolic	BP_ Diastolic	Amount of pressure in the arteries while the heart is resting in between heart beats
Follicle No.	FN	Follicle number in the left side

Figure 2 :

2.2.3 Implementation of CNN Model

2.2.3.1 Dataset Splitting

To achieve a balanced division of the dataset, we utilized a function named ``get_dataset_partitions_tf``. This function accepts a dataset (``ds``) and three split ratios for training, validation, and testing. The function ensures that the sum of these ratios equals 1, asserting the correctness of the input splits.

2.2.3.2 Splitting Process

The dataset size (``ds_size``) is determined, and if specified, shuffling is performed using a seed of 12. The sizes of training, validation, and test sets are then calculated based on the provided split ratios. The ``take`` and ``skip`` operations are utilized to create distinct subsets:

- **Training set (``train_ds``):** The initial portion of the dataset according to the training split.
- **Validation set (``val_ds``):** The subsequent portion of the dataset after the training set, based on the validation split.
- **Test set (``test_ds``):** The remaining portion of the dataset after both training and validation sets.

2.2.3.3 Dataset Preprocessing

Image preprocessing techniques play a crucial role in preparing images for machine learning model training. They enhance image quality, optimize trainable parameters for AI models, and address variability in disease classification.

Following the dataset splitting, we incorporated caching, shuffling, and prefetching operations to optimize the efficiency of data processing during model training.

2.2.3.3.1. Caching

All three subsets—training, validation, and test—are cached. Caching involves storing elements of the dataset in memory or on disk to accelerate subsequent access.

2.2.3.3.2. Shuffling

To introduce randomness into the training process, shuffling is applied. Each subset is shuffled with a buffer size of 1000.

2.2.3.3.3 Prefetching

Prefetching is implemented to overlap data preprocessing and model execution. It allows the preparation of the next batch of data while the current batch is being processed, optimizing overall training performance. The `AUTOTUNE` parameter is used to dynamically adjust the prefetch buffer size.

2.2.3.4 Building the Model

2.2.3.4.1 Resizing and Normalization Layer

In order to prepare our image data for the neural network, a preprocessing layer has been implemented in the Sequential Model. This layer is designed to resize the images to a specified size (in this case, 224x224 pixels) and normalize pixel values to enhance model performance. Normalization is achieved by scaling pixel values between 0 and 1 through division by 224. This resizing and normalization step is applied both during training and inference to maintain consistency.

It may seem redundant to resize (224,224) images to the same dimensions, but this step serves a crucial purpose post-training. During inference, users might input images of

different dimensions, and this layer ensures uniformity by resizing them to the desired dimensions.

2.2.3.4.2 Enhancing Dataset with Augmentation

Recognizing the importance of data augmentation, a Sequential Model has been employed to apply various transformations to the training dataset. Data augmentation plays a pivotal role in enhancing model accuracy, particularly when the dataset is limited. Two key augmentation techniques, horizontal and vertical flipping, as well as random rotation (up to 0.2 radians), have been incorporated to diversify the dataset.

To effectively incorporate data augmentation, the `train_ds` (training dataset) is mapped using a lambda function. This mapping applies the data augmentation transformations to each image in the training set, with the 'training' parameter set to `True` to signify the training phase. Additionally, the dataset is prefetched for optimal performance during training.

These steps collectively contribute to the construction of a robust image classification model, emphasizing the importance of preprocessing and data augmentation in achieving superior performance.

2.2.3.5 Model Architecture

We implemented a Convolutional Neural Network (CNN) for image classification, leveraging its effectiveness in handling such tasks. The model architecture consists of initial layers for resizing, normalization, and data augmentation, followed by multiple convolutional layers and max-pooling layers. The final layers include a flattening layer, a densely connected layer with ReLU activation, and an output layer with softmax activation for multi-class classification.

2.2.3.6 Compiling the Model

We used the Adam optimizer, Sparse Categorical Cross Entropy for loss calculation, and accuracy as the evaluation metric.

2.2.3.7 Training

The model was trained on a dataset with 15 epochs, utilizing a batch size of 32. The training process yielded promising result.

2.2.3.8 Plotting the Accuracy and Loss Curves

In this section, the training history of the model is visualized through accuracy and loss curves. The training process was conducted over 15 epochs. The history object, obtained from TensorFlow's Keras, contains various metrics such as accuracy, loss, validation accuracy, and validation loss.

The structure of the history object is explored using the `history.params` attribute, revealing details about verbosity, epochs, and steps. The keys present in the `history.history` dictionary include 'loss,' 'accuracy,' 'val_loss,' and 'val_accuracy.' These keys are associated with Python lists containing values corresponding to each epoch.

2.2.3.9 Analysis of Training Metrics

The type of the loss values is confirmed as a list, and the length of the list corresponds to the number of epochs (15 in this case). A snippet of the initial loss values for the first five epochs is displayed for examination.

2.2.3.10. Visual Representation of Training and Validation Metrics

Two subplots are generated to visually represent the training and validation accuracy along with the training and validation loss. The plots provide insights into the model's performance and help identify trends, overfitting, or underfitting.

2.2.3.11 Run Prediction on a Sample Image

A sample image is taken from the test dataset, and the model's prediction is showcased. Both the actual and predicted labels are displayed for visual verification. This serves as an illustrative example of the model's performance on individual instances.

2.2.3.12. Function for Inference

A reusable function for model inference is defined, named `predict(model, img)`. This function takes an image as input, preprocesses it, and returns the predicted class and confidence level. The confidence level represents the model's certainty in its prediction.

2.2.3.13 Inference on Sample Images

Several sample images from the test dataset are selected, and the previously defined inference function is applied. The results are visualized in a 3x3 grid, displaying the actual class, predicted class, and confidence level for each image. This provides a qualitative assessment of the model's performance on multiple instances.

→ Changed the activation function in the layers of my CNN model and compared the results

The Activation Functions can be basically divided into 2 types-

- 1. Linear Activation Function**
- 2. Non-linear Activation Functions**

1. Rectified Linear Units (ReLU)

The Rectified Linear Unit is the most commonly used activation function in deep learning models. ReLU function increases the complexity of the neural network by introducing non-linearity, which allows the network to learn more complex representations of the data.

The ReLU function is defined as

$$f(x) = \max(0, x),$$

which sets all negative values to zero. By setting all negative values to zero, the ReLU function creates multiple linear regions in the network, which allows the network to represent more complex functions.

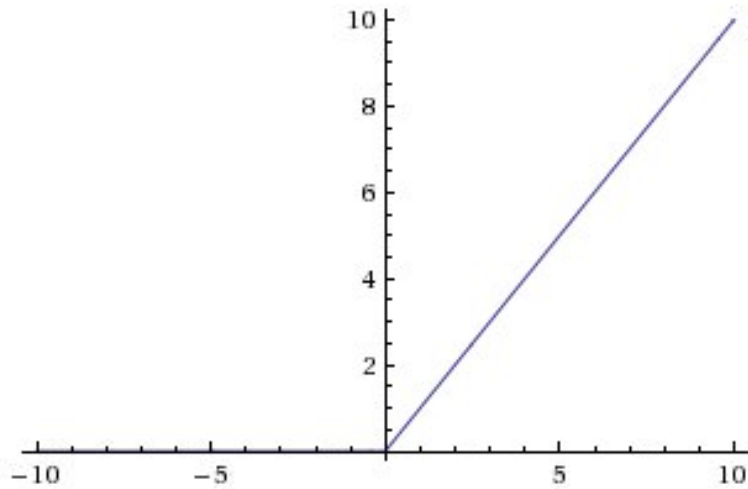


Figure 3: ReLU

2. Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape.

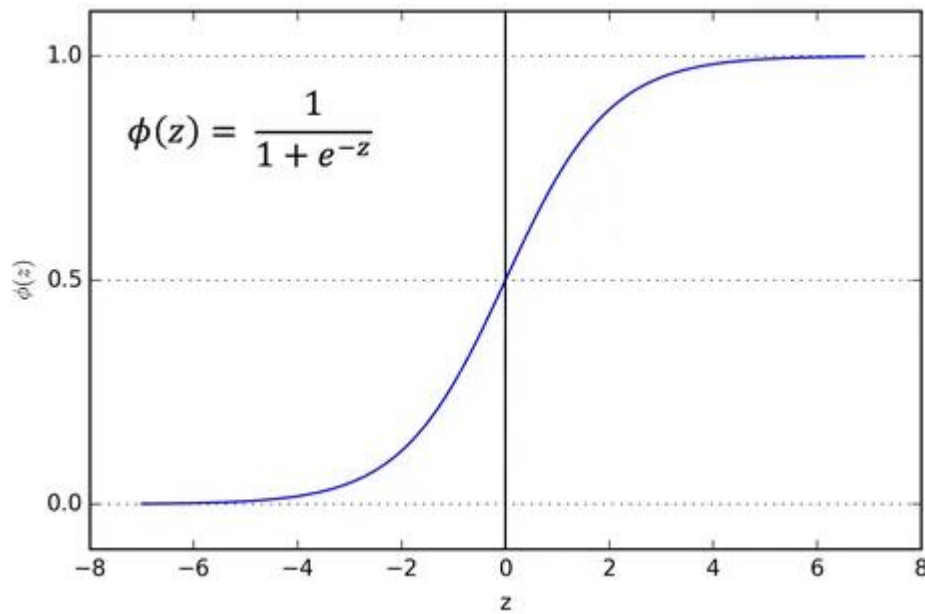


Figure 4: Sigmoid Function

The main reason why we use sigmoid function is because it exists between **(0 to 1)**. Therefore, it is especially used for models where we have to **predict the probability** as

an output. Since probability of anything exists only between the range of **0 and 1**, sigmoid is the right choice.

The function is **differentiable**. That means, we can find the slope of the sigmoid curve at any two points. The function is **monotonic** but function's derivative is not.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

The **softmax function** is a more generalized logistic activation function which is used for multiclass classification.

3. Linear or Identity Activation Function

As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.

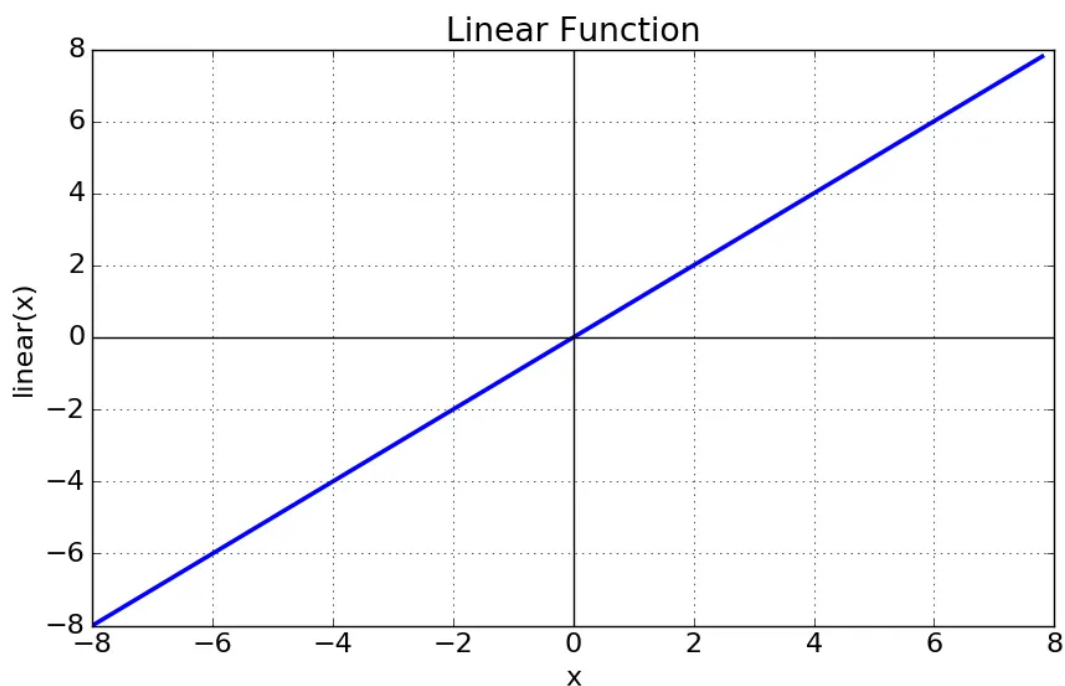


Figure 5: Linear function

Equation : $f(x) = x$

Range : (-infinity to infinity)

It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

4. **Tanh or hyperbolic tangent Activation Function**

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

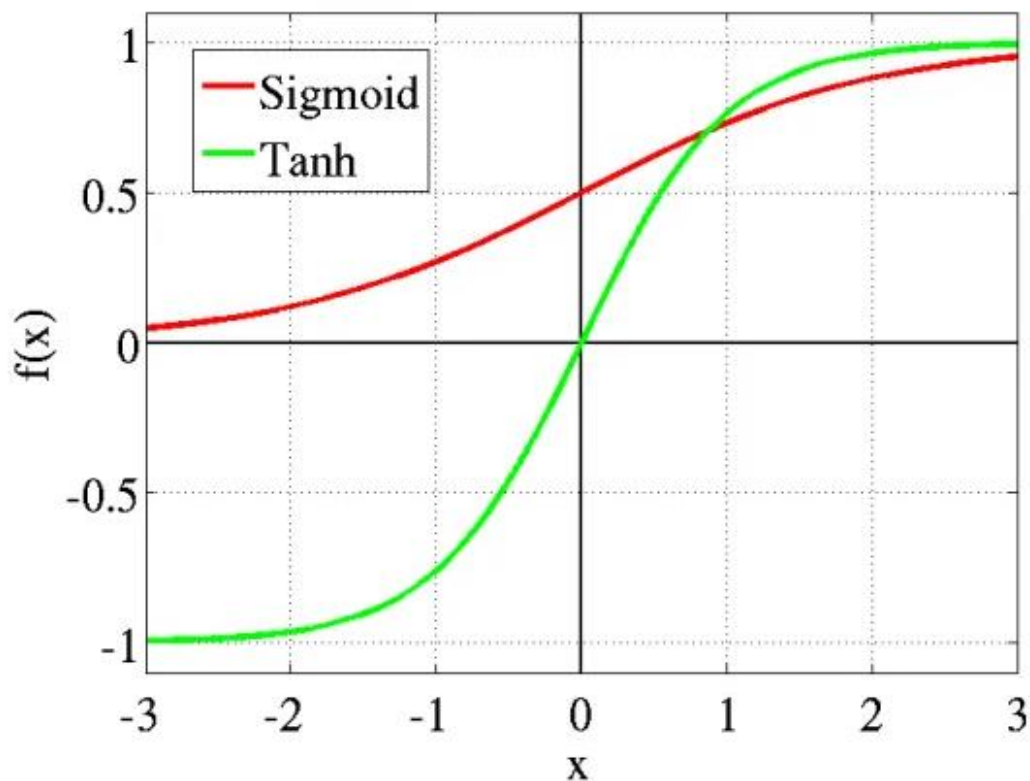


Figure 6: tanh v/s Logistic Sigmoid

The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

The function is **differentiable**.

The function is **monotonic** while its **derivative is not monotonic**.

The tanh function is mainly used for classification between two classes.

Both tanh and logistic sigmoid activation functions are used in feed-forward nets.

5. Leaky ReLU

It is an attempt to solve the dying ReLU problem

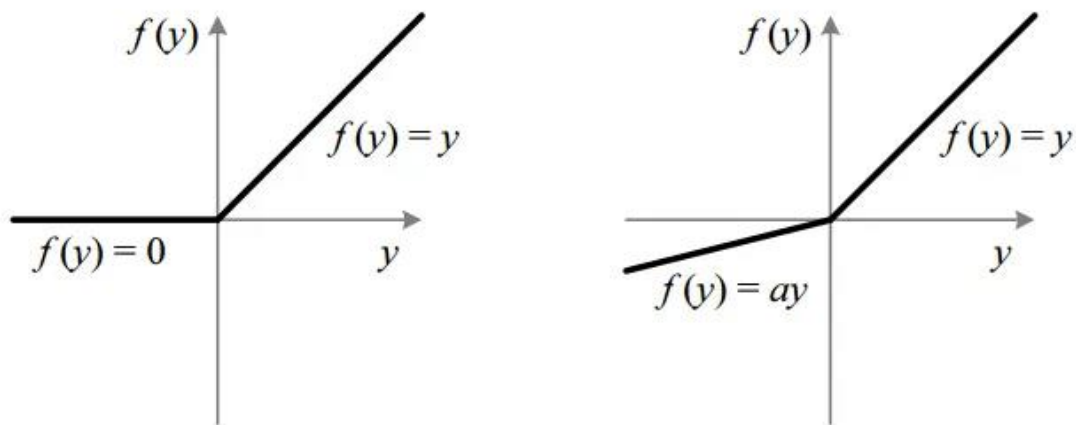


Figure 7 : ReLU v/s Leaky ReLU

The leak helps to increase the range of the ReLU function. Usually, the value of **a** is 0.01 or so.

When **a** is **not 0.01** then it is called **Randomized ReLU**.

Therefore the **range** of the Leaky ReLU is (-infinity to infinity).

Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.

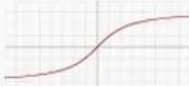
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Figure 8: Activation Function

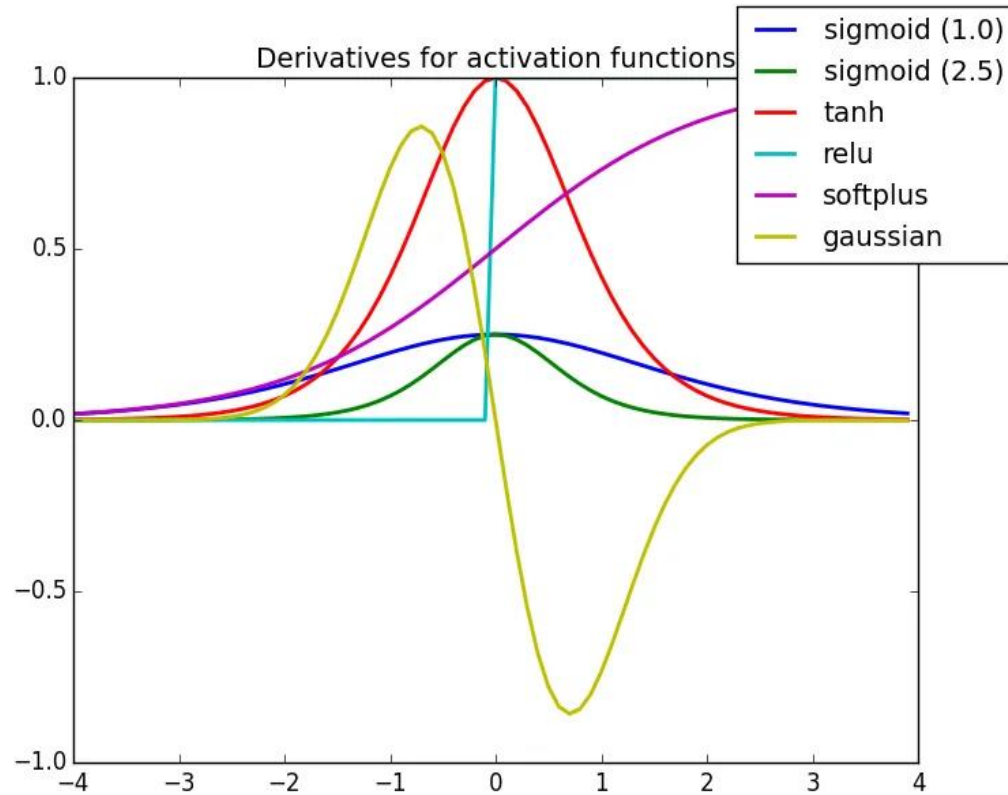


Figure 9: Derivative of Activation Functions

Statistical model

We applied different ML models: SVM, NB, LR, KNN, and RF to predict PCOS.

Data Acquisition

We used the PCOS dataset from Kaggle, which includes 541 instances and 41 attributes.

Data Preprocessing

Essential libraries such as pandas, numpy, and scikit-learn modules are imported for data manipulation, analysis, and machine learning. The dataset, loaded from an Excel file, undergoes basic exploratory data analysis, including handling missing values through mean imputation. To address class imbalance, the minority class is upsampled using the resample function. Features and target variables are separated, followed by a split of the upsampled dataset into training and testing sets. Standard scaling is then applied to ensure optimal model performance.

Models Implementation

PART I

We explore the performance of Random Forest and Support Vector Machine (SVM) classifiers on both the upsampled and downsampled datasets.

Random Forest Classifier (Upsampled Dataset):

- A Random Forest Classifier is trained with 200 estimators on the upsampled training data (X_{train} , y_{train}).
- Predictions are made on the upsampled test set (X_{test}), and performance metrics such as accuracy, precision, recall, and F1 score are calculated and displayed.

Support Vector Machine (Upsampled Dataset):

- An SVM classifier is trained on the upsampled training data (X_{train} , y_{train}).
- Predictions are made on the upsampled test set, and performance metrics are computed and printed.

Downsampling the Majority Class:

- The majority class ('PCOS (Y/N)' = 0) is downsampled to address class imbalance using the resample function.
- The downsampled dataset is created by combining the downsampled majority class with the minority class.

Random Forest Classifier (Downsampled Dataset):

- A new Random Forest Classifier is trained with 200 estimators on the downsampled training data (XD_{train} , YD_{train}).
- Predictions are made on the downsampled test set (XD_{test}), and performance metrics are displayed.

Support Vector Machine (Downsampled Dataset):

- An SVM classifier is trained on the downsampled training data (XD_{train} , YD_{train}).
- Predictions are made on the downsampled test set, and performance metrics are printed.

This comprehensive analysis on both upsampled and downsampled datasets using two different classifiers allows for a comparison of their performances in handling class imbalance. The metrics include accuracy, precision, recall, and F1 score, providing a thorough evaluation of the models' effectiveness in predicting PCOS.

PART II

We address class imbalance using the Synthetic Minority Over-sampling Technique (SMOTE) and evaluate the performance of various classifiers on the balanced datasets. Here's a breakdown of the code:

Dataset Splitting:

The original dataset (data) is split into training (df_train) and testing (df_test) sets.

SMOTE Oversampling:

The SMOTE technique is applied to the training and testing sets to oversample the minority class and balance the class distribution.

Random Forest Classifier:

- A Random Forest Classifier is trained on the SMOTE-upsampled training data (X_train_res_std, Y_train_res).
- Predictions are made on the SMOTE-upsampled test set (X_test_res_std), and performance metrics (accuracy, precision, recall, F1 score, and classification report) are printed.

Support Vector Machine (SVM):

- An SVM classifier is trained on the SMOTE-upsampled training data.
- Predictions are made on the SMOTE-upsampled test set, and performance metrics are displayed.

Logistic Regression:

- Logistic regression is applied to the entire dataset (without SMOTE) for comparison.

- The model is trained on the SMOTE-upsampled training data, and predictions are made on the original test set (XS_test).

Performance metrics are printed, including accuracy, precision, recall, F1 score, and the classification report. The aim is to demonstrate the impact of SMOTE on the performance of different classifiers, specifically Random Forest, SVM, and Logistic Regression, in the context of PCOS prediction. The evaluation metrics provide insights into how well each model handles the imbalanced class distribution.

PART III

WE focus on addressing class imbalance using two different sampling techniques: ADASYN (Adaptive Synthetic Sampling) and SMOTE ENN (SMOTE + Edited Nearest Neighbors). It then evaluates the performance of a logistic regression model on the balanced datasets. Here's a breakdown of the code:

ADASYN Sampling:

- The training data (df_train) is prepared by removing irrelevant columns.
- ADASYN is applied to the training set to oversample the minority class.
- The logistic regression model is trained on the ADASYN-upsampled training data, and predictions are made on the original test set (X_test).
- Performance metrics (accuracy, precision, recall, F1 score, and classification report) are printed.

SMOTE ENN Sampling:

- Similar to the previous step, SMOTEENN (a combination of SMOTE and Edited Nearest Neighbors) is applied to the training set.
- The logistic regression model is trained on the SMOTEENN-upsampled training data, and predictions are made on the original test set.

These techniques aim to balance the class distribution and improve the performance of the logistic regression model in predicting PCOS. The choice between ADASYN and SMOTE ENN depends on the specific characteristics of the dataset and the desired trade-offs between precision, recall, and overall accuracy.

PART IV

We introduces two additional classifiers, Gaussian Naive Bayes and k-Nearest Neighbors (KNN), for predicting PCOS.

Gaussian Naive Bayes:

- A Gaussian Naive Bayes classifier is built using the GaussianNB class from scikit-learn.
- The model is trained on the training data (X_train, y_train).
- Predictions are made on the test set (X_test), and the performance metrics (accuracy, precision, recall, F1 score) are computed and printed.

k-Nearest Neighbors (KNN):

- A KNN classifier is instantiated with n_neighbors=3.
- The model is trained on the training data (X_train, y_train).
- Predictions are made on the test set (X_test), and the performance metrics (accuracy, precision, recall, F1 score) are computed and printed.

This code allows for the comparison of the performance of Gaussian Naive Bayes and k-Nearest Neighbors classifiers alongside the previously implemented Random Forest, Support Vector Machine, and Logistic Regression models for PCOS prediction.

RESULT

Results for Statistical Model

Model	Acc(%)	Pre(%)	Rec(%)	F1
RF (Unsampled Data)	94.4	94.2	94.2	94.2
RF (Downsampled data)	85.9	86.11	86.11	86.11
RF (SMOTE Data)	75.7	77.27	72.85	75
LR (SMOT Data)	88.28	77.56	90.3	83.46
LR (ADASYN)	90.3	88.7	91.3	90
LR (SMOT ENN)				
NB	50.3	48.8	89.85	63.2
KNN	86.2	48.8	89.8	63.2
SVM (unsampled data)	91	92.4	88.4	90.3
SVM (downsampled data)	78.8	86.2	69.44	76.92
SVM (SMOTE Data)	77.8	79.1	75.7	77.37

Results for CNN Model

Accuracy and Loss graphs for Training and Validation



Figure 10: for Relu activation function



Figure 11: first image to predict

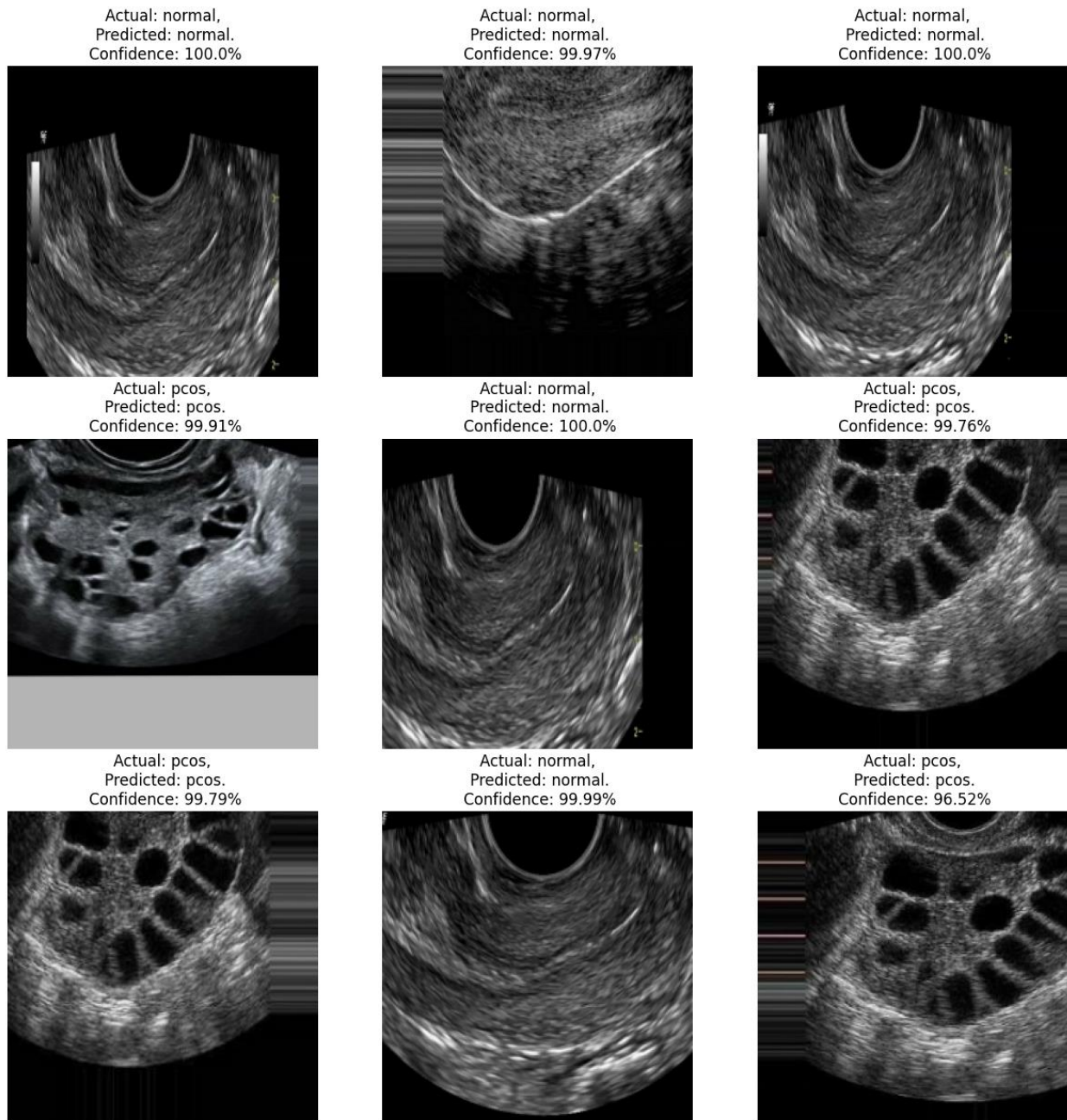


Figure 12: Predicted images



Figure 13: sigmoid function

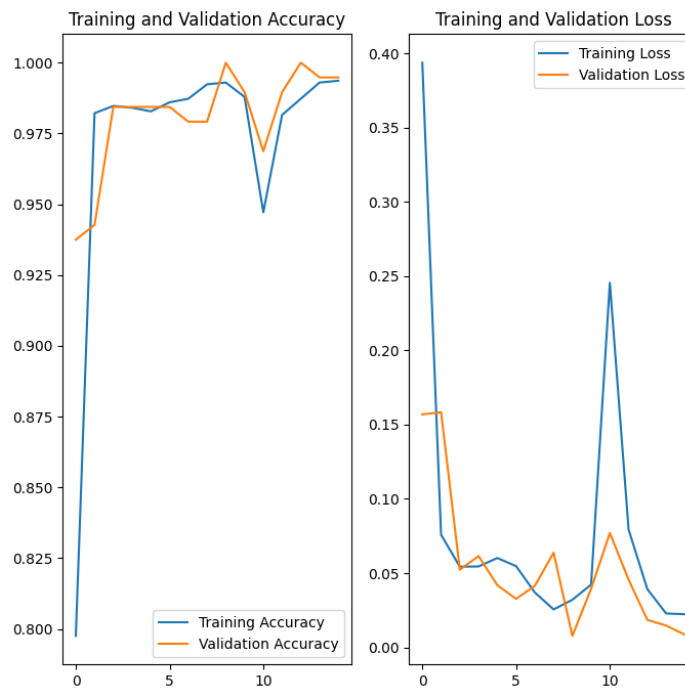


Figure 14: linear function

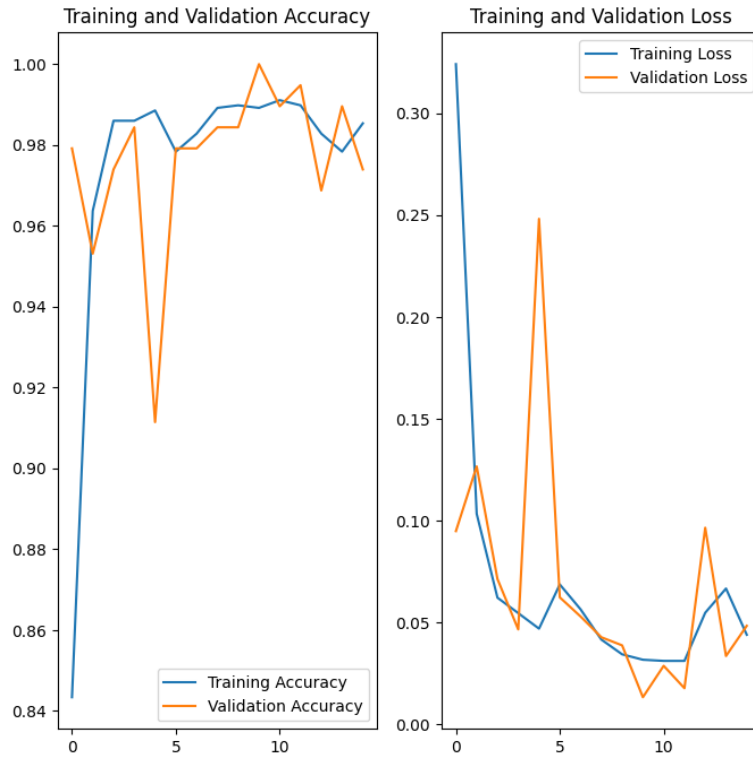


Figure 14: Tanh

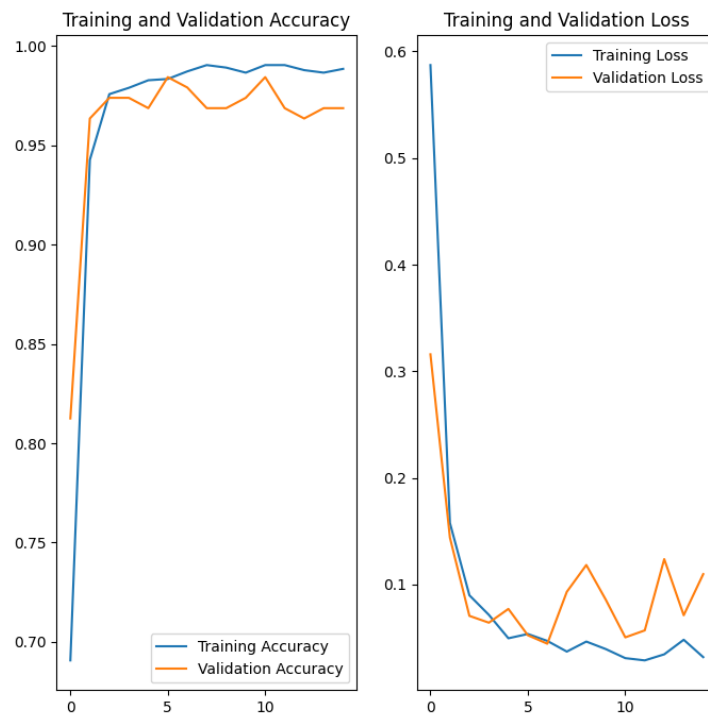


Figure 14: Leaky Relu

API Development with FastAPI

To facilitate the seamless integration of the PCOS detection model into existing healthcare systems, an API was developed using FastAPI, a modern web framework for building APIs with Python. The API exposes endpoints for receiving medical images as input and provides predictions regarding the likelihood of PCOS. The API documentation details the expected input formats, and examples showcase the simplicity of interacting with the system programmatically. This choice of FastAPI enables efficient and scalable deployment of the PCOS detection model. The API developed using FastAPI is locally hosted, providing a practical solution for PCOS detection within a controlled environment. The deployment involves setting up a local server, ensuring accessibility to authorized users. The integration process seamlessly connects the API with existing data pipelines and patient management systems within the local infrastructure.

The image shows the FastAPI API documentation interface. At the top, the 'FastAPI' logo is displayed with version '0.1.0' and 'OAS 3.1' badges, and a link to '/openapi.json'. Below this, a 'default' section is visible. The interface lists three endpoints: 'GET / Index', 'GET /{name} Get Name', and 'POST /predict Predict'. The 'POST /predict Predict' endpoint is selected and expanded. It shows 'Parameters' as 'No parameters'. The 'Request body' is marked as 'required' and set to 'multipart/form-data'. A file upload field is present, labeled 'file * required' and 'string(binary)', with a 'Choose File' button and the filename 'img3.jpg'. At the bottom, there is a 'Servers' section with a dropdown menu showing a single server '/'. An 'Execute' button is located at the bottom left, and a 'Clear' button is at the bottom right.

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@img1.jpg;type=Image/jpeg'
```

Request URL

http://127.0.0.1:8000/predict

Server response

Code Details

200

Response body

```
{
  "prediction": "Non-Infected"
}
```

Response headers

```
content-length: 29
content-type: application/json
date: Sun, 17 Dec 2023 18:52:28 GMT
server: uvicorn
```

Responses

Code Description

Links

200

Successful Response

No links

Media type

application/json

Controls Accepted header.

Example Value | Schema

```
"string"
```

422

Validation Error

No links

Media type

application/json

Example Value | Schema

```
{
  "detail": [
    {
      "loc": [
        "string",
        "mg": "string",
        "type": "string"
      ]
    }
  ]
}
```

4. CONCLUSION

PCOS is a widespread gynecological disease which has been seen to be hereditary, and comes with symptoms including hormonal disorders and ovulatory dysfunction, with knock-on effects spanning the endocrine and metabolic state of the affected female. Traditional means towards the diagnosis of the condition can carry shortcomings that the introduction of an AI-driven prediction machine can help towards solving, with a view and emphasis towards the early identification of the disease in its early and latent stages. The main objective of our paper is to provide an early detection model for PCOS. The early detection of PCOS reduces the possibility of long-term complications. Several ML utilized to build the proposed stacking ensemble ML model. It combines diverse ML models (LR, RF, DT, NB, SVM, KNN) at the base learner level with RF at the meta-learner level to improve the performance of a single ML.

The implementation of a Convolutional Neural Network (CNN) model for the prediction of Polycystic Ovary Syndrome (PCOS) demonstrates promising results in leveraging advanced deep learning techniques for medical diagnosis. Through the utilization of image data, the CNN model exhibits a high level of accuracy of 99% in identifying patterns and features associated with PCOS.

5. REFERENCES

- [1] IF. Stein, ML. Leventhal, Amenorrhea associated with bilateral polycystic ovaries, Am. J. Obstet. Gynecol. 29 (2) (1935) 181–191.
- [2] AH. Balen, GS. Conway, G. Kaltsas, K. Techatrasak, PJ. Manning, C. West, et al., Polycystic ovary syndrome: the spectrum of the disorder in 1741 patients, Hum Reprod. 10 (8) (1995) 2107–2111.
- [3] C. Farquhar, Introduction and history of polycystic ovary syndrome, in: GT Kovacs, R Norman (Eds.), Polycystic Ovary Syndrome [Internet], second ed., Cambridge University Press, 2001, pp. 4–241].
- [4] <https://www.geeksforgeeks.org/>
- [5] <https://scikit-learn.org/stable/>
- [6] <https://www.wikipedia.org/>

APPENDIX

CODE FOR STATISTICAL MODEL

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
#importing dataset
dataset = pd.read_excel("PCOS_data.xlsx")
dataset.head()
dataset.info()
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = "mean")
#checking to see if there's any null variables
dataset.isnull().sum()
print ("Number of rows in the dataset : ",dataset.shape[0])
print ("Number of Columns in the dataset : ",dataset.shape[1])
print ("Number of Features : \n",dataset.columns.tolist())
print ("Missing values : ", dataset.isnull().sum().values.sum())
print ("Unique values : \n",dataset.nunique())
# Yes = 1, No = 0
```

```

dataset['PCOS (Y/N)'].value_counts()
from sklearn.model_selection import train_test_split

#splitting the testing and training data
df = dataset

# Separate majority and minority classes
df_majority = df[df['PCOS (Y/N)'] == 0]
df_minority = df[df['PCOS (Y/N)'] == 1]
df_majority.shape, df_minority.shape
from sklearn.utils import resample
df_minority_upsampled = resample(df_minority,
                                replace=True, # sample with replacement
                                n_samples=362, # to match majority class #length of df_majority
                                random_state=42)
df_upsampled = pd.concat([df_majority, df_minority_upsampled])
df_upsampled
df_up = df_upsampled.sample(frac=1)
df_up['PCOS (Y/N)'].value_counts()
#Now separate the dataset as response variable and feature variables
X = df_up.drop(['Sl. No', 'Patient File No.', 'PCOS (Y/N)'], axis = 1)
y = df_up['PCOS (Y/N)']
#Train and Test splitting of data
#train_test_split: This is a function from a library like scikit-learn X is the feature matrix
with the features ofc,
#y prediction thing, 0.2 means train:test ratio is 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 123)
#Applying Standard scaling to get optimized result
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

rfc = RandomForestClassifier(n_estimators=200, random_state = 420)
rfc.fit(X_train, y_train)
pred_rfc = rfc.predict(X_test)
#Let's see how our model performed

acc = accuracy_score(y_test, pred_rfc)
prec = precision_score(y_test, pred_rfc)

```

```
recall = recall_score(y_test, pred_rfc)
f1 = f1_score(y_test, pred_rfc)
```

```
print("Accuracy : ", acc)
print("Precision Score : ", prec)
print("Recall Score : ", recall)
print("F1 Score : ", f1)
print(classification_report(y_test, pred_rfc))
from sklearn import svm
```

```
clf=svm.SVC(random_state = 421)
clf.fit(X_train, y_train)
pred_clf = clf.predict(X_test)
```

```
#Let's see how our model performed
acc = accuracy_score(y_test, pred_clf)
prec = precision_score(y_test, pred_clf)
recall = recall_score(y_test, pred_clf)
f1 = f1_score(y_test, pred_clf)
```

```
print("Accuracy : ", acc)
print("Precision Score : ", prec)
print("Recall Score : ", recall)
print("F1 Score : ", f1)
print(classification_report(y_test, pred_clf))
df_majority_downsampled = resample(df_majority,
                                   replace=True, # sample without replacement
                                   n_samples=176, # to match minority class
                                   random_state=123) # reproducible results
```

```
# Combine minority class with downsampled majority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])
df_down = df_downsampled.sample(frac=1)
df_down['PCOS (Y/N)'].value_counts()
# Dividing the dataset into two part one having only the target value and other having
all other columns
#Now seperate the dataset as response variable and feature variabes
X_down = df_down.drop(['Sl. No', 'Patient File No.', 'PCOS (Y/N)'], axis = 1)
```

```

Y_down = df_down['PCOS (Y/N)']
#Train and Test splitting of data
#train_test_split: This is a function from a library like scikit-learn X is the feature matrix
with the features ofc,
#y prediction thing, 0.2 means train:test ratio is 80:20
XD_train, XD_test, YD_train, YD_test = train_test_split(X_down, Y_down, test_size = 0.2,
random_state = 123)
#Applying Standard scaling to get optimized result
sc = StandardScaler()
XD_train = sc.fit_transform(XD_train)
XD_test = sc.fit_transform(XD_test)
rfc = RandomForestClassifier(n_estimators=200, random_state = 420)
rfc.fit(XD_train, YD_train)
pred_rfc = rfc.predict(XD_test)

#Let's see how our model performed

acc = accuracy_score(YD_test, pred_rfc)
prec = precision_score(YD_test, pred_rfc)
recall = recall_score(YD_test, pred_rfc)
f1 = f1_score(YD_test, pred_rfc)

print("Accuracy : ", acc)
print("Precision Score : ", prec)
print("Recall Score : ", recall)
print("F1 Score : ", f1)
print(classification_report(YD_test, pred_rfc))
from sklearn import svm

clf=svm.SVC()
clf.fit(XD_train, YD_train)
pred_clf = clf.predict(XD_test)

#Let's see how our model performed
acc = accuracy_score(YD_test, pred_clf)
prec = precision_score(YD_test, pred_clf)
recall = recall_score(YD_test, pred_clf)
f1 = f1_score(YD_test, pred_clf)

```

```

print("Accuracy : ", acc)
print("Precision Score : ", prec)
print("Recall Score : ", recall)
print("F1 Score : ", f1)
print(classification_report(YD_test, pred_clf))

#splitting the testing and training data
df = data
df_test = df[430:] #430 k baad wale
df_train = df[:430]

df_train['PCOS (Y/N)'].value_counts()
X_train_smote = df_train.drop(['Sl. No', 'Patient File No.', 'PCOS (Y/N)'],axis=1)
Y_train_smote = df_train['PCOS (Y/N)']
X_train_smote.shape,Y_train_smote.shape

X_test_smote = df_test.drop(['Sl. No', 'Patient File No.', 'PCOS (Y/N)'],axis=1)
Y_test_smote = df_test['PCOS (Y/N)']
X_test_smote.shape,Y_test_smote.shape
!pip install imblearn
pip install --upgrade imbalanced-learn scikit-learn
!pip install scikit-learn
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
X_train_res, Y_train_res = sm.fit_resample(X_train_smote, Y_train_smote.ravel())
X_train_res.shape, Y_train_res.shape
len(Y_train_res[Y_train_res==0]),len(Y_train_res[Y_train_res==1])
sm = SMOTE(random_state = 3)
X_test_res, Y_test_res = sm.fit_resample(X_test_smote, Y_test_smote.ravel())
X_test_res.shape, Y_test_res.shape
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Standardize the features for both training and testing data
scaler = StandardScaler()
X_train_res_std = scaler.fit_transform(X_train_res)
X_test_res_std = scaler.transform(X_test_res)
rfc = RandomForestClassifier(random_state=42)

```

```
rfc.fit(X_train_res_std, Y_train_res)
```

```
# Predict on the standardized test data
```

```
y_pred_smote = rfc.predict(X_test_res_std)
```

```
#Let's see how our model performed
```

```
acc = accuracy_score(Y_test_res, y_pred_smote)
```

```
prec = precision_score(Y_test_res, y_pred_smote)
```

```
recall = recall_score(Y_test_res, y_pred_smote)
```

```
f1 = f1_score(Y_test_res, y_pred_smote)
```

```
print("Accuracy : ", acc)
```

```
print("Precision Score : ", prec)
```

```
print("Recall Score : ", recall)
```

```
print("F1 Score : ", f1)
```

```
print(classification_report(Y_test_res, y_pred_smote))
```

```
from sklearn import svm
```

```
clf = svm.SVC()
```

```
clf.fit(X_train_res_std, Y_train_res)
```

```
# Predict on the standardized test data
```

```
y_pred_smote2 = clf.predict(X_test_res_std)
```

```
#Let's see how our model performed
```

```
acc = accuracy_score(Y_test_res, y_pred_smote2)
```

```
prec = precision_score(Y_test_res, y_pred_smote2)
```

```
recall = recall_score(Y_test_res, y_pred_smote2)
```

```
f1 = f1_score(Y_test_res, y_pred_smote2)
```

```
print("Accuracy : ", acc)
```

```
print("Precision Score : ", prec)
```

```
print("Recall Score : ", recall)
```

```
print("F1 Score : ", f1)
```

```
print(classification_report(Y_test_res, y_pred_smote2))
```

```
from sklearn.linear_model import LogisticRegression
```

```
XS_test = data.drop(['Sl. No', 'Patient File No.', 'PCOS (Y/N)'], axis = 1) #axis=1 means we  
are removing a column
```

```
YS_test = data['PCOS (Y/N)']
```

```
logistic_model = LogisticRegression(penalty='l2', C=1.0, solver='liblinear').fit(X_train_res,  
y_train_res)
```

```
yS_pred = logistic_model.predict(XS_test)
```

```
acc = accuracy_score(YS_test, yS_pred)
```

```
prec = precision_score(YS_test, yS_pred)
```

```
recall = recall_score(YS_test, yS_pred)
```

```
f1 = f1_score(YS_test, yS_pred)
```

```
print("accuracy_score : ", acc)
```

```
print("precision_score : ", prec)
```

```
print("recall_score : ", recall)
```

```
print("f1_score : ", f1)
```

```
target_names = ['class 0', 'class 1']
```

```
print(classification_report(YS_test, yS_pred, target_names=target_names))
```

```
cf_matrix=confusion_matrix(yS_pred, YS_test)
```

```
Cf_matrix
```

```
X_train_adas=df_train.drop(['Sl. No', 'Patient File No.','PCOS (Y/N)'],axis=1)
```

```
Y_train_adas=df_train['PCOS (Y/N)']
```

```
X_train_adas.shape,Y_train_adas.shape
```

```
from imblearn.over_sampling import ADASYN
```

```
sm = ADASYN(random_state = 2)
```

```
X_train_ada, y_train_ada = sm.fit_resample(X_train_adas, Y_train_adas.ravel())
```

```
X_train_ada.shape, y_train_ada.shape
```

```
len(y_train_ada[y_train_ada==0]),len(y_train_ada[y_train_ada==1])
```

```
from sklearn.linear_model import LogisticRegression
```

```
logistic_model = LogisticRegression(penalty='l2', C=1.0,  
solver='liblinear').fit(X_train_ada, y_train_ada)
```

```
y_pred = logistic_model.predict(X_test)
```

```
acc = accuracy_score(Y_test, y_pred)
```

```
prec = precision_score(Y_test, y_pred)
```

```
recall = recall_score(Y_test, y_pred)
```

```
f1 = f1_score(Y_test, y_pred)
```

```
print("accuracy_score : ", acc)
```

```
print("precision_score : ", prec)
```

```

print("recall_score : ", recall)
print("f1_score : ", f1)

target_names = ['class 0', 'class 1']
print(classification_report(Y_test, y_pred, target_names=target_names))
cf_matrix=confusion_matrix(y_pred, Y_test)
cf_matrix
X_train_se=df_train.drop(['Sl. No', 'Patient File No.','PCOS (Y/N)'],axis=1)
Y_train_se=df_train['PCOS (Y/N)']
X_train_se.shape,Y_train_se.shape
from imblearn.combine import SMOTEENN
sm = SMOTEENN(random_state = 2)
X_train_sen, y_train_sen = sm.fit_resample(X_train_se, Y_train_se.ravel())
X_train_sen.shape,y_train_sen.shape
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(penalty='l2', C=1.0,
solver='liblinear').fit(X_train_sen, y_train_sen)
y_pred = logistic_model.predict(X_test)

acc = accuracy_score(Y_test, y_pred)
prec = precision_score(Y_test, y_pred)
recall = recall_score(Y_test, y_pred)
f1 = f1_score(Y_test, y_pred)
print("accuracy_score : ", acc)
print("precision_score : ", prec)
print("recall_score : ", recall)
print("f1_score : ", f1)

target_names = ['class 0', 'class 1']
print(classification_report(Y_test, y_pred, target_names=target_names))
cf_matrix=confusion_matrix(y_pred, Y_test)
cf_matrix

from sklearn.naive_bayes import GaussianNB

# Build a Gaussian Classifier
model = GaussianNB()

```

```
# Model training
model.fit(X_train, y_train)

# Predict Output
predicted = model.predict(X_test)

y_testt = np.array(df['PCOS (Y/N)'])

print(y_testt)
print(predicted)
```

```
acc = accuracy_score(y_test, predicted)
prec = precision_score(y_test, predicted)
recall = recall_score(y_test, predicted)
f1 = f1_score(y_test, predicted)
print("accuracy_score : ", acc)
print("precision_score : ", prec)
print("recall_score : ", recall)
print("f1_score : ", f1)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(preds)
acc = accuracy_score(y_test, preds)
prec = precision_score(y_test, preds)
recall = recall_score(y_test, preds)
f1 = f1_score(y_test, preds)
print("accuracy_score : ", acc)
print("precision_score : ", prec)
print("recall_score : ", recall)
print("f1_score : ", f1)
```

CODE FOR CNN MODEL

```
from google.colab import drive
drive.mount('/content/drive/')
```

PCOS Detection

```
#Imports necessary libraries
import os #Provides a way to interact with the operating system, including
file and directory manipulation.
import cv2 #(OpenCV): Used for image processing tasks.
from tqdm import tqdm # A library for displaying progress bars during
iterations.

datadir = r'/content/drive/MyDrive/dataset' # Path to directory with
class subdirectories holding the image files
bad_img_list = [] #A list that will store the paths of defective images.
classes = sorted(os.listdir(datadir)) #A list of classes within the
specified data directory.

#Defines a function is_infected_image
def is_infected_image(image_path): #Takes the path of an image file as an
argument.
    try:
        index = image_path.rfind('.') # Find the rightmost . in
image_path
        ext = image_path[index + 1:].lower() # Get the file's extension
and convert it to lowercase
        good_ext = ['jpg', 'jpeg', 'bmp', 'png'] # List of allowable
extensions for image_dataset_from_directory
        if ext not in good_ext:
            return True # Return True if the extension is not allowed
        img = cv2.imread(image_path) # Read in the image
        shape = img.shape # Get the image shape (height, width) or
(height, width, channels)
        count = len(shape)
        if count == 2: # If shape is (width, height), the image is
single-channel
            channels = 1
```

```

        else:
            channels = shape[2] # Shape is (width, height, channels)
            if channels == 2:
                return True # Return True if the image has 2 channels
            except:
                return True # Return True if any other exception occurs
            return False # Return False if the image is not infected

# Iterate through each class and its images
for klass in classes:
    classpath = os.path.join(datadir, klass)
    flist = sorted(os.listdir(classpath)) # List of files in the current
class

    for f in tqdm(flist, ncols=100, unit='files', colour='blue',
desc=klass):
        fpath = os.path.join(classpath, f) # Path to the image file

        if is_infected_image(fpath):
            # Append the path to the list of bad_img_list
            bad_img_list.append(fpath)
            os.remove(fpath) # Remove the infected image file

if len(bad_img_list) > 0:
    print('Below is a list of defective image filepaths that were
removed:')
    for f in bad_img_list:
        print(f)

```

Import all the Dependencies

```

import tensorflow as tf
from tensorflow.keras import models, layers

import matplotlib.pyplot as plt
from IPython.display import HTML

```

Set all the Constants

```

BATCH_SIZE = 32 #standard batch size is 32
IMAGE_SIZE = 224 #image_size -> 224 * 224
CHANNELS=3

```

EPOCHS=15

Import data into tensorflow dataset object

```
#tf.keras.preprocessing.image_dataset_from_directory -> this call will
load all the images into your tensor/dataset
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/dataset",
    seed=123,
    shuffle=True, #randomly shuffle the images
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
#to obtain class names that is folder names
class_names = dataset.class_names
Class_names

#68 * 32 -> 2176 and last batch is not perfect. so, it is showing more
files
len(dataset)
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape) #rgb -> 3 channels
    print(labels_batch.numpy()) #to convert tensor into numpy

#to print first image
for image_batch, labels_batch in dataset.take(1):
    print(image_batch[0])

#Each number in the matrix represents the intensity of a pixel at a
specific location in the image.
#every number is btw 0 to 255
for image_batch, labels_batch in dataset.take(1):
    print(image_batch[0].numpy())

for image_batch, labels_batch in dataset.take(1):
    print(image_batch[0].shape)
for image_batch, label_batch in dataset.take(1):
    plt.imshow(image_batch[0].numpy()) #imshow expects 3d array
    #it is in float
```

```

    #image_batch[0].numpy(): Converts the first image from a TensorFlow
    tensor to a NumPy array. The numpy() method is used to extract the
    numerical values from the TensorFlow tensor.

for image_batch, label_batch in dataset.take(1):
    plt.imshow(image_batch[0].numpy().astype("uint8"))
    plt.axis("off") #to hide axis
    #Converting to uint8 is a common practice when working with images
    because it ensures that the pixel values are within the expected range for
    color representation.

plt.figure(figsize=(10, 10)) #to increase the area of image
for image_batch, labels_batch in dataset.take(1):
    for i in range(12): #to print 12 images
        ax = plt.subplot(3, 4, i + 1) #to make a matrix of 3 * 4
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]]) #to display label on image
        plt.axis("off")

```

Function to Split Dataset

```

len(dataset)

#80% ==> training
#20% ==> 10% validation, 10% test
train_size = 0.8
len(dataset)*train_size
# 68 * 0.8 ==> 54

train_ds = dataset.take(54) #take first 54 samples
len(train_ds)

test_ds = dataset.skip(54) #skip first 54 samples
len(test_ds)

val_size=0.1
len(dataset)*val_size

val_ds = test_ds.take(6)
len(val_ds)

```

```

test_ds = test_ds.skip(6)
len(test_ds)

#creating a nice looking python function
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1,
test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle: #shuffling before splitting
        ds = ds.shuffle(shuffle_size, seed=12) #seed for predictability
        #if same seed everytime, we'll see same results

    train_size = int(train_split * ds_size) #to get int
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

len(train_ds)

len(val_ds)

len(test_ds)

#tf.data.Dataset.prefetch(1) -> 1 is how many batches i want to prefetch
#tf.data.Dataset.prefetch(AUTOTUNE) -> we'll let tensorflow framework
decide for itself ie. how many batches it wants to load in advance
#prefetch -> cpu and gpu can work in parallel -> to make optimal use of
hardware resources
#tf.data.Dataset.cache() -> open, read, map and train for first epoch but
for the second epoch, it will just train -> will keep the image
# in memory -> improve the performance of the pipeline

```

```

train_ds =
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds =
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds =
test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

#pre-processing
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/224) #to get value
between 0 and 1,
])

data_augmentation = tf.keras.Sequential ([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2),
])

#tf.keras.Sequential -> helps to form a cluster of a layer that is
linearly stacked into tf. keras. Model.

#Data Augmentation to address overfitting
#from existing training samples we try to generate new samples
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)

#    Merits of pooling -
#    - reduces dimensions and computation
#    - reduces overfitting as there are less parameters
#    - model is tolerant towards variations, distortions

#    Merits of CNN:-
#    - Connections sparsity reduces overfitting.
#    - Conv + Pooling gives location invariant feature detection
#    - Parameter sharing

#    Merits of ReLU:-
#    - Introduces nonlinearity
#    - Speeds up training, faster to compute

```

```
# CNN by itself doesn't take care of rotation and scale
# - you need to have rotated, scaled samples in training dataset
# - if you don't have such samples than use data augmentation methods
  to generate new rotated/
# scaled samples from existing training samples
```

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3
```

```
# -> 32 layers, relu is fast to compute
# softmax activation function will normalise the probability of
  classes
```

```
model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])
```

```
model.build(input_shape=input_shape)
```

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3
```

```
model2 = models.Sequential([
    resize_and_rescale,
```

```

        layers.Conv2D(32, kernel_size = (3,3), activation='sigmoid',
input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, kernel_size = (3,3), activation='sigmoid'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, kernel_size = (3,3), activation='sigmoid'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='sigmoid'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='sigmoid'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='sigmoid'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(64, activation='sigmoid'),
        layers.Dense(n_classes, activation='softmax'),
    ])

```

```

model2.build(input_shape=input_shape)

```

```

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

```

```

model3 = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='linear',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='linear'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='linear'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='linear'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='linear'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='linear'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='linear'),
])

```

```

        layers.Dense(n_classes, activation='softmax'),
    ])

model3.build(input_shape=input_shape)

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model4 = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='tanh',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='tanh'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='tanh'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='tanh'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='tanh'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='tanh'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='tanh'),
    layers.Dense(n_classes, activation='softmax'),
])

model4.build(input_shape=input_shape)

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model5 = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='LeakyReLU',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='LeakyReLU'),
    layers.MaxPooling2D((2, 2)),

```

```

        layers.Conv2D(64, kernel_size = (3,3), activation='LeakyReLU'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='LeakyReLU'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='LeakyReLU'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='LeakyReLU'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(64, activation='LeakyReLU'),
        layers.Dense(n_classes, activation='softmax'),
    ])

```

```

model5.build(input_shape=input_shape)

```

```

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

```

```

model6 = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu',
input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='sigmoid'),
])

```

```

model6.build(input_shape=input_shape)

```

```
model.summary()
#total parameters = total weight

model2.summary()

model3.summary()

model4.summary()

model5.summary()

model6.summary()

model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

model2.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

model3.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

model4.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

model5.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
```

```
        metrics=['accuracy']
    )

model6.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

history = model.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose=1,
    epochs=15,
)

history2 = model2.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose=1,
    epochs=15,
)

history3 = model3.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose=1,
    epochs=15,
)

history4 = model4.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose=1,
    epochs=15,
)
```

```
history5 = model5.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose=1,
    epochs=15,
)

history6 = model6.fit(
    train_ds,
    batch_size = BATCH_SIZE,
    validation_data = val_ds,
    verbose=1,
    epochs=15,
)

scores = model.evaluate(test_ds)

scores2 = model2.evaluate(test_ds)

scores3 = model3.evaluate(test_ds)

scores4 = model4.evaluate(test_ds)

scores5 = model5.evaluate(test_ds)

scores6 = model6.evaluate(test_ds)

scores    #Scores is just a list containing loss and accuracy value

Scores2

scores3

scores4

scores5

scores6
```

```
history.params

history2.params

history3.params

history.history.keys()

history2.history.keys()

history3.history.keys()

type(history.history['loss'])

history.history['loss'][:5] # show loss for first 5 epochs

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

acc2 = history2.history['accuracy']
val_acc2 = history2.history['val_accuracy']

loss2 = history2.history['loss']
val_loss2 = history2.history['val_loss']

acc3 = history3.history['accuracy']
val_acc3 = history3.history['val_accuracy']

loss3 = history3.history['loss']
val_loss3 = history3.history['val_loss']

acc4 = history4.history['accuracy']
val_acc4 = history4.history['val_accuracy']

loss4 = history4.history['loss']
val_loss4 = history4.history['val_loss']
```

```

acc5 = history5.history['accuracy']
val_acc5 = history5.history['val_accuracy']

loss5 = history5.history['loss']
val_loss5 = history5.history['val_loss']

acc6 = history6.history['accuracy']
val_acc6 = history6.history['val_accuracy']

loss6 = history6.history['loss']
val_loss6 = history6.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc2, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc2, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss2, label='Training Loss')
plt.plot(range(EPOCHS), val_loss2, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

```

```

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc3, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc3, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss3, label='Training Loss')
plt.plot(range(EPOCHS), val_loss3, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc4, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc4, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss4, label='Training Loss')
plt.plot(range(EPOCHS), val_loss4, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc5, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc5, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss5, label='Training Loss')
plt.plot(range(EPOCHS), val_loss5, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

```

```

plt.show()

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc6, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc6, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss6, label='Training Loss')
plt.plot(range(EPOCHS), val_loss6, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:", class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:", class_names[np.argmax(batch_prediction[0])])

def predict(model, img):
    img_array =
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence

```

```
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted:
{predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```