ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)
ORGANISATION OF ISLAMIC COOPERATION (OIC)
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Project Title :

Comparative Analysis of Machine Learning and Deep Learning Models: Evaluating Regression Based Covid Fatality Predictions and CNN Architectures for Image Nearest Neighbours

Group no. A5

Project Members: (i) Masrur Ibne Ali (200021131)
(ii) Mustafid Bin Mostofa (200021141)
(iii) Imtiaz Tanweer Rahim (200021150)

Course Title: Artificial Intelligence

Course No. EEE-4709

Instructor Name: Md. Arefin Emon

Submission Date: 21.03.2025

ABSTRACT:

This study compares machine learning (ML) models for COVID-19 death rate prediction and convolutional neural networks (CNNs) for image-based nearest-neighbor retrieval. K-Nearest Neighbors, Logistic Regression, and Random Forest were evaluated on a dataset of 1048575 patients, with Random Forest achieving the highest accuracy (0.9497) and Logistic Regression excelling in recall. In the deep learning segment, GoogLeNet, ResNet-101, and ZFNet were analyzed for image similarity search using pre-trained embeddings. ResNet demonstrated superior accuracy and generalization. The findings provide insights into model selection for healthcare and computer vision applications, with future improvements focusing on hyperparameter tuning and advanced deep learning architectures.

INTRODUCTION

BACKGROUND AND MOTIVATION

The COVID-19 pandemic has highlighted the critical role of data-driven decision-making in healthcare, epidemiology, and public policy. Machine learning (ML) models have been widely employed to predict infection rates, disease severity, and patient outcomes. However, the performance of different ML algorithms varies based on dataset structure, feature engineering, and classification criteria. While KNN, Logistic Regression, and Random Forest Classification are commonly used, their comparative effectiveness in COVID-related fatality predictions remains an open question. Understanding these variations is crucial for selecting the most suitable model for different predictive tasks.

Simultaneously, deep learning has revolutionized image analysis, particularly with the advent of Convolutional Neural Networks (CNNs). In medical imaging and other visual data applications, CNN-based models such as GoogLeNet, ZFNet, and ResNet-101 have demonstrated remarkable feature extraction capabilities. One of their key applications is nearest-neighbor search, which helps in tasks like disease diagnosis, anomaly detection, and image retrieval.

Why is this project important?

COVID-19 datasets exhibit complex patterns, making it essential to determine which ML model best captures these trends. While KNN is commonly used for classification tasks, logistic regression and random forest offer better classification capabilities. Understanding their trade-offs is necessary for improving model selection. A systematic evaluation will help in identifying the most efficient model based on predictive accuracy, computational cost, and generalizability.

GoogLeNet, ZFNet, and ResNet-101 differ in depth, filter sizes, and computational efficiency, but their performance trade-offs in nearest-neighbor search remain unclear. Understanding these differences will assist in selecting the best architecture for applications such as medical image retrieval, automated diagnosis, and content-based image search.This study aims to address these gaps by conducting a comparative analysis of both traditional ML models for COVID-19 fatality prediction and CNN architectures for nearest-neighbor search. The findings will provide insights into model selection and optimization, contributing to more effective applications in pandemic response and deep learning-based image processing.

PROBLEM STATEMENT

The increasing availability of COVID-19 data has led to the widespread application of machine learning techniques for predictive modeling. However, the performance of different machine learning models, such as K-Nearest Neighbours, Logistic Regression, and Random Forest Classification, varies significantly depending on dataset characteristics, feature selection, and preprocessing techniques. Understanding these variations is crucial for optimizing predictive accuracy and reliability in COVID-related studies. Additionally, deep learning models have gained prominence in image-based tasks, including nearest-neighbor search for medical image retrieval. Convolutional Neural Networks (CNNs) such as GoogLeNet, ZFNet, and ResNet-101 exhibit differing levels of efficiency and accuracy based on architectural depth and feature extraction capabilities. However, a comprehensive performance comparison of these models in the context of image-based nearest-neighbor search remains limited.

OBJECTIVES

Comparative Evaluation of Traditional ML Models for COVID-19 Prediction:

The objective here is to assess the performance of KNN, Random Forest, and Logistic Regression for predicting COVID-19 fatality outcomes based on various features in the dataset.

Evaluation Metrics: The models will be evaluated based on several performance metrics, including accuracy, precision, recall, F1 score, ROC-AUC, MAE, MSE, RMSE, and R2 score. These metrics will provide a comprehensive view of model performance, such as how well the model distinguishes between positive and negative cases (classification metrics), its error rates (regression metrics), and its overall prediction quality.
Outcome: The goal is to identify the most efficient and effective model in terms of both predictive accuracy and generalizability. By evaluating multiple models, we aim to understand their trade-offs and determine the best-suited algorithm for COVID-19 based fatality prediction in a real-world scenario.

Comparison of CNN Architectures for Nearest-Neighbor Search:

The objective is to evaluate the performance of deep learning architectures like GoogLeNet, ResNet, and ZFNet for nearest-neighbor search on a Kaggle image dataset with 10 classes.

Evaluation Metrics: The CNN architectures will be assessed based on training accuracy, training loss, and validation loss. These metrics will provide insight into how well each architecture is able to learn and generalize from the data, as well as how effectively they can minimize error during training and validation.
Outcome: The aim is to identify the most efficient CNN architecture for tasks like medical image retrieval, automated diagnosis, and content-based image search. By comparing these architectures, the study will provide a clearer understanding of the trade-offs between network depth, filter sizes, computational efficiency, and model performance.

Identification of Optimal Model for COVID-19 Prediction: The project aims to determine the most efficient ML model for predicting COVID-19 based fatality outcomes, balancing accuracy, computational cost, and generalizability.

Selection of Best CNN Architecture for Image Classification: By evaluating training accuracy, loss, and validation loss, the study will identify the CNN architecture most suitable for tasks like medical image retrieval and automated diagnosis. The findings will guide the selection of appropriate models for real-world applications, such as healthcare, where fast and accurate predictions are critical.

SCOPE AND LIMITATIONS

The models included for Covid detection are KNN, Random Forest, and Logistic Regression. The CNN architectures included for image classification are GoogLeNet, ResNet-101, and ZFNet. Our study will explore how these models and architectures perform across different evaluation criteria, ultimately aiming to identify the most effective approach for both COVID-19based fatality prediction and image classification.

Machine Learning part:
For our project, the dataset for COVID-19 contains entries of 1048575 patients with 21 features. This is a reasonably large dataset that helped us to provide ample information for training and evaluating the machine learning models. However the availability and quality of other COVID-19 dataset may pose challenges. The other datasets may have missing or incomplete records, which could impact model accuracy. Additionally, variations in dataset distribution (e.g., demographic imbalances) may affect generalizability.

There is a risk of overfitting in our case as the data size is huge. Balancing model complexity, regularization, and the size of the training data will be key challenges. We wanted the study's scope to include basic hyperparameter tuning, but unfortunately our laptop resources could not perform exhaustive grid search or cross-validation for such a large dataset, it would just require an unreasonable amount of time.

Deep Learning part:
Deep learning models such as GoogLeNet, ResNet-101, and ZFNet are computationally intensive and require significant processing power, especially for large datasets. Without GPU resources,training times can be unreasonably long and as a result we will try to implement this part of the project on a PC that has a powerful GPU configuration.

LITERATURE REVIEW/RELATED WORK

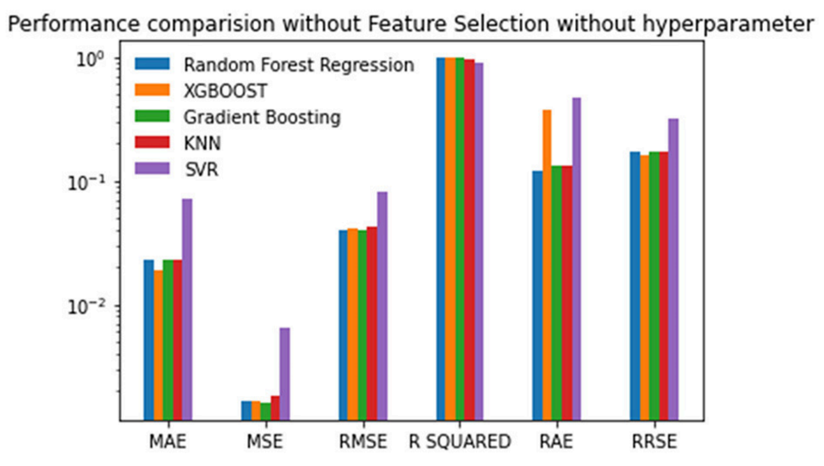
EXISTING STUDIES

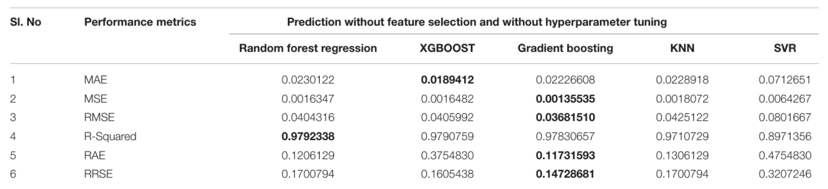
For the Machine Learning part, we have mainly gone through the following three articles.

(i) Performance Evaluation of Regression Models for the Prediction of the COVID-19 Reproduction Rate
Authors: Jayakumar Kaliappan1, Kathiravan Srinivasan1, Saeed Mian Qaisar2, Karpagam Sundararajan3, Chuan-Yu Chang4 and Suganthan C5*
Published: 14 September,2021

This study evaluated the performance of multiple nonlinear regression techniques—SVR, KNN, Random Forest Regressor, Gradient Boosting, and XGBOOST—for predicting the COVID-19 reproduction rate. Sixteen key features related to testing, deaths, positivity rate, active cases, stringency index, and population density were ranked using Random Forest, Gradient Boosting, and XGBOOST feature selection methods, from which seven were selected. Prediction performance was assessed using MAE, MSE, RMSE, R², RAE, and RRSE.

Since we will not be performing the hyperparameter tuning part for our project, we only mention about the following performance metrics found on this paper:





Sl. No	Performance metrics	Prediction without feature selection and without hyperparameter tuning				
		Random forest regression	XGBOOST	Gradient boosting	KNN	SVR
1	MAE	0.0230122	**0.0189412**	0.02226608	0.0228918	0.0712651
2	MSE	0.0016347	0.0016482	**0.00135535**	0.0018072	0.0064267
3	RMSE	0.0404316	0.0405992	**0.03681510**	0.0425122	0.0801667
4	R-Squared	**0.9792338**	0.9790759	0.97830657	0.9710729	0.8971356
5	RAE	0.1206129	0.3754830	**0.11731593**	0.1306129	0.4754830
6	RRSE	0.1700794	0.1605438	**0.14728681**	0.1700794	0.3207246

(ii) Prediction of COVID-19 Possibilities using KNN Classification Algorithm
Authors: Prasannavenkatesan Theerthagiri,I. Jeena Jacob,A.Usha Ruby,Vamsidhar Yendapalli,Published: September,2020
The COVID-19 dataset used for this study contained the patient's details with recovered and deceased status. The vital patient's information was used to diagnose and predict the COVID-19 disease among the infected population. The considered COVID-19 dataset contained 100284 records. The data preprocessing and cleaning process removed the missing and outliers data values from the dataset. The resulting dataset after preprocessing was reduced to 730 records with three required relevant features of patient details,with 99554 records missing required essential values.

The following accuracy scores were obtained for the different classifiers used:

S. No	Classifier	Accuracy
1.	Logistic Regression (LR)	78.5388
2.	K Neighbors Classifier (KNN)	80.3653
3.	Decision Tree (DT)	75.3425
4.	Support Vector Machines (SVM)	78.9954
5.	Multi-Layer Perceptron (MLP)	77.1689

The following error metrics of the classifiers were found as follows:

S. No	Classifier	MSE	RMSE	Kappa
1.	Logistic Regression (LR)	0.2146	0.4633	0.4109
2.	K Neighbors Classifier (KNN)	0.1963	0.4431	0.469
3.	Decision Tree (DT)	0.2466	0.4966	0.3043
4.	Support Vector Machines (SVM)	0.21	0.4583	0.4266
5.	Multi-Layer Perceptron (MLP)	0.2283	0.4778	0.3411

(iii) Covid-19 detection using Machine Learning
Authors: Bhuvaneswar, Harshitha K, Sahana- 191IT247

This project aimed at comparing different machine learning algorithms like K-nearest neighbors, Random forest and Naive Bayes with respect to their accuracies and then used the best one among them to develop a system which predicts whether a person has COVID or not using the data provided to the model. The data set used was from kaggle.com and it had 5434 × 21 rows of columns. This dataset contained 20 variables that could be determinants in the prediction of COVID-19. The following performance parameters were found:

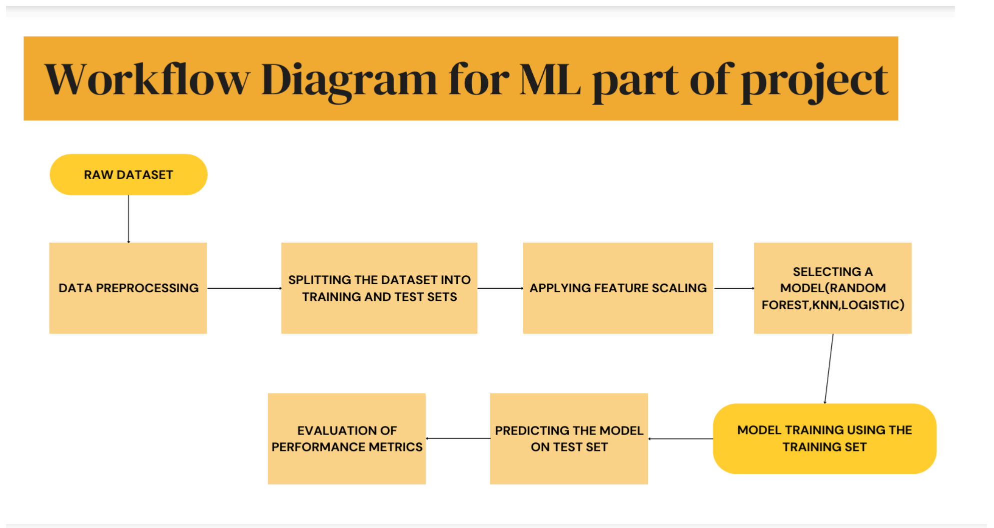
	Accuracy	MSE	R2 score	ROC score	Running time
KNN	98.37%	2.57	83.1	98.58	24.252
Logistic Regression	97.03%	3.036	80.086	93.23	0.038
Random Forest	98.39%	2.207	85.51	97.41	213.331

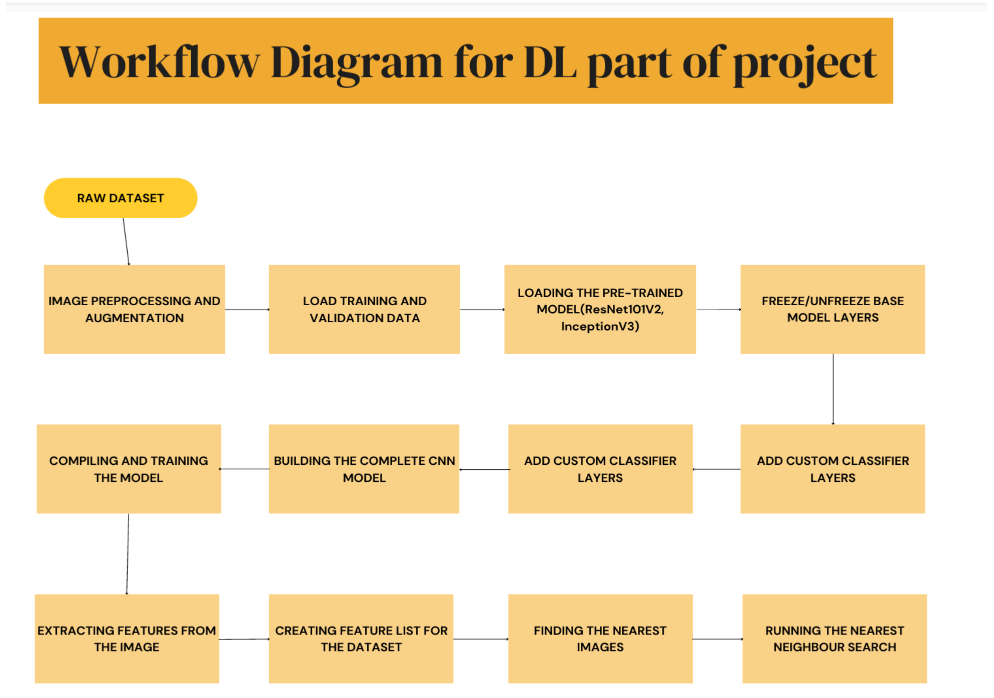
COMPARISON WITH EXISTING WORK

In our study for the ML part, a significantly larger dataset is utilized, which is expected to yield improved performance metrics compared to findings of the articles that have been shown. A larger dataset typically enhances model performance by providing more diverse and representative samples, reducing variance, and minimizing the risk of overfitting. With more data points available, machine learning models can better capture underlying patterns, leading to higher predictive accuracy and more reliable generalization. The key evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-Squared (R^2), Relative Absolute Error (RAE), and Root Relative Squared Error (RRSE) are anticipated to show noticeable improvements over previously reported results. Our study aims to leverage the advantages of this large dataset to achieve more precise and robust predictions of fatality based on COVID-19 cases.
For Deep Learning, we have gone through the following paper:
(iv) Comparative analysis of VGG, ResNet, and GoogLeNet architectures evaluating performance, computational efficiency, and convergence rates
DOI:10.54254/2755-2721/44/20230676

SYSTEM ARCHITECTURE / EXPERIMENTAL SETUP

SYSTEM DESIGN


Workflow Diagram for ML part of project
RAW DATASET
DATA PREPROCESSING
SPLITTING THE DATASET INTO TRAINING AND TEST SETS
APPLYING FEATURE SCALING
SELECTING A MODEL(RANDOM FOREST,KNN,LOGISTIC)
MODEL TRAINING USING THE TRAINING SET
PREDICTING THE MODEL ON TEST SET
EVALUATION OF PERFORMANCE METRICS


Workflow Diagram for DL part of project
RAW DATASET
IMAGE PREPROCESSING AND AUGMENTATION
LOAD TRAINING AND VALIDATION DATA
LOADING THE PRE-TRAINED MODEL(ResNet101V2, InceptionV3)
FREEZE/UNFREEZE BASE MODEL LAYERS
ADD CUSTOM CLASSIFIER LAYERS
ADD CUSTOM CLASSIFIER LAYERS
BUILDING THE COMPLETE CNN MODEL
COMPILING AND TRAINING THE MODEL
EXTRACTING FEATURES FROM THE IMAGE
CREATING FEATURE LIST FOR THE DATASET
FINDING THE NEAREST IMAGES
RUNNING THE NEAREST NEIGHBOUR SEARCH

SOFTWARE REQUIREMENTS

The project is implemented in VS Code with the Anaconda Python kernel, utilizing Python as the core programming language.
Libraries used for our project:

- ☐ Pandas
- ☐ Numpy
- ☐ Matplotlib and Seaborn
- ☐ Scikit-Learn
- ☐ TensorFlow
- ☐ Keras

DATA SOURCES AND PREPROCESSING

Data Preprocessing for ML part:
The dataset contains entries of 1048575 individuals(rows) and 21 features (these are the columns which include the target variables as well).
In our project, we have performed data preprocessing on the COVID-19 raw dataset from kaggle to present it in such a manner that anyone can understand what each column of this dataset is representing. This is done because the original dataset contains some values that are very ambiguous and can be difficult for people with no knowledge of the raw dataset to capture the essence of what the dataset is really trying to imply.
First, we transformed categorical values into more interpretable labels; for instance, we recoded the 'STATUS' column to indicate whether a patient "Survived" or "Died" and mapped 'SEX' to "male" and "female". Similarly, we modified 'PATIENT_TYPE' to distinguish between "Returned home" and "Hospitalization", while 'CLASIFFICATION_FINAL' was categorized into "Covid positive" and "Covid negative" based on its values. The 'USMER' column, representing healthcare levels, was converted into "Primary care," "Secondary care," and "Tertiary care", and the 'PREGNANT' column was adjusted to indicate "yes" or "no" based on the encoded values. To handle missing or ambiguous data, we replaced values 97 and 99 with NaN to better account for missing information.
Additionally, for multiple binary categorical columns—excluding key attributes like 'SEX', 'AGE', 'PATIENT_TYPE', 'STATUS', 'CLASIFFICATION_FINAL', 'USMER', and 'MEDICAL_UNIT'—we replaced 1 with "yes" and 2 with "no", making the dataset more interpretable. These preprocessing steps ensure that our dataset is clean, structured, and ready for further analysis and model training.

Data Preprocessing for DL part:
Dataset used: CALTECH - 101 from Kaggle; 3000 Images,10 Classes,Image Size: 224x224 pixel
In our project, we have implemented image preprocessing and data augmentation using Keras' ImageDataGenerator to enhance the performance of our deep learning model. We have applied rescaling, which normalizes pixel values to the range [0,1], along with shearing, zooming, and horizontal flipping to introduce variations in the training data and improve generalization. Additionally, we have split our dataset, reserving 20% for validation while using the remaining 80% for training. To efficiently load and process the images, we have used flow_from_directory, which retrieves images from the "dataset" directory, resizes them to 224x224 pixels, batches them in groups of 32, and organizes them for multi-class classification. This setup ensures that our model learns from diverse augmented images while also being evaluated on separate validation data. By incorporating these techniques, we aim to reduce overfitting, and optimize the performance of CNN architectures such as GoogLeNet, ResNet, and ZFNet, which we are analyzing in our project.

METHODOLOGY

THEORETICAL FOUNDATIONS

With the growth of computer technology, predictive modeling is changing. We are now able to make predictable modeling more efficient, and less expensive than before. In our project, we use various classification algorithms for prediction.

(i) Logistic Regression
Logistic regression is a data categorization technique that uses machine learning. This algorithm models the odds of the potential outcomes of a single experiment using a logistic function. The easiest way to understand the influence of numerous independent factors on a single outcome variable is to use logistic regression, which was designed for this purpose. In general, the algorithm calculates the probability of belonging to a particular class. We have two classes here, y=0,1.

(ii) KNN
The oldest supervised machine learning algorithm for classification is KNN, which classifies a given instance according to the majority of categories among its k-nearest neighbours in the dataset. The distance between the item to be categorized and every other item in the data set is calculated by the algorithm.

(iii) Random Forest Regression
This classifier is a meta-estimator that adapts to decision trees on the dataset's different sub-samples and utilizes the average to increase the model's predicted accuracy and control over-fitting. In most circumstances, this random forest classifier seems to be more accurate than decision trees, and it also minimizes overfitting. At the Random Forest level, average over all the trees is the final feature importance.
In our model evaluation, we have used several error metrics to determine which model performs the best:

Mean Absolute Error (MAE): It is the average of the absolute differences between predicted and actual values. It gives a simple measure of prediction error in the same units as the data.
Mean Squared Error (MSE): It is the average of the squared differences between predicted and actual values. It penalizes large errors more than small ones, making it sensitive to outliers.
Mean Squared Error (RMSE): It is the square root of MSE. It allows us to understand how large the errors are in the original scale of the data.
Cohen's Kappa (κ): This is a measure of agreement between two raters (or models), correcting for agreement occurring by chance. It ranges from -1 (complete disagreement) to 1 (complete agreement).
F1 Score: This is the harmonic mean of precision and recall, providing a balance between them. This is useful when class distribution is imbalanced and both false positives and false negatives are important.
R-Squared (R^2): It measures how well the model's predictions explain the variance of the true values. A higher value indicates a better model fit.
Precision: This is the proportion of correctly predicted positive instances out of all predicted positives. It shows how many predicted positives are actually true positives.
Recall: This is the proportion of correctly predicted positive instances out of all actual positives. It shows how well the model identifies all positive cases.
ROC-AUC Value: This is the area under the Receiver Operating Characteristic curve. It represents the ability of the model to distinguish between classes; a higher AUC means better performance.
ROC-AUC Curve: This provides a graphical representation showing the trade-off between true positive rate (recall) and false positive rate across different classification thresholds..

In the deep learning task, we have extracted image embeddings from the second-to-last layer of three pre-trained models (GoogLeNet, ResNet, and ZFNet) and then found the nearest 10 neighbors based on those embeddings.
Image Embedding Extraction:
In deep learning models, the second-to-last layer (often called the penultimate layer) represents high-level features of an image. The outputs from this layer are used as the image embeddings.
These embeddings are high-dimensional vectors that capture the essence of the image's content in a compressed form.
Mathematically, if an image I is passed through a pre-trained model $f$model, the embedding e can be represented as:

$$e=f\text{model}(I)$$

where $f$model represents the pre-trained network (GoogLeNet, ResNet, or ZFNet), and e is the resulting embedding vector.
Nearest Neighbor Search:
In order to measure how similar the embeddings of different images are, the two common methods used are:

(i)Cosine Similarity: Cosine similarity measures the similarity between two vectors (in this case, image embeddings) by calculating the cosine of the angle between them. The closer the cosine similarity is to 1, the more similar the two embeddings are, meaning the images are likely to be similar. A cosine similarity of 0 would indicate no similarity, and -1 indicates completely opposite directions.

(ii)Euclidean Distance: Euclidean distance is a metric that measures the straight-line distance between two points (in this case, the embeddings of two images) in the high-dimensional space. A smaller Euclidean distance means that the images are more similar, as the distance between their embeddings is smaller. Larger distances imply more dissimilarity.

The dimensionality of an embedding refers to the number of features or components in the vector representing an image. For example, an embedding may have hundreds or thousands of dimensions, each capturing different aspects of the image. The dimensionality affects how detailed or compressed the representation is.

Nearest neighbors are the images whose embeddings are most similar to a given image based on either cosine similarity or Euclidean distance. The algorithm identifies the top 10 images whose embeddings are closest to the target image, helping in finding visually similar images.

EXPERIMENTAL SETUP

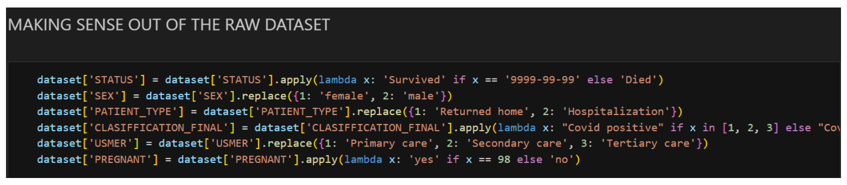
For ML part,the following steps were taken to perform the study:

Loading and Preprocessing the Dataset
The UPDATED_COVID.csv dataset was loaded with defined column names, skipping the first row. 'NaN', 'ICU', and 'INTUBED' were dropped, 'STATUS' moved to the last column, and missing values were handled. Categorical features were label-encoded, and 'STATUS' was binarized (1 = 'Died', 0 = 'Survived').
Splitting Data into Features and Target Variables
The feature set X (independent variables) and target variable y (dependent variable) were separated. The dataset was split into training and testing sets using train_test_split() with a test size of 20%.
Feature Scaling
The 'AGE' feature was scaled using StandardScaler. The scaling was performed separately for the training and testing datasets. Similarly, the 'MEDICAL_UNIT' column was also scaled.
Model Training and Evaluation
KNN,Logistic Regression and Random Forest Regression models were used.
Model Performance Evaluation
The confusion matrix was computed to evaluate the performance of the 3 models. Several performance metrics were computed for the KNN model, including: Mean Absolute Error (MAE),Mean Squared Error (MSE),Root Mean Squared Error (RMSE) ,R^2 Score,F1 Score, Precision,Recall,Cohen's Kappa, and ROC-AUC.

Visualization of Results

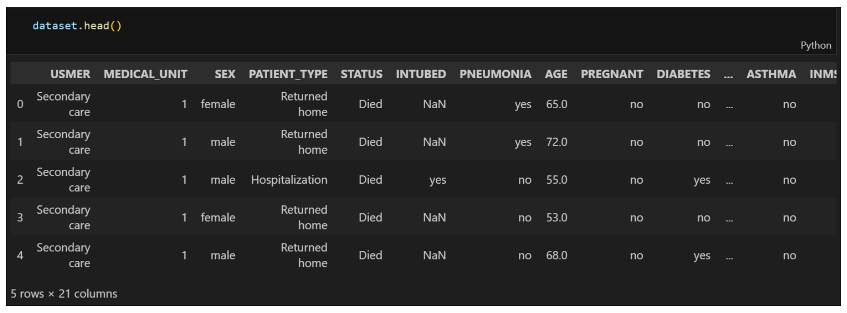
Confusion Matrix Visualization: The confusion matrix was plotted using ConfusionMatrixDisplay() to visually assess the classifier's performance.
Bar Chart Comparison: A bar chart comparing the error metrics (MSE, RMSE, Kappa) across Logistic Regression, Random Forest, and KNN models was generated.
ROC Curve Comparison: The ROC curve for KNN, Random Forest, and Logistic Regression models was plotted and compared, showing the AUC values for each model.
Bar Plot for Model Performance Metrics: A bar plot comparing Accuracy, Precision, Recall, and F1-Score across KNN, Random Forest, and Logistic Regression models was generated.

Below are some code snippets of how we have processed the raw dataset,splitted it into training and test sets and also performed feature scaling.


MAKING SENSE OUT OF THE RAW DATASET

```python
dataset['STATUS'] = dataset['STATUS'].apply(lambda x: 'Survived' if x == '9999-99-99' else 'Died')
dataset['SEX'] = dataset['SEX'].replace({1: 'female', 2: 'male'})
dataset['PATIENT_TYPE'] = dataset['PATIENT_TYPE'].replace({1: 'Returned home', 2: 'Hospitalization'})
dataset['CLASIFFICATION_FINAL'] = dataset['CLASIFFICATION_FINAL'].apply(lambda x: "Covid positive" if x in [1, 2, 3] else "Co
dataset['USMER'] = dataset['USMER'].replace({1: 'Primary care', 2: 'Secondary care', 3: 'Tertiary care'})
dataset['PREGNANT'] = dataset['PREGNANT'].apply(lambda x: 'yes' if x == 98 else 'no')
```

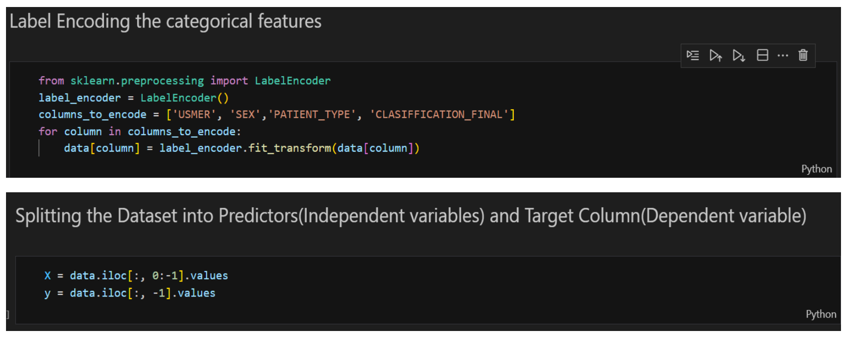
Fig: A code snippet of how the raw dataset is transformed to a meaningful representation. (We have in fact designed a separate .ipynb file just to make the raw dataset seem unambiguous and understandable)



```python
dataset.head()
```

```
     USMER    MEDICAL_UNIT     SEX  PATIENT_TYPE  STATUS  INTUBED  PNEUMONIA   AGE  PREGNANT  DIABETES  ...  ASTHMA  INMS
0  Secondary             1  female      Returned    Died      NaN        yes  65.0        no        no  ...      no
        care                                home
1  Secondary             1    male      Returned    Died      NaN        yes  72.0        no        no  ...      no
        care                                home
2  Secondary             1    male Hospitalization  Died      yes         no  55.0        no       yes  ...      no
        care
3  Secondary             1  female      Returned    Died      NaN         no  53.0        no        no  ...      no
        care                                home
4  Secondary             1    male      Returned    Died      NaN         no  68.0        no       yes  ...      no
        care                                home
5 rows × 21 columns
```

Fig: The dataset contains many columns which are categorical in type and needs to be converted to numerical representations.

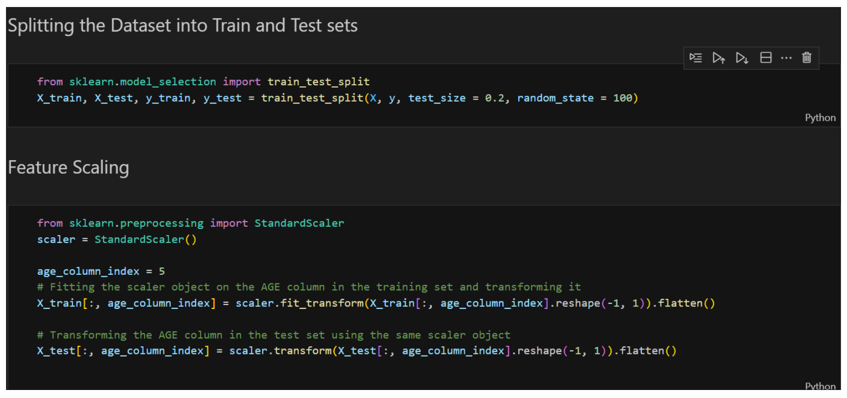

Label Encoding the categorical features

```python
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
columns_to_encode = ['USMER', 'SEX','PATIENT_TYPE', 'CLASIFFICATION_FINAL']
for column in columns_to_encode:
    data[column] = label_encoder.fit_transform(data[column])
```

Splitting the Dataset into Predictors(Independent variables) and Target Column(Dependent variable)

```python
X = data.iloc[:, 0:-1].values
y = data.iloc[:, -1].values
```

Fig: Label Encoding categorical features and splitting the dataset into independent variables and dependent variable


Splitting the Dataset into Train and Test sets

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)
```

Feature Scaling

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

age_column_index = 5
# Fitting the scaler object on the AGE column in the training set and transforming it
X_train[:, age_column_index] = scaler.fit_transform(X_train[:, age_column_index].reshape(-1, 1)).flatten()

# Transforming the AGE column in the test set using the same scaler object
X_test[:, age_column_index] = scaler.transform(X_test[:, age_column_index].reshape(-1, 1)).flatten()
```

Fig: Splitting the dataset using help from a module of the sklearn library and also performing feature scaling

For DL part,the following steps were taken to perform the study:

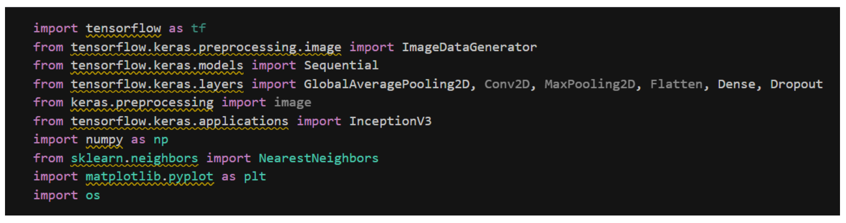
Data Preprocessing:
Image Resizing: The image data was scaled using ImageDataGenerator to normalize the pixel values by dividing by 255, ensuring that all images are in the range of 0 to 1.
Augmentation: Data augmentation was applied (shear, zoom, and horizontal flip) to introduce variability in the training data, which helps improve the generalization of the model. A validation split of 20% was also specified.
Image Size: Images were resized to (299, 299) pixels to match the input size required by the pretrained models.
Loading the Dataset:
The dataset was loaded from a directory ('dataset') using flow_from_directory. This method automatically labels images based on their directory structure and applies data augmentation for training and validation sets.
The images were loaded in batches of 32, and the labels were one-hot encoded using class_mode='categorical'.
Loading Pretrained Model:
A base model like (InceptionV3) was loaded with pretrained weights from ImageNet (weights='imagenet') without the top layers (include_top=False). This allows for feature extraction without interference from the model's original classification layers. The model's layers were frozen until layer 40 to prevent overfitting and speed up training. Layers from layer 40 onward were left trainable, enabling fine-tuning.
Creating a New Classifier:
A new classifier was added on top of the base model:
GlobalAveragePooling2D: It reduces the spatial dimensions of the feature map.
Dense Layer (1024 units): This is a fully connected layer with ReLU activation function.
Dropout: Regularization technique (50%) was used to prevent overfitting.
Output Layer (Dense with 10 units): Output layer with softmax activation was used for multi-class classification.
Model Compilation:
The model was compiled using the Adam optimizer and categorical cross-entropy loss for multi-class classification, with accuracy as the evaluation metric.
Model Training:
The model is trained for 10 epochs using the fit method on the training data (training_set) and validated using the validation data (validation_set).
Performance Metrics: Training accuracy, training loss, validation accuracy, and validation loss were evaluated during each epoch.
Feature Extraction:
After training, the model was used to extract features from images.
extract_features_fixed: This function loads an image, resizes it to the required input size (299x299), normalizes it, and extracts the features using the base model (InceptionV3).These features were then stored in a list for all images in the dataset.
Nearest Neighbor Search:
Using the NearestNeighbors algorithm (with cosine distance), the features of a query image were compared with the stored features of all images in the dataset. For each query image, the nearest n_neighbors (10 in our case) were identified, and the images were displayed along with their distance values. This was done for multiple query images (e.g., one from the 'Motorbikes' class and one from a different class, 'plane').

Comparison Between Models:
GoogLeNet, ResNet, and ZFNet all were used in our study for finding nearest neighbors, following similar steps as with InceptionV3.
For each model, the following metrics were recorded: Training Accuracy, Training Loss, Validation Accuracy, and Validation Loss.
Results Analysis and Comparison:
The performance of each model (InceptionV3, GoogLeNet, ResNet, ZFNet) was compared using the following:
Training Accuracy and Loss: To assess how well the model fits the training data.
Validation Accuracy and Loss: To evaluate the model's generalization capabilities.
Nearest Neighbor Results: To compare the ability of each model to find similar images in terms of feature space.

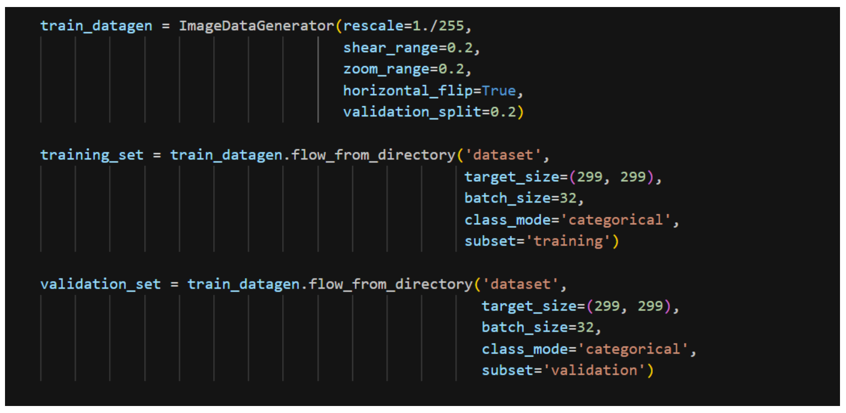
Architecture for ZFNet:
For AlexNet and InceptionV3, we utilized pre-trained weights up to specific layers, leveraging their learned feature representations while fine-tuning the later layers for our specific task. However, for ZFNet, we designed the entire architecture from scratch, defining all convolutional, pooling, and fully connected layers without using any pre-trained weights.The ZFNet architecture in our implementation consists of multiple convolutional, pooling, and fully connected layers designed from scratch. It begins with a Conv2D layer with 96 filters of size (7,7) and a stride of (2,2), followed by a MaxPooling2D layer with a pool size of (3,3) and a stride of (2,2) to reduce spatial dimensions. The second convolutional layer has 256 filters of size (5,5) with a ReLU activation, followed by another max pooling layer. Then, two consecutive Conv2D layers with 384 filters of size (3,3) extract deeper features. Another convolutional layer with 256 filters is added before the final max pooling layer. The extracted features are flattened and passed through two Dense layers of 4096 neurons each, with ReLU activation and Dropout (0.5) to prevent overfitting. Finally, a Dense layer with 10 neurons and a softmax activation function is used for classification.

Visualization:
For each query image, the nearest images were displayed alongside their distances. In addition plots of performance metrics (accuracy, loss, etc.) for each model were presented for visual comparison.

Below are some code snippets of the libraries that have been used and the process of data preprocessing.



```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.preprocessing import image
from tensorflow.keras.applications import InceptionV3
import numpy as np
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import os
```



```python
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   validation_split=0.2)

training_set = train_datagen.flow_from_directory('dataset',
                                                 target_size=(299, 299),
                                                 batch_size=32,
                                                 class_mode='categorical',
                                                 subset='training')

validation_set = train_datagen.flow_from_directory('dataset',
                                                   target_size=(299, 299),
                                                   batch_size=32,
                                                   class_mode='categorical',
                                                   subset='validation')
```

ASSUMPTIONS AND CONSTRAINTS

Assumptions for the ML study:
It was assumed that the dataset provided was clean and representative of the problem being studied, except for missing values, which were handled through imputation or removal.It was assumed that imputing missing values (e.g., replacing missing 'AGE' values with the mean) was an appropriate approach, assuming the missing data did not introduce bias. The dataset was assumed to be sufficiently large for training and validation of machine learning models, ensuring reliable results.

Limitations in the methodology for ML:
Columns like 'ICU' and 'INTUBED' were dropped. The decision to exclude these features could lead to loss of important information that might have improved the model's predictive power.There is a lack of cross validation since the study used a single train-test split (80% train, 20% test). While this is common, it may lead to variability in the results depending on how the data is split. The presence of correlated features could lead to redundancy in the models, potentially reducing their performance. Since models like Random Forest and KNN were used without hyperparameter tuning, there remains a risk of overfitting.

Assumptions for the DL study:
It is assumed that the images in the dataset are of good quality and that preprocessing steps (rescaling, augmentation, etc.) will improve the model's ability to generalize without introducing significant noise. The study assumes that pretrained models like InceptionV3, GoogLeNet, ResNet, and ZFNet can effectively extract relevant features from the images and that these features can be used for nearest neighbor search, irrespective of the image class or dataset. It is assumed that cosine distance is an appropriate metric for finding the nearest neighbors in the feature space. The study assumes that fine-tuning the last layers of the pretrained models will allow for better performance on the new dataset while avoiding overfitting. Only the latter layers are made trainable to prevent overfitting on small datasets.

Limitations in the methodology for DL:
Fine-tuning the models could potentially lead to overfitting on the training data, especially if the dataset is small. If the model is too complex for the available data, it may memorize the data instead of learning generalized patterns. The study only compares the performance of InceptionV3, GoogLeNet, ResNet, and ZFNet. There are many other models that could potentially outperform these, such as EfficientNet, DenseNet, or other newer architectures. Deep learning models like InceptionV3, GoogLeNet, ResNet, and ZFNet are computationally expensive. Running these models on a CPU may lead to longer training times and even inability to train the models effectively.

RESULTS AND ANALYSIS

The following performance metrics were obtained for KNN,Logistic Regression and Random Forest Classification:

	Accuracy	Precision	Recall	F1	ROC-AUC	Kappa
KNN	0.9497	0.5167	0.4046	0.4539	0.8930	0.4279
Logistic Regression	0.9099	0.3564	0.9263	0.5147	0.9626	0.4756
Random Forest	0.9314	0.3995	0.6563	0.4967	0.9300	0.4622

	MAE	MSE	RMSE	R^2
KNN	0.0503	0.0503	0.2242	-0.0268
Logistic Regression	0.0901	0.0901	0.3002	-0.8416
Random Forest	0.0686	0.0686	0.2620	-0.4025

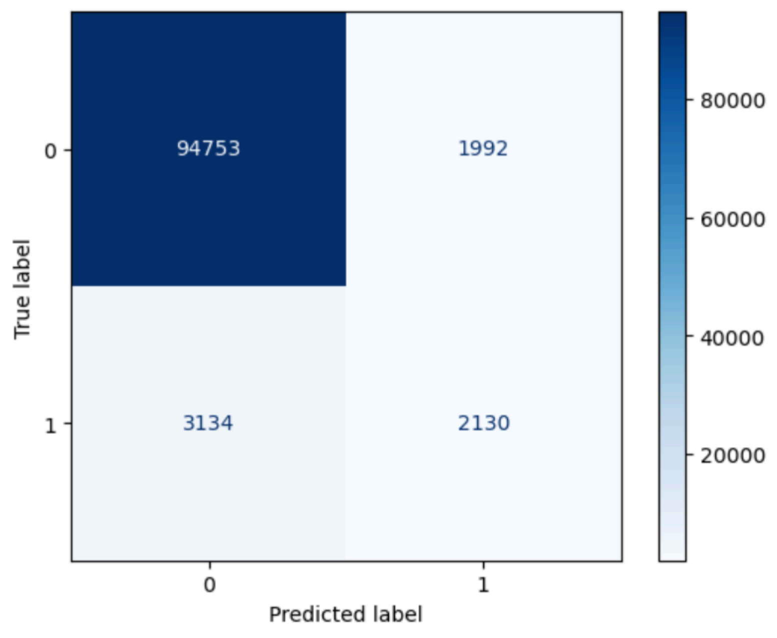


Fig. Confusion Matrix for KNN

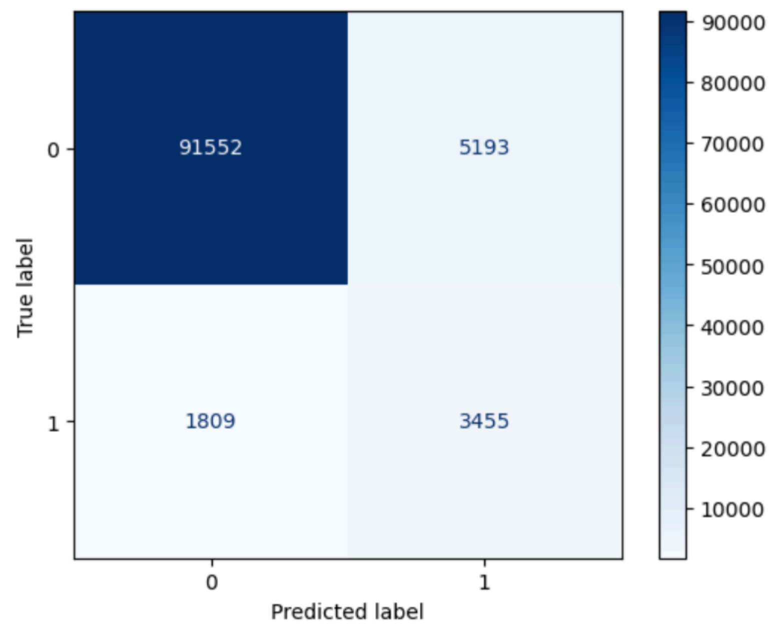


Fig. Confusion Matrix for Random Forest

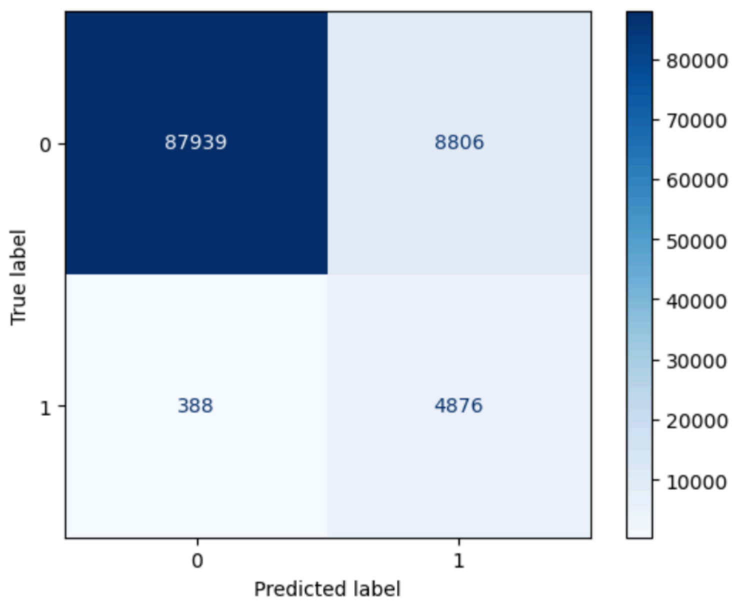
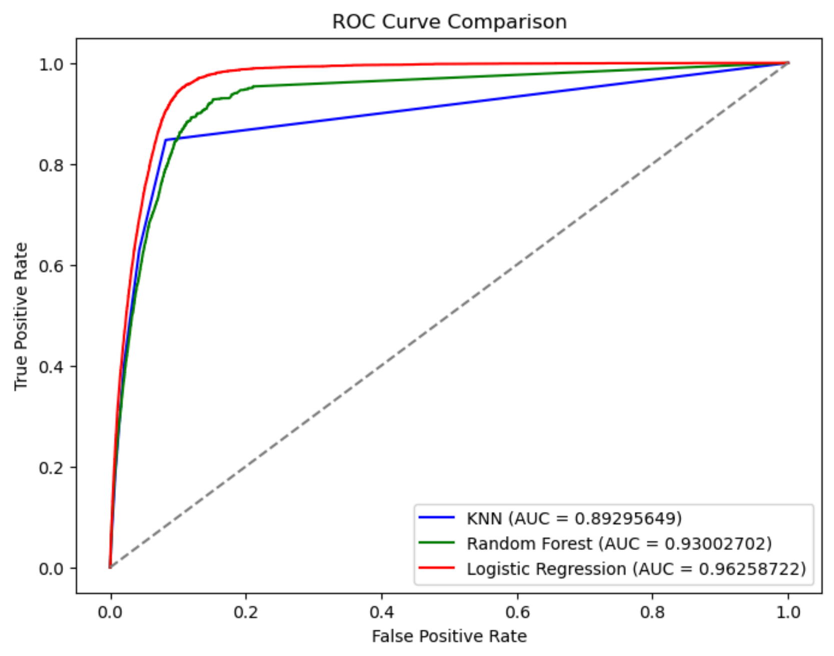
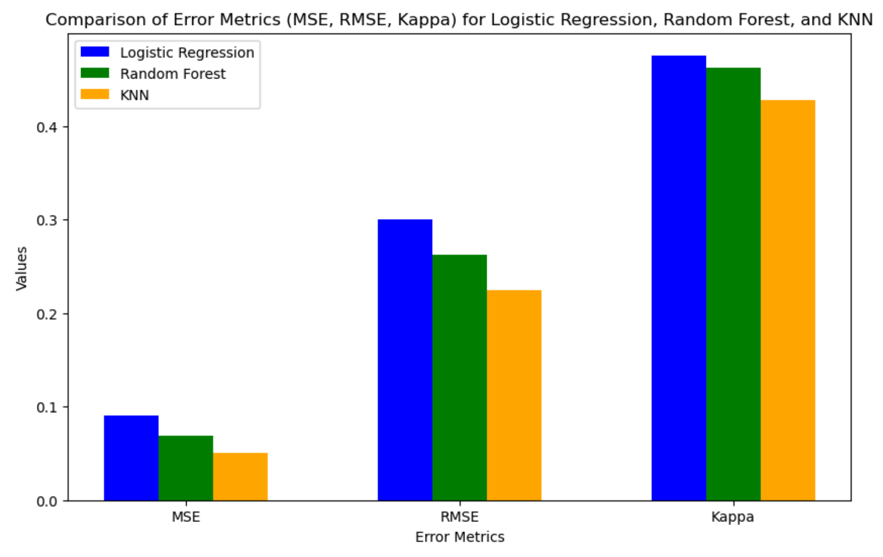
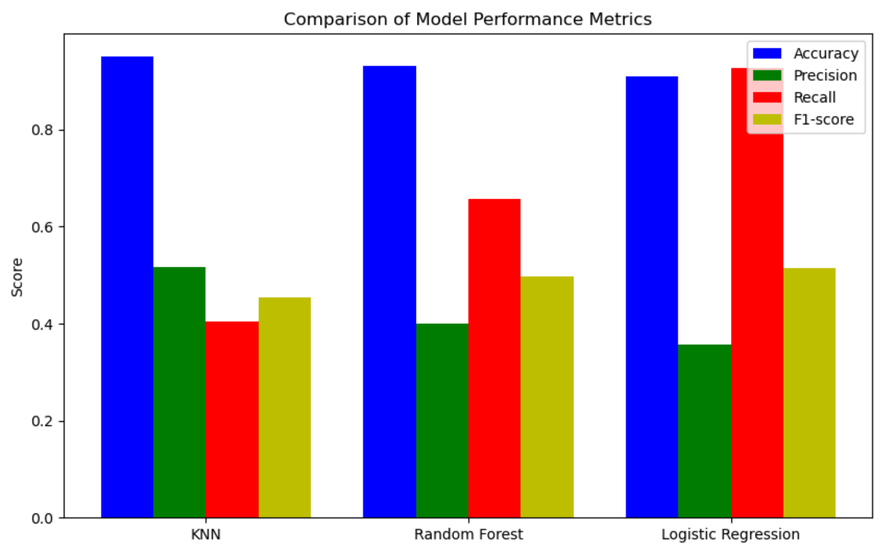


Fig. Confusion Matrix for Logistic Regression





The classification results indicate that KNN achieved the highest accuracy (0.9497), followed by Random Forest (0.9314) and Logistic Regression (0.9099), though accuracy alone may not be the best performance measure. Logistic Regression excelled in recall (0.9263), F1-score (0.5147), ROC-AUC (0.9626), and Kappa (0.4756), making it the best model for detecting positive cases. KNN had the highest precision (0.5167) but struggled with recall (0.4046), meaning it missed many true positives. Random Forest showed a balanced performance between these metrics. In regression-based evaluations, KNN had the lowest errors (MAE: 0.0503, MSE: 0.0503, RMSE: 0.2242), indicating better prediction accuracy, whereas Logistic Regression had the highest errors and the worst R² (-0.8416), meaning it performed poorly compared to a simple mean-based predictor. Notably, all models had negative R² values, suggesting they failed to capture variance effectively. Overall, KNN is preferable for accuracy-driven tasks, Logistic Regression for high recall, and Random Forest for a balanced trade-off.



The following performance metrics were obtained for KNN,Logistic Regression and Random Forest Classification after balancing class distribution:

	Accuracy	Balanced Accuracy	Precision	Recall	F1	ROC-AUC	Kappa
KNN	0.9146	0.9145	0.9093	0.9232	0.9162	0.9483	0.8291
Logistic Regression	0.9199	0.9199	0.9186	0.9236	0.9211	0.9660	0.8398
Random Forest	0.9497	0.9435	0.8626	0.9320	0.8960	0.9830	0.8629

	MAE	MSE	RMSE	R^2
KNN	0.0854	0.0854	0.2923	0.66
Logistic Regression	0.0801	0.0801	0.2830	0.68
Random Forest	0.0503	0.2243	0.2620	0.72

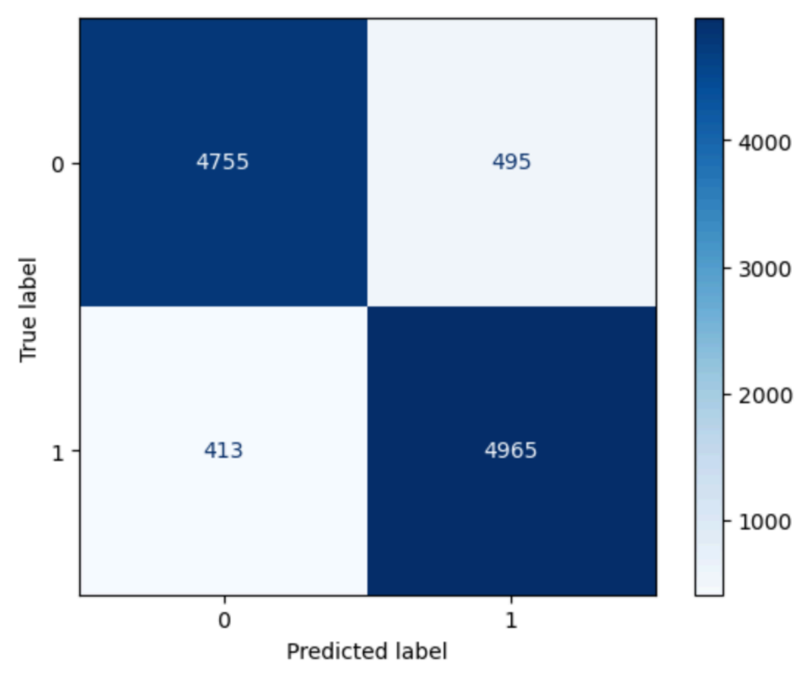
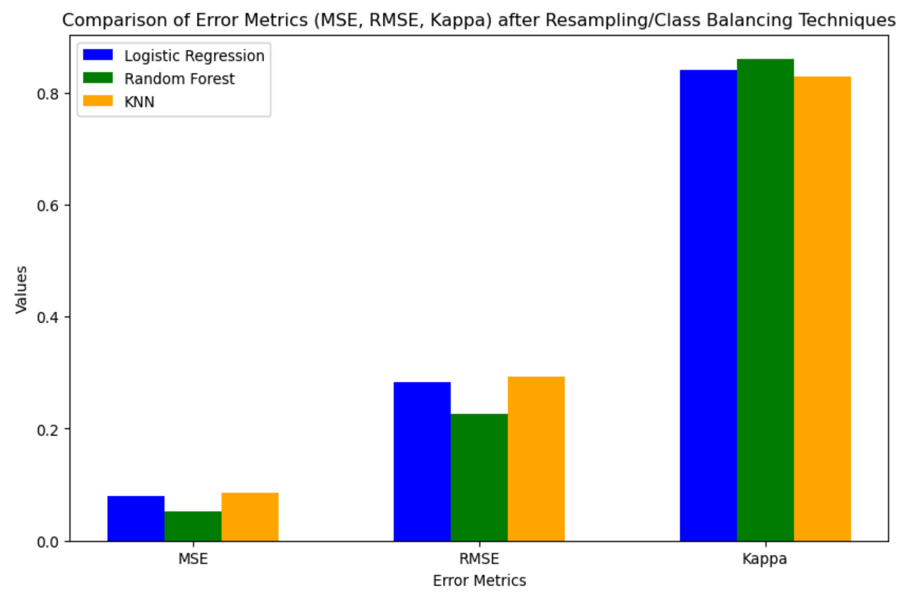


Fig. Confusion Matrix for KNN



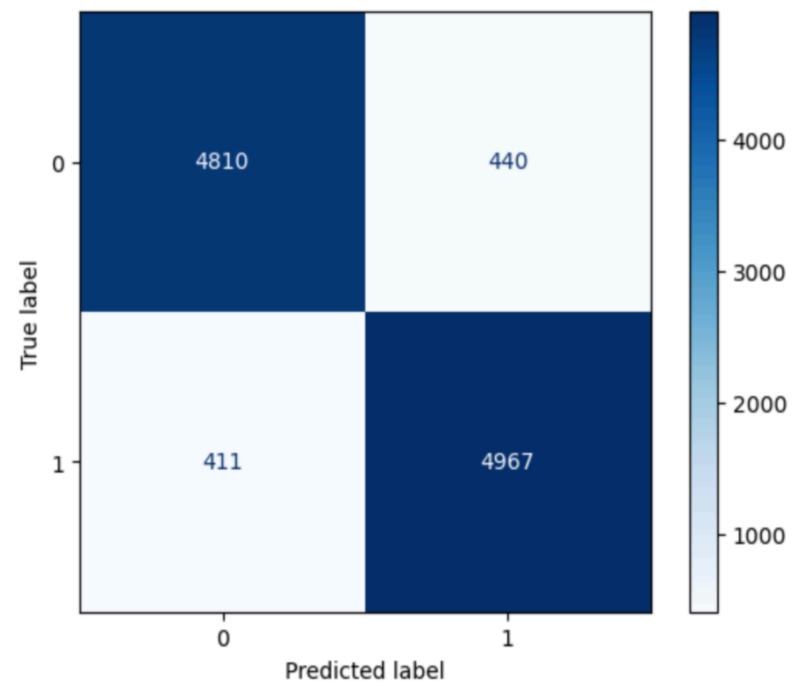
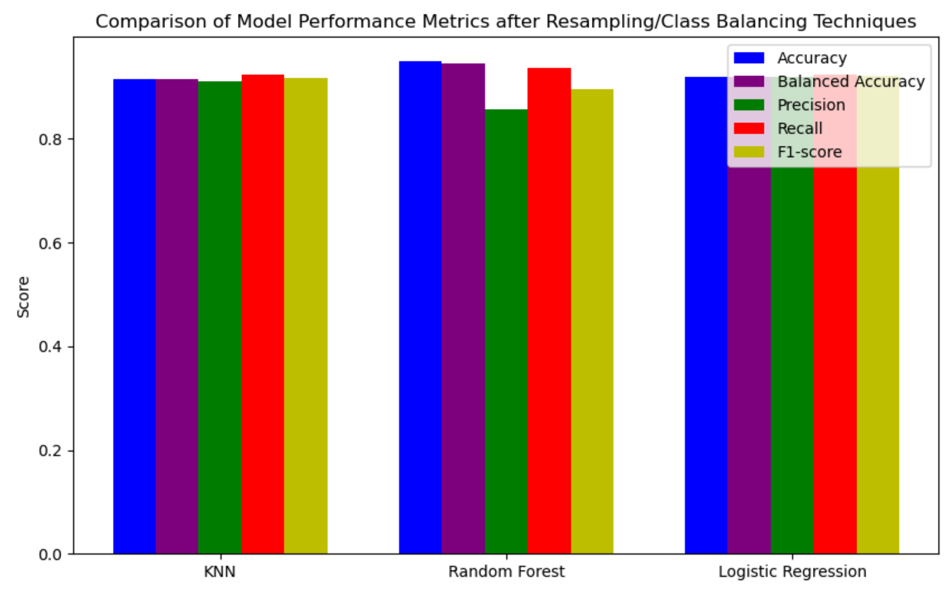


Fig. Confusion Matrix for Logistic Regression



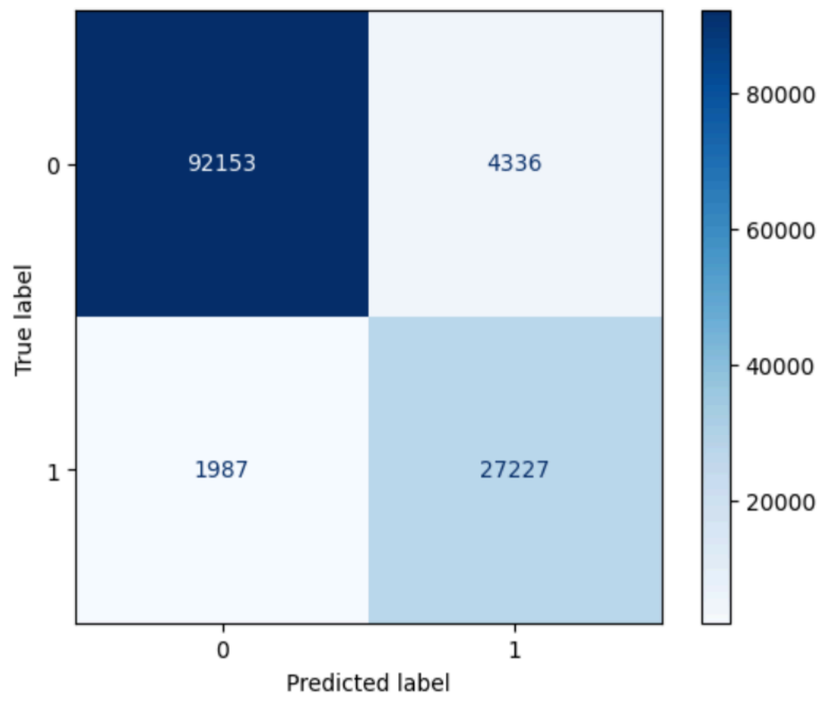
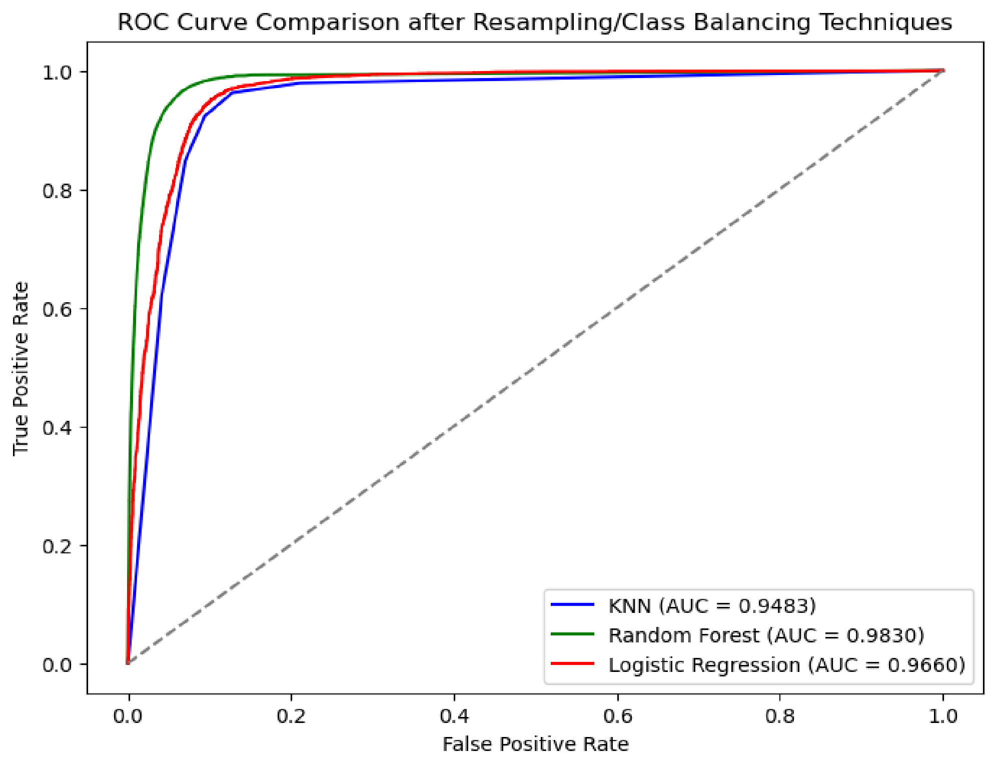


Fig. Confusion Matrix for Random Forest



After balancing the class distribution, Random Forest achieved the highest accuracy (0.9497), ROC-AUC (0.9830), and Kappa (0.8629), demonstrating its robustness even after balancing. Logistic Regression and KNN performed closely, with Logistic Regression excelling in precision (0.9186) and recall (0.9236), resulting in the highest F1-score (0.9211), making it ideal for recall-sensitive tasks. KNN maintained competitive recall (0.9232) and precision (0.9093) but had slightly lower accuracy (0.9146). Compared to the previous results before balancing, Random Forest's accuracy improved (from 0.9314 to 0.9497), and its F1-score slightly dropped (from 0.4967 to 0.8960), indicating better overall classification performance across classes. Logistic Regression also showed improved balance, with an F1-score rising from 0.5147 to 0.9211. Regression-based metrics improved significantly, with Random Forest now showing the lowest error (MAE: 0.0503, RMSE: 0.2620) and highest R^2 (0.72), indicating a stronger fit. KNN and Logistic Regression also improved their R^2 values (from negative values previously to 0.66 and 0.68, respectively), demonstrating enhanced predictive power. However, balancing the class distribution may have affected Random Forest's diversity, potentially reducing its generalization ability. Nonetheless, the results indicate that balancing led to overall better classification and regression performance, making Random Forest the best-performing model post-balancing.

CONCLUSION FOR ML PORTION:

Balancing the class distribution significantly improved the overall classification and regression performance of all models, with Random Forest emerging as the best performer in terms of accuracy (0.9497), ROC-AUC (0.9830), and regression metrics (lowest error and highest R^2 of 0.72). Logistic Regression achieved the highest F1-score (0.9211) , making it ideal for recall-focused tasks, while KNN maintained competitive performance. Compared to the initial results, balancing enhanced model stability and predictive power, particularly for Random Forest and Logistic Regression. However, undersampling may have reduced data diversity, potentially affecting Random Forest's ability to generalize. Overall, Random Forest remains the strongest choice, especially for balanced classification and regression, while Logistic Regression is preferable for high recall applications.

FUTURE WORK FOR ML PORTION:

Hyperparameter tuning was not performed due to computational limitations, but Grid Search, Random Search, or Bayesian Optimization could be explored in future work for better model optimization. Additionally, Ensemble Learning, Deep Learning models, and Feature Selection techniques like PCA or RFE could further enhance performance but were not feasible due to resource constraints.

RESULT AND ANALYSIS FOR DL PART:

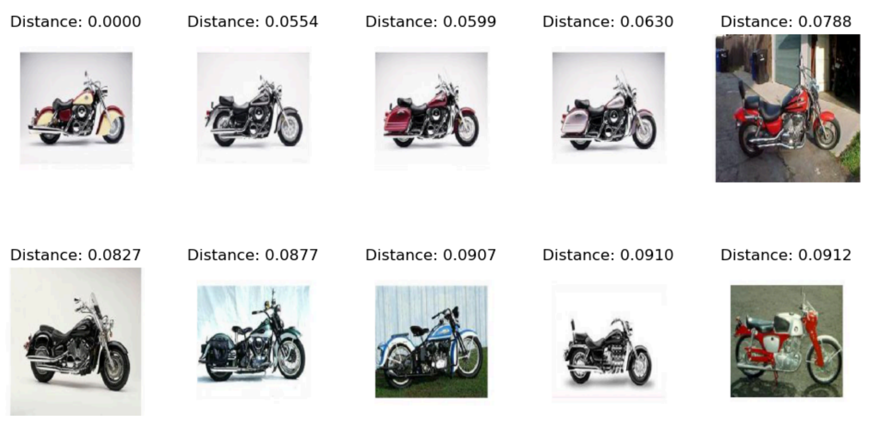


Fig: Nearest Neighbours(Images) to a sample photo using GoogleNet

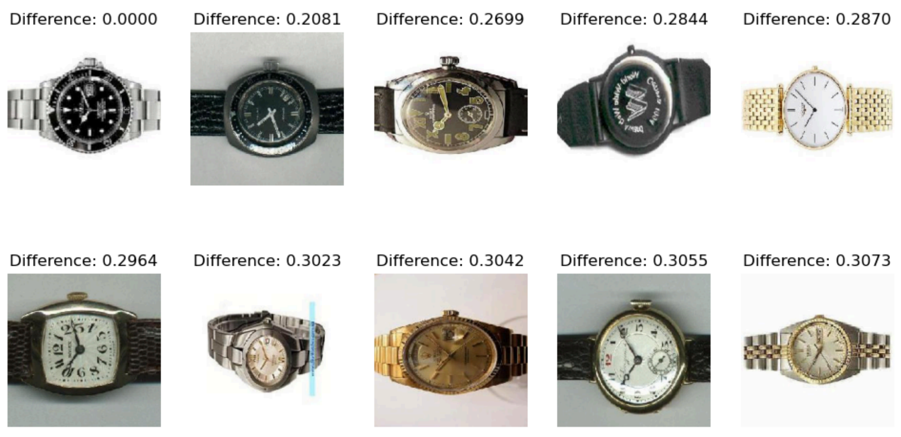


Fig: Nearest Neighbours(Images) to a sample photo using ResNet

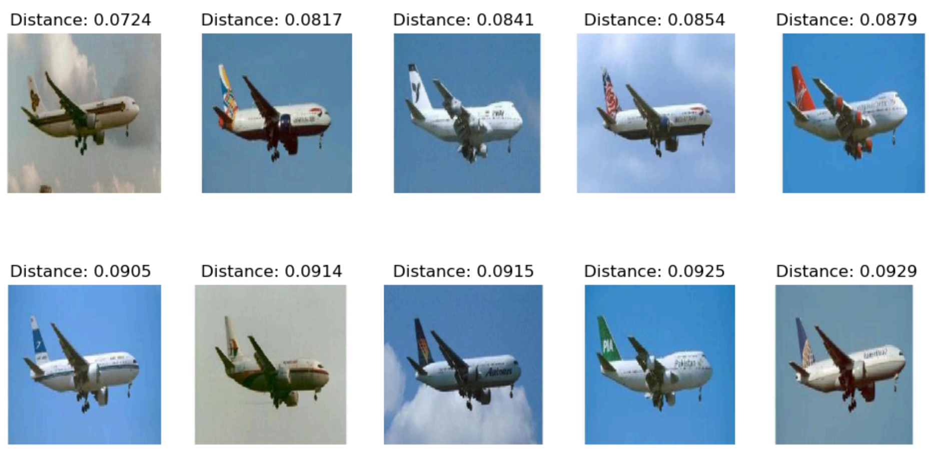
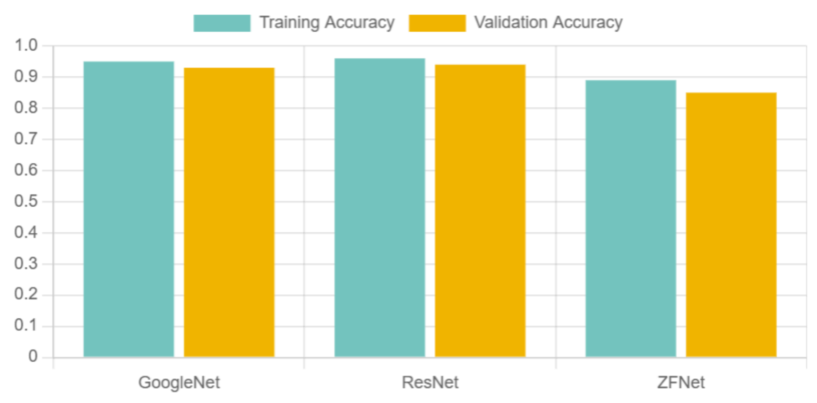


Fig: Nearest Neighbours(Images) to a sample photo using ResNet



The bar chart compares the Training Accuracy (cyan) and Validation Accuracy (yellow) of GoogLeNet, ResNet, and ZFNet. Both GoogLeNet and ResNet achieve high training and validation accuracy, indicating strong generalization with minimal overfitting. ZFNet, however, shows slightly lower accuracy, suggesting it may not capture features as effectively as the others or might require further tuning. The small gap between training and validation accuracy across all models suggests a well-regularized training process. Overall, ResNet performs best, while ZFNet lags slightly, indicating potential for further optimization.

FUTURE WORK FOR DL PORTION:

Future work for this project could focus on optimizing ZFNet through hyperparameter tuning, advanced regularization techniques, and exploring architectural improvements. Additionally, enhancing model generalization via data augmentation and cross-validation could be beneficial. Given that ResNet performed the best, exploring advanced versions like ResNet-152 or using ensemble learning could further boost performance. Incorporating additional performance metrics, conducting error analysis, and experimenting with other architectures like VGGNet or InceptionV3 could also provide deeper insights. Finally, deploying the best models in real-world applications and evaluating their real-time performance would help assess their practical utility.

CONTRIBUTIONS:

200021131:
(i)Covid fatality Dataset cleaning(making sense out of the raw dataset).
(ii)Data Augmentation and Implementation of Transfer Learning to evaluate performance metrics of three different CNN Architectures.
(iii)Documentation for the Deep Learning portion including methodology,workflow,theoretical background,limitations,result & analysis and future work.

200021141:
(i)Identifying the imbalance ratio in the target class.
(ii)Mitigating imbalance ratio by undersampling majority class for KNN and Logistic Regression and implementation of SMOTE for generating synthetic samples for minority class (for Random Forest).
(iii)Evaluating performance metrics after adjusting imbalance ratio and documentation for the corresponding segments.

200021150:
(i)Performing EDA on the dataset and preparing the data set for training and testing.
(ii)Comparing and illustrating performance parameters of different ML algorithms after feature selection and before adjusting class balance.
(iii)Documentation for Machine Learning portion including methodology,findings of past papers,workflow for the algorithm.

ETHICAL CONSIDERATIONS AND SUSTAINABILITY

ETHICAL ISSUES

Both of our studies handle sensitive data. The COVID-19 study must ensure health data is anonymized and secure. In image classification, the use of personal images could raise privacy concerns. The compliance with data protection regulations like GDPR is essential. Biases in data can lead to unfair predictions or classifications. In the COVID-19 project, non-representative data could affect healthcare decisions. In image classification, biased datasets may result in poor performance for certain groups. There could be ethical concerns over using models for decision-making in healthcare, where human oversight may be necessary. Similarly, image classification could raise issues around surveillance and privacy if misused.

SUSTAINABILITY

The COVID-19 model could improve healthcare efficiency and reduce costs. Image classification models could automate processes and reduce operational costs, but could also lead to job displacement in certain sectors. Both projects could benefit society by improving healthcare and efficiency. However, there are concerns about AI-driven surveillance and job displacement. Sustainable and fair AI deployment is essential for positive societal impact.

FINAL VERDICT:

Machine Learning part

From the performance metrics,Random Forest Classification emerged as the top-performing model, achieving the highest accuracy (0.9457) and balanced accuracy (0.9458). This indicates that Random Forest is highly effective in correctly classifying instances across both classes. Additionally, it attained the highest ROC-AUC score (0.9830), suggesting excellent performance in distinguishing between the classes. The Kappa score of 0.8829 further underscores its reliability and consistency in predictions. However, it is worth noting that Random Forest had slightly lower precision (0.8636) and recall (0.8320) compared to the other models, which may indicate a trade-off between overall accuracy and the ability to correctly identify positive cases.Logistic Regression also performed well, with accuracy and balanced accuracy scores of 0.9199. It achieved the highest precision (0.9186) and recall (0.9236) among the three models, indicating its strength in correctly identifying positive cases and minimizing false positives. The F1 score of 0.9211 and ROC-AUC score of 0.9660 further confirm its robustness in classification tasks. The Kappa score of 0.8398 reflects a strong agreement between predicted and actual classifications. KNN, while slightly less accurate than the other two models, still demonstrated solid performance with an accuracy of 0.9146 and balanced accuracy of 0.9145. It achieved a high recall (0.9232), indicating its effectiveness in identifying positive cases, though its precision (0.9033) was slightly lower. The F1 score of 0.9162 and ROC-AUC score of 0.9448 suggest that KNN is a reliable model for classification tasks, particularly when recall is a priority. In terms of error metrics, Random Forest had the lowest Mean Absolute Error (MAE) of 0.0503 and the highest R^2 score (0.72), indicating the best fit to the data. Logistic Regression followed closely with an MAE of 0.0801 and an R^2 score of 0.68, while KNN had an MAE of 0.0854 and an R^2 score of 0.66. Overall, the findings underscore the importance of model selection based on specific performance requirements. Random Forest is the best choice for maximizing overall accuracy and ROC-AUC, while Logistic Regression excels in precision and recall. KNN remains a viable option, particularly when recall is a critical metric. These insights are crucial for guiding model selection in real-world classification tasks, ensuring optimal performance based on the specific needs of the application.

Deep Learning part

GoogleNet, ResNet, and ZFNet are well-known convolutional neural networks (CNNs) that have been widely used in various computer vision tasks. The training accuracy values reflect their learning capabilities, with higher values indicating better performance in fitting the training data. However, it is important to consider that high training accuracy does not always translate to high validation or test accuracy, as overfitting can occur. The significance of this work lies in its potential to guide the selection of appropriate neural network architectures for specific tasks. By comparing the training accuracies of these models, practitioners can make informed decisions about which architecture might be best suited for their particular application. Additionally, this analysis can serve as a foundation for further research, such as exploring techniques to improve generalization, reduce overfitting, or enhance model efficiency. In conclusion, the work underscores the importance of evaluating and comparing different neural network architectures to achieve optimal performance in machine learning tasks. The insights gained from this analysis can contribute to the development of more accurate and robust models, ultimately advancing the field of computer vision and deep learning.

REFERENCES

➢ *Performance Evaluation of Regression Models for the Prediction of the COVID-19 Reproduction Rate*
Authors: Jayakumar Kaliappan1, Kathiravan Srinivasan1, Saeed Mian Qaisar2, Karpagam Sundararajan3, Chuan-Yu Chang4 and Suganthan C5*

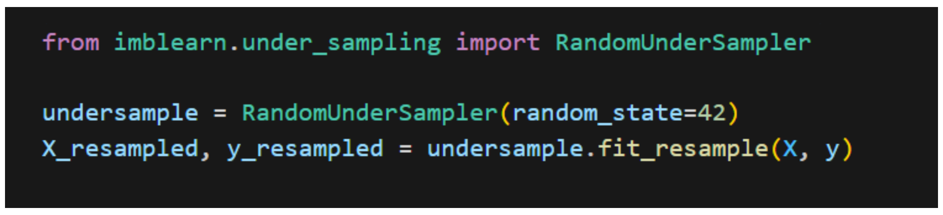
➢ *Prediction of COVID-19 Possibilities using KNN Classification Algorithm*
Authors: Prasannavenkatesan Theerthagiri,I. Jeena Jacob,A.Usha Ruby,Vamsidhar Yendapalli

➢ *Covid-19 detection using Machine Learning*
Authors: Bhuvaneswar, Harshitha K, Sahana- 191IT247

➢ *Comparative analysis of VGG, ResNet, and GoogLeNet architectures evaluating performance, computational efficiency, and convergence rates*
DOI:10.54254/2755-2721/44/20230676

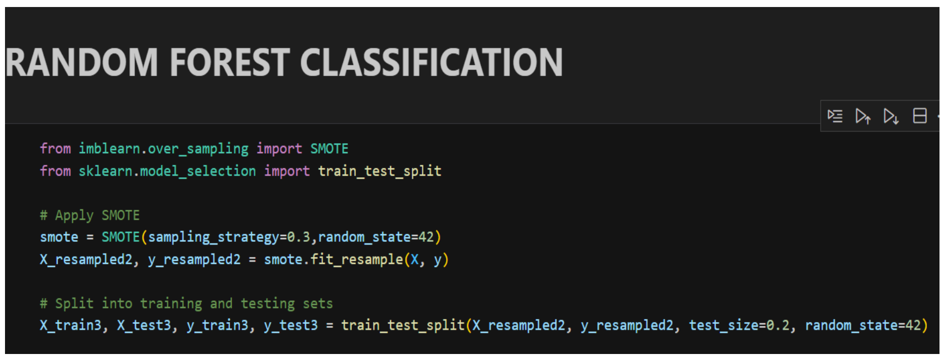
APPENDIX

A.Python Code for Adjusting the imbalance ratio



```python
from imblearn.under_sampling import RandomUnderSampler

undersample = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = undersample.fit_resample(X, y)
```

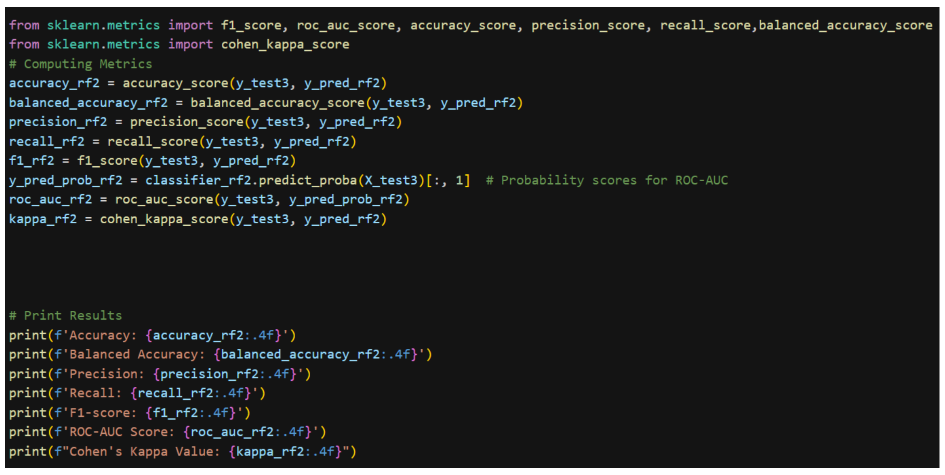
B.Python Code for implementing SMOTE for Random Forest



RANDOM FOREST CLASSIFICATION

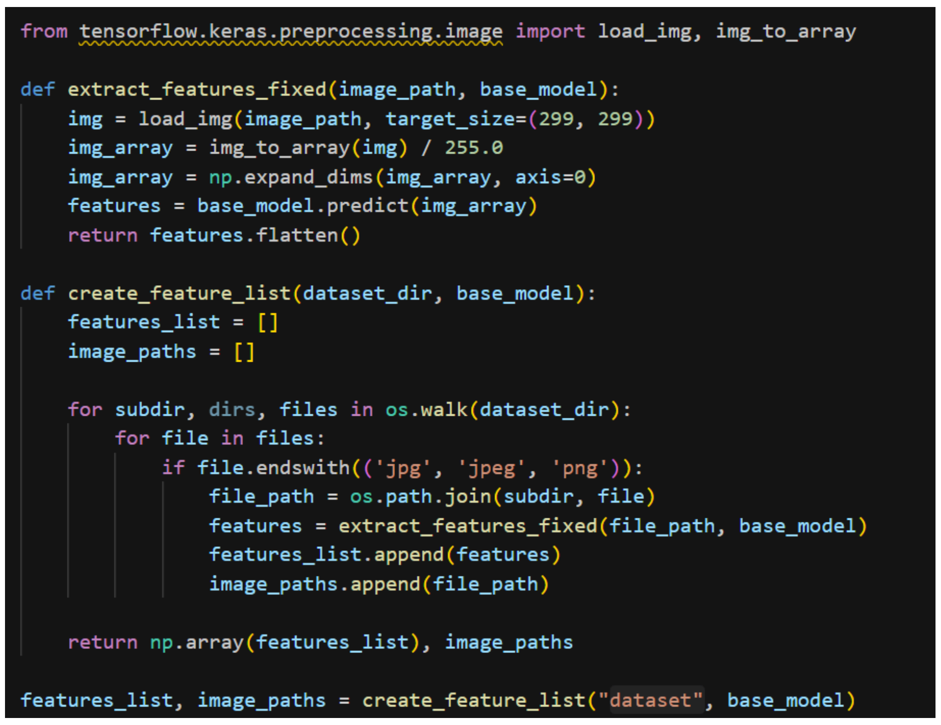
```python
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Apply SMOTE
smote = SMOTE(sampling_strategy=0.3,random_state=42)
X_resampled2, y_resampled2 = smote.fit_resample(X, y)

# Split into training and testing sets
X_train3, X_test3, y_train3, y_test3 = train_test_split(X_resampled2, y_resampled2, test_size=0.2, random_state=42)
```

C.Python Code for evaluating ML model performance metrics



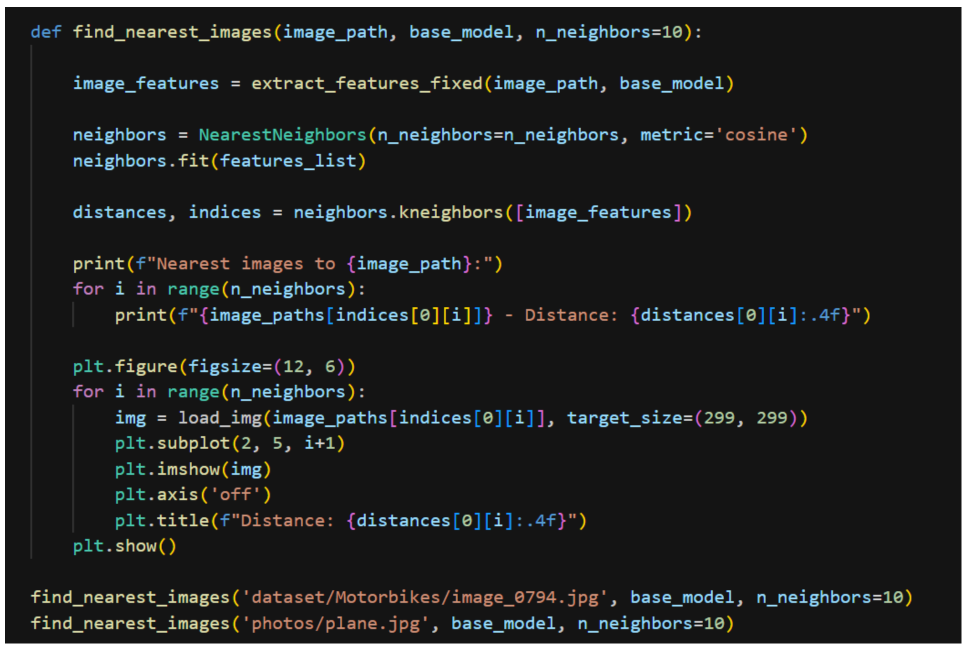
```python
from sklearn.metrics import f1_score, roc_auc_score, accuracy_score, precision_score, recall_score,balanced_accuracy_score
from sklearn.metrics import cohen_kappa_score
# Computing Metrics
accuracy_rf2 = accuracy_score(y_test3, y_pred_rf2)
balanced_accuracy_rf2 = balanced_accuracy_score(y_test3, y_pred_rf2)
precision_rf2 = precision_score(y_test3, y_pred_rf2)
recall_rf2 = recall_score(y_test3, y_pred_rf2)
f1_rf2 = f1_score(y_test3, y_pred_rf2)
y_pred_prob_rf2 = classifier_rf2.predict_proba(X_test3)[:, 1]  # Probability scores for ROC-AUC
roc_auc_rf2 = roc_auc_score(y_test3, y_pred_prob_rf2)
kappa_rf2 = cohen_kappa_score(y_test3, y_pred_rf2)




# Print Results
print(f'Accuracy: {accuracy_rf2:.4f}')
print(f'Balanced Accuracy: {balanced_accuracy_rf2:.4f}')
print(f'Precision: {precision_rf2:.4f}')
print(f'Recall: {recall_rf2:.4f}')
print(f'F1-score: {f1_rf2:.4f}')
print(f'ROC-AUC Score: {roc_auc_rf2:.4f}')
print(f"Cohen's Kappa Value: {kappa_rf2:.4f}")
```

D.Python code for extracting features from images:



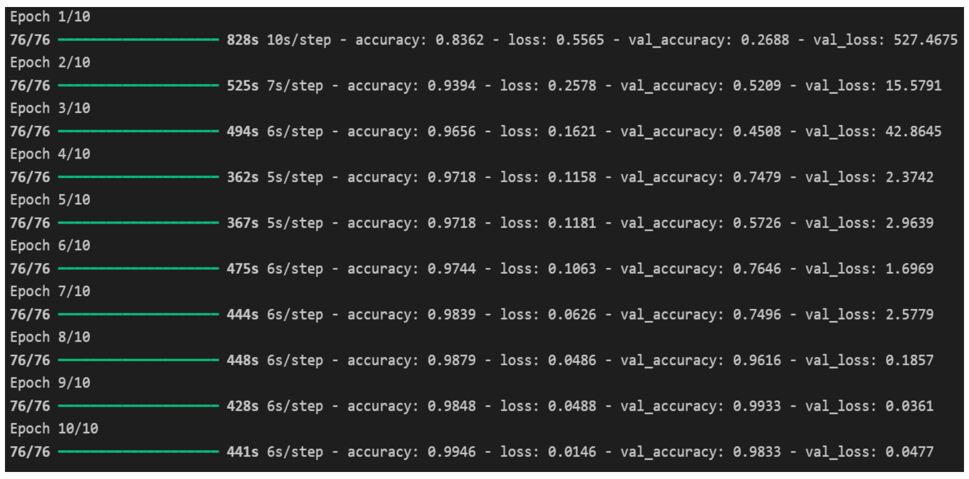
```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array

def extract_features_fixed(image_path, base_model):
    img = load_img(image_path, target_size=(299, 299))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    features = base_model.predict(img_array)
    return features.flatten()

def create_feature_list(dataset_dir, base_model):
    features_list = []
    image_paths = []

    for subdir, dirs, files in os.walk(dataset_dir):
        for file in files:
            if file.endswith(('jpg', 'jpeg', 'png')):
                file_path = os.path.join(subdir, file)
                features = extract_features_fixed(file_path, base_model)
                features_list.append(features)
                image_paths.append(file_path)

    return np.array(features_list), image_paths

features_list, image_paths = create_feature_list("dataset", base_model)
```

E.Python Code for finding nearest neighbours



```python
def find_nearest_images(image_path, base_model, n_neighbors=10):

    image_features = extract_features_fixed(image_path, base_model)

    neighbors = NearestNeighbors(n_neighbors=n_neighbors, metric='cosine')
    neighbors.fit(features_list)

    distances, indices = neighbors.kneighbors([image_features])

    print(f"Nearest images to {image_path}:")
    for i in range(n_neighbors):
        print(f"{image_paths[indices[0][i]]} - Distance: {distances[0][i]:.4f}")

    plt.figure(figsize=(12, 6))
    for i in range(n_neighbors):
        img = load_img(image_paths[indices[0][i]], target_size=(299, 299))
        plt.subplot(2, 5, i+1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"Distance: {distances[0][i]:.4f}")
    plt.show()

find_nearest_images('dataset/Motorbikes/image_0794.jpg', base_model, n_neighbors=10)
find_nearest_images('photos/plane.jpg', base_model, n_neighbors=10)
```

F. Training plus Validation set accuracy and loss for GoogleNet



```
Epoch 1/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 828s 10s/step - accuracy: 0.8362 - loss: 0.5565 - val_accuracy: 0.2688 - val_loss: 527.4675
Epoch 2/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 525s 7s/step - accuracy: 0.9394 - loss: 0.2578 - val_accuracy: 0.5209 - val_loss: 15.5791
Epoch 3/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 494s 6s/step - accuracy: 0.9656 - loss: 0.1621 - val_accuracy: 0.4508 - val_loss: 42.8645
Epoch 4/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 362s 5s/step - accuracy: 0.9718 - loss: 0.1158 - val_accuracy: 0.7479 - val_loss: 2.3742
Epoch 5/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 367s 5s/step - accuracy: 0.9718 - loss: 0.1181 - val_accuracy: 0.5726 - val_loss: 2.9639
Epoch 6/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 475s 6s/step - accuracy: 0.9744 - loss: 0.1063 - val_accuracy: 0.7646 - val_loss: 1.6969
Epoch 7/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 444s 6s/step - accuracy: 0.9839 - loss: 0.0626 - val_accuracy: 0.7496 - val_loss: 2.5779
Epoch 8/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 448s 6s/step - accuracy: 0.9879 - loss: 0.0486 - val_accuracy: 0.9616 - val_loss: 0.1857
Epoch 9/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 428s 6s/step - accuracy: 0.9848 - loss: 0.0488 - val_accuracy: 0.9933 - val_loss: 0.0361
Epoch 10/10
76/76 ━━━━━━━━━━━━━━━━━━━━ 441s 6s/step - accuracy: 0.9946 - loss: 0.0146 - val_accuracy: 0.9833 - val_loss: 0.0477
```