Fault Detection and Diagnosis in Artificial Intelligence

Information Technology and Artificial Intelligence

Grade Two

International Applied Technology Schools (IATS)

Thanks and Appreciation

USAID Workforce Egypt Project in partnership with the Ministry of Education and Technical Education extends its sincere thanks and appreciation to everyone who participated in the preparation, authoring, reviewing, and printing the content of this book, including the authors and publishers.

The content of this book including the competencies and students' assessment guides has been prepared by a large group of experts specialized in the field of information technology and artificial intelligence.

The project would sincerely like to thank the participating companies for their great efforts and transferring their professional knowledge and expertise in developing the content of this book. Companies that participated in the development of the book include:

Ahmed Diefalla Group, UMAMI For Development, Appsinnovate, Secured Smart Systems 3S, Technospace, DM Arts Academy, SOfCO, Ideas Gym, Silver Key, AWS, Integrated Knowledge Dynamics, BEDO Company, and Discovery Academy, Deep-Iris AI.

This is in addition to other companies operating in the field of information technology and artificial intelligence that took part in the development of the competencies and students' assessment guides of the specialization.

Special gratitude goes to "Nahdet Misr Publishing House" for preparing, authoring, revising, and printing the book.

Disclaimer:

"This Publication is made possible by the support of the American People through the United States Agency for International Development (USAID) and does not necessarily reflect the views of USAID or the United States Government."

Table of Content

Main Competence

Diagnosing and Troubleshooting Malfunctions of AI Software Systems

Code: ECU-ITAI-3-02

Learning Outcomes

Upon completion of this course, students will be able to:

1. **Identifying software problems of AI systems.**
   - Locate failures of AI systems.
   - Explain the relationship between observed failures and codes' problems.
   - Explain the problem.
   - Analyse problem consequences.
2. **Provide common solutions in light of understanding the inputs and outputs of the program.**
   - Test various variables until the problem is isolated.
   - Enumerate potential solutions in light of known problem-solving mechanisms.
   - Determine the most appropriate solution, according to the available tools.
3. **Implementing an action plan to apply the proposed solutions.**
   - Determine the steps and stages of the implementation process in light of the proposed solutions.
   - Estimate the time required to solve the problem.
   - Implement the proposed solution according to the strategy.
4. **Test the effectiveness of the solutions implemented in solving the problem.**
   - Test the system to make sure the problem is resolved.
   - Record the results of tests and experiments according to the organization's procedures.
   - Implement periodic maintenance strategies.
   - Prepare a report containing the features and the solution of the problem following the organization's rules.
   - Submit the report to higher managerial level.

Book Introduction

This book is like a treasure map, guiding you through the exciting world of fixing and making AI systems better. Think of AI as super-smart computer programs that help us do amazing things, like spotting diseases from X-rays or predicting when someone might stop using a service.

Each chapter here is like an adventure, teaching you how to find and solve problems in these super-smart programs. You'll learn how to spot mistakes (called errors), understand why they happen, and fix them.

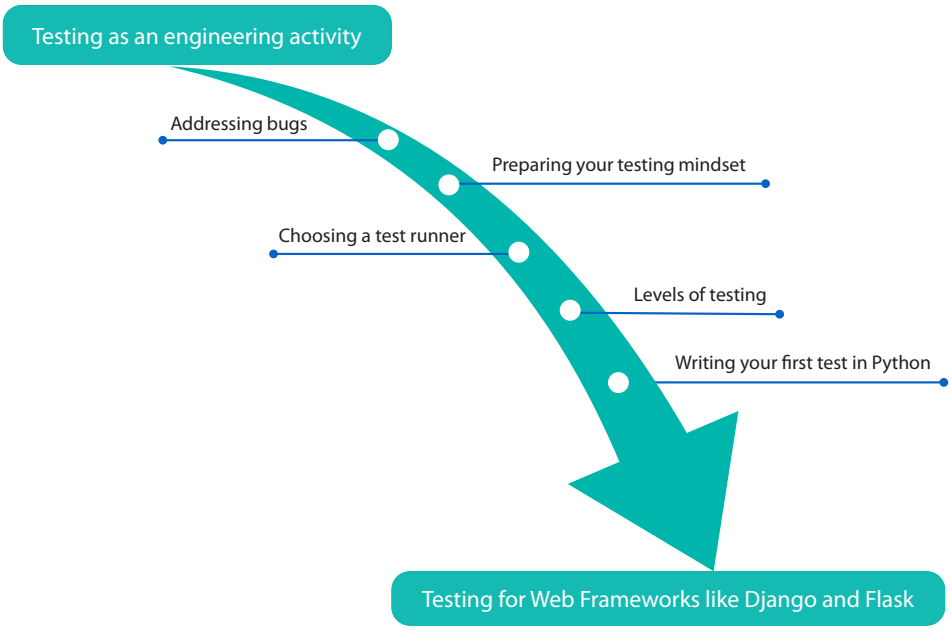
From learning why AI systems make mistakes to cool tricks for making them work even better, you'll explore real examples and fun exercises. Imagine you're a detective solving puzzles, but instead of clues, you're finding and fixing errors in these AI systems.

Starting from elucidating the importance of troubleshooting in AI to exploring practical implementations and testing methodologies, this book aims to equip readers with invaluable insights into the realm of AI systems' functionality and resilience.

Dive into each chapter to discover the intricacies of identifying errors, addressing system malfunctions, implementing effective solutions, and evaluating the impact of these measures. Through real-world examples, exercises, and assessments, this guide offers a hands-on approach to mastering the diagnosis and enhancement of AI systems across various domains, including COVID-19 detection, customer churn prediction, and autonomous vehicle technologies.

By the end, you'll be a pro at understanding and improving these clever computer programs, and you might even inspire the next big idea in AI!

Get ready to dive in and explore the exciting world of AI troubleshooting!



List of Meaning

Abbreviation:

AI	Artificial intelligence
ML	Machine Learning
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
CNN	Convolutional Neural Network
VGG	Visual Geometry Group (a type of CNN architecture)
ResNet	Residual Network (a type of deep CNN architecture)
PNG	Portable Network Graphics
SMOTE	Synthetic Minority Over-sampling Technique

Table of Definitions

Term	Definition
Cross-Validation	Technique to assess a model's performance on new data
Overfitting	When a model learns training data too well, performing poorly on new data
Underfitting	When a model fails to capture the underlying patterns in the data
Error	Difference between a predicted value and the true value
Confusion Matrix	Table used to evaluate the performance of a classification model
Data Preprocessing	Steps to clean, transform, and organize data for analysis
Model	Representation of patterns and relationships in data
Classification report	Summary of a classification model's performance metrics
Accuracy	Measure of a model's correctness on a dataset
Class Imbalance	Uneven distribution of classes in a dataset
Precision	Proportion of correctly predicted positive instances
Recall	Proportion of actual positive instances correctly predicted
Label Encoding	Converting categorical labels into numerical values
Data Augmentation	Technique to artificially increase dataset diversity
Evaluation Metrics	Measures used to assess model performance
Feature Scaling	Process of normalizing numerical features in a dataset

1st Learning Outcome

Software Problems of AI Systems

After completing this section, students should be able to:

- Locate failures of AI systems.
- Explain the relationship between observed failures and codes' problems.
- Explain the problem.
- Analyse problem consequences.

Software Problems of AI Systems

Introduction

In the rapidly evolving field of artificial intelligence (AI), the ability to diagnose and troubleshoot malfunctions within AI software systems is crucial for ensuring their effectiveness and reliability, identifying and resolving software faults is of paramount importance for ensuring the reliability and performance of AI systems, in various domains of AI, including interactive display software, embedded system software, web-based software, and smart machine firmware. By focusing on these specific domains, you will find practical examples and real-world scenarios that demonstrate the process of diagnosing and resolving software faults in AI systems.

1. Interactive Display Software: Imagine a scenario where an interactive display in a public space, such as a museum or shopping mall, starts experiencing glitches. The touch screen becomes unresponsive, and the displayed information appears distorted or incomplete. This book provides guidance on diagnosing and resolving such software faults, enabling professionals to identify the underlying issues and implement effective solutions to restore the interactive display's functionality.
2. Embedded System Software: Consider an embedded system in an autonomous vehicle that controls its navigation and decision-making processes. If the system starts exhibiting erratic behavior or fails to respond appropriately to real-time inputs, it indicates a software fault that needs to be addressed promptly. This book equips readers with the knowledge and tools to diagnose the fault and apply relevant troubleshooting techniques to rectify the issue, ensuring the safe and efficient operation of the autonomous vehicle.

3. Web-Based Software: Web-based software plays a crucial role in various AI applications, such as online recommendation systems or natural language processing algorithms. However, if the web-based software starts encountering performance issues, such as slow response times or incorrect output generation, it can severely impact the user experience and the effectiveness of AI algorithms. This book offers insights into diagnosing and resolving software faults specific to web-based AI systems, empowering professionals to identify the root cause of the problem and implement effective solutions.

4. Smart Machine Firmware: Smart machines, such as industrial robots or intelligent surveillance systems, rely on firmware to execute their tasks accurately and efficiently. If the firmware malfunctions, it can lead to unpredictable behavior or a complete system failure. This book covers the diagnosis and resolution of software faults in smart machine firmware, enabling engineers and technicians to identify the faulty code, debug it, and restore the proper functioning of the smart machine.

The Importance of Troubleshooting and Diagnosing in AI

- **Overview of AI systems:** AI presence in various domains, such as healthcare, finance, and transportation. It emphasizes the critical role that AI plays in enabling automation, prediction, and decision-making processes.

1. **Healthcare:** In the healthcare industry, AI systems are utilized for various purposes, such as diagnosing diseases, predicting patient outcomes, and personalizing treatment plans. Imagine a scenario where a malfunction in the AI software responsible for diagnosing diseases leads to incorrect diagnoses or missed critical symptoms.
2. **Finance:** AI algorithms are extensively used in financial institutions for tasks like fraud detection, risk assessment, and algorithmic trading. Suppose there is a software
3. malfunction in an AI-powered fraud detection system that incorrectly flags legitimate transactions as fraudulent, causing inconvenience and financial losses for customers.
4. **Transportation:** AI is revolutionizing the transportation industry with applications like autonomous vehicles, traffic management systems, and predictive maintenance. Consider a situation where an AI-powered autonomous vehicle experiences unexpected behavior due to a software malfunction, such as misinterpreting traffic signals or failing to respond appropriately to changing road conditions.

Understanding software malfunctions

Understanding software malfunctions is crucial for AI systems to comprehend the potential consequences they can have. Software malfunctions have resulted in costly errors, compromised security, or biased decision-making. Let's apply this concept to the three examples mentioned earlier:

1 Healthcare

In the healthcare industry, if the software misinterprets certain symptoms or fails to consider crucial medical information, it may provide inaccurate diagnoses. This can result in delayed or inappropriate treatments, potentially endangering patients' lives and impacting their well-being. Understanding such software malfunctions is essential to develop effective troubleshooting and diagnostic techniques that can address these issues and ensure reliable and accurate disease diagnoses.

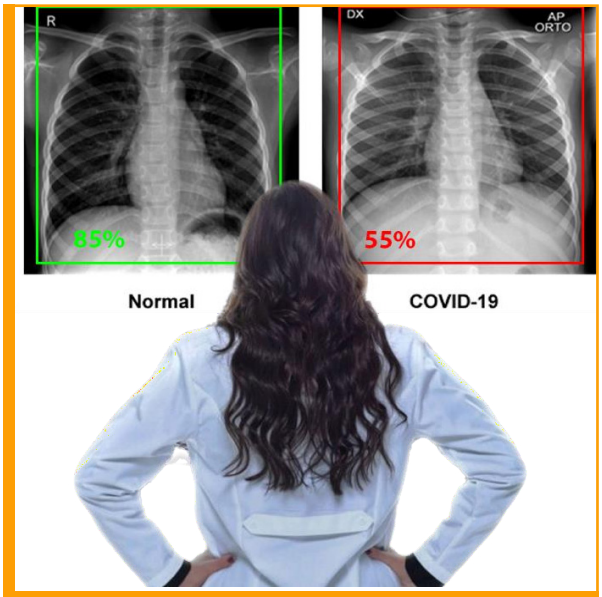


Figure (1): Misdiagnoses, AI algorithms can make mistakes in diagnosing diseases

2 Finance

Within the realm of finance, a malfunction in an algorithmic trading system can lead to erroneous transactions or financial losses. Additionally, a software glitch in a fraud detection system can result in legitimate transactions being falsely flagged as fraudulent, causing inconvenience for customers and damaging their trust in the financial institution. Understanding these software malfunctions enables financial organizations to implement robust troubleshooting and diagnostic strategies to rectify the issues promptly, protect financial assets, and maintain the integrity of their systems.

Figure (2): Online fraud, that generates fake financial statements or by manipulating stock prices

3 Transportation

In the transportation industry, if an autonomous vehicle experiences a software malfunction, it may exhibit erratic behavior on the road, jeopardizing the safety of passengers and other road users. Understanding the potential software malfunctions that can occur in autonomous vehicle systems, such as faulty perception algorithms or decision-making processes, is crucial for developing effective troubleshooting and diagnostic methods. By identifying and addressing these malfunctions, the transportation industry can enhance the safety and reliability of autonomous vehicles, ensuring smooth and secure transportation operations.

Figure (3): Self-driving cars crashing

By understanding the potential consequences of software malfunctions in these AI systems, we recognize the significance of effective troubleshooting and diagnostics. This knowledge motivates us to develop robust strategies and methodologies to identify, diagnose, and resolve software faults in order to enhance the reliability and performance of AI systems in various domains.

Exercise 1

1. Why is troubleshooting and diagnosing important in AI systems?
2. In what ways can performance issues in web-based AI systems impact the user experience and AI algorithm effectiveness?
3. How might a software malfunction in an AI-powered autonomous vehicle pose safety risks on the road?

Steps To Test and Troubleshoot AI Systems

Testing and troubleshooting AI systems can be a complex and iterative process.

Here are some general steps to test and troubleshoot AI systems effectively:

1. Define Clear Objectives:

Clearly define the objectives of the AI system and what it is intended to achieve. This will provide a basis for evaluating the system's performance.

2. Data Validation and Preprocessing:

Ensure that the input data used for training and testing the AI system is accurate, relevant, and properly preprocessed. Data validation helps prevent issues arising from flawed or biased data.

3. Unit Testing:

Begin with unit testing of individual components and algorithms within the AI system. Test each part in isolation to ensure it functions as intended.

4. Integration Testing:

After testing individual components, integrate them into the AI system and conduct integration testing. Verify that the different components work together harmoniously.

5. Functional Testing:

Test the AI system's overall functionality against various scenarios and use cases to ensure it produces the expected outputs.

6. Performance Testing:

Evaluate the AI system's performance in terms of speed, accuracy, and resource utilization. Test the system under different workloads and stress conditions.

7. Validation with Real-World Data:

Test the AI system with real-world data that it is likely to encounter in its operational environment. Validate its performance against ground truth or expert judgments.

8. Error Analysis and Debugging:

When issues arise, conduct thorough error analysis and debugging. Identify the root causes of the problems and address them systematically.

9. Feedback Loop and Continuous Improvement:

Gather feedback from users, experts, and other stakeholders to identify areas for improvement. Incorporate this feedback into the system to continuously enhance its performance.

10. Documentation and Monitoring:

Maintain detailed documentation of tests conducted, identified issues, and their resolutions. Implement monitoring mechanisms to detect anomalies and performance degradation in real- time.

11. User Acceptance Testing:

Involve end-users and stakeholders in the testing process to gather feedback and assess user satisfaction.

Remember that troubleshooting AI systems is an ongoing process. As the system evolves and faces new challenges, it's essential to continue testing, monitoring, and refining the system to maintain its effectiveness and reliability.

Errors In AI Systems

can occur at various levels, including data preparation, algorithm implementation, and model training.

Here, let's focus on two specific examples of errors in AI systems:

- **Syntax Errors:** Syntax errors in AI systems refer to mistakes in the code's structure that violate the rules of the programming language being used. These errors prevent the code from being compiled or executed correctly.

- **Example of a syntax error in Python:**

```
# Syntax error - Missing closing parenthesis
print("Hello, World!"
```

In this example, the missing closing parenthesis in the print statement causes a syntax error. When executing the code, Python will raise a syntax error message, indicating the location of the issue.

- **Logical Errors:** Logical errors occur when the code's logic is flawed, leading to incorrect calculations or decisions. The code may run without raising any errors, but it produces unexpected or inaccurate results.
- **Example of a logical error in AI system code (Python):**

```
# Incorrect logic - Division instead of multiplication
def calculate_area(radius):
    pi = 3.14
    return pi / 2 * radius ** 2
```

In this example, the function is intended to calculate the area of a circle given its radius. However, due to the incorrect logic of using division instead of multiplication, the calculated area will be half of the correct value.

Logical errors are especially tricky because they do not cause the code to crash or raise syntax errors, making them harder to detect. They require careful examination of the code and thorough testing to identify and fix them.

It's important to address code errors promptly in AI systems, as they can lead to inaccurate predictions, unreliable decision-making, or even safety concerns in critical applications. Proper debugging, testing, and code review processes help minimize code errors and ensure the AI system performs as intended.

To determine the location of the fault, we need to analyze the model's performance and its predictions on the test dataset. Here are the steps to perform this analysis:

1. Analyze Misclassifications:

First, we need to analyze the misclassified samples from the test dataset. Extract the input images and the corresponding predicted labels (y_pred) and ground truth labels (y_test).

2. Compare Misclassified Samples:

For each misclassified sample, compare the predicted label with the ground-truth label. Identify the specific classes that are misclassified and the frequency of such misclassifications.

3. Examine Confusion Matrix:

Generate a confusion matrix using the predicted labels (y_pred) and ground truth labels (y_test). The confusion matrix will provide an overview of the model's performance on each class and the nature of misclassifications.

4. Error Localization:

Based on the confusion matrix and the analysis of misclassified samples, determine the location of the fault:

a) If the misclassifications are concentrated in specific classes (e.g., COVID-19 and Viral Pneumonia), it may indicate that the trained model is struggling to correctly identify these classes. In such cases, the fault lies in the trained model, and further improvements are needed,such as using more advanced models, tuning hyperparameters, or increasing the dataset size.

b) If misclassifications are distributed across all classes and there is no pattern of specific misclassifications, it may indicate issues with the code logic, data preprocessing, or model evaluation process. In this case, re-evaluate the data preprocessing steps, model training, and evaluation procedures for potential errors.

5. Overfitting or Underfitting:

Check for signs of overfitting or underfitting. Overfitting occurs when the model performs well on the training data but poorly on the test data, indicating that the model has learned to memorize the training data instead of generalizing well to new data. Underfitting occurs when the model fails to learn the underlying patterns in the data and performs poorly on both training and test datasets.

6. Cross-Validation:

Perform cross-validation to get a more robust assessment of the model's performance. If the model consistently performs poorly on different folds of the data, it may indicate fundamental issues with the model's architecture or data preprocessing.

7. Evaluate Training Process:

Examine the model's training process, including the loss curve, learning rate, and validation performance over epochs. This can provide insights into the training dynamics and potential issues that may affect the model's performance.

® By conducting these analyses, you can gain a better understanding of the location of the fault and identify areas that require improvement.

■ **Let's illustrate some examples with real projects:**

AI system for COVID-19 detection using X-ray images

`https://www.kaggle.com/code/thura1601/covid19-xray-w-ml-and-dl`

In the context of the AI system for COVID-19 detection, the symptoms could include:

- **Inconsistent Predictions:** The AI system might provide inconsistent or unstable predictions for the same X-ray image when run multiple times.
- **High False Positives or False Negatives:** The AI system may generate a significant number of false-positive predictions (predicting COVID-19 when the X-ray is normal) or false-negative predictions (missing COVID-19 cases in X-rays that show symptoms).

- **Low Accuracy and Performance:** The overall accuracy of the AI system in detecting COVID-19 from X-ray images might be lower than expected, impacting its reliability.
- **Unexplained Model Behavior:** The AI model's behavior might be difficult to interpret or explain, making it challenging to understand how it arrived at specific predictions.
- **High Variance in Predictions:** The AI system might exhibit high variance in predictions for different X-ray images, even if they show similar patterns.
- **Slow Prediction Times:** The AI system's prediction process might take an unusually long time, causing delays in getting results.
- **Sensitivity to Image Quality:** The AI system's performance might be significantly affected by the quality of the X-ray images, leading to varying results for different image resolutions or noise levels.
- **Stability Issues:** The AI system could experience crashes or instability during model training or prediction phases.

Identifying and understanding these symptoms is crucial, as they can provide valuable insights into the root causes of potential issues in the AI system. Once the symptoms are clearly defined, further analysis, debugging, and troubleshooting can be performed to diagnose and address the underlying problems. This process helps improve the performance, accuracy, and reliability of the AI system, ensuring its effectiveness in COVID-19 detection from X-ray images.

Analyzes the Results Associated with the Problem:

refers to the process of examining the outcomes and consequences that arise due to the identified problems in the AI system. It involves studying the impact of these issues on the system's performance, accuracy, reliability, and overall effectiveness in achieving its intended objectives.

In the specific context of a COVID-19 detection AI system using X-ray images, analyzing the results associated with the problem can have additional meanings:

- **Impact on Healthcare Decisions:** The accuracy of COVID-19 detection is critical for medical decisions and patient care. The analysis assesses how system errors affect the appropriate diagnosis and treatment plans.
- **Public Health Implications:** The reliability of the COVID-19 detection AI system can have broader public health implications, affecting the spread of the disease and resource allocation in healthcare facilities.
- **Ethical Considerations:** The analysis evaluates any potential bias in the AI system's predictions, which could disproportionately impact specific demographic groups or patient populations.

- **Integration into Clinical Workflow:** The analysis examines how well the AI system integrates into the existing clinical workflow and whether it provides valuable assistance to healthcare professionals.
- **Timeliness of Results:** In a pandemic scenario like COVID-19, timely detection is crucial. The analysis addresses any delays or issues in the AI system's prediction times.

In summary, analyzing the results associated with the problem in a COVID-19 detection project involves evaluating the system's limitations, performance metrics, impact on healthcare decisions, public health implications, ethical considerations, and integration into the clinical workflow. This analysis helps in refining the AI system, addressing the identified issues, and ensuring that it contributes effectively to COVID-19 detection efforts.

Project Documentation - COVID Detection AI System

1 Introduction

The COVID Detection AI System is designed to diagnose and detect COVID-19, Normal cases, Lung Opacity (Non-COVID lung infection), and Viral Pneumonia from chest X-ray images. This document outlines the development, implementation, and evaluation of the AI system, along with the results obtained during testing.

2 Dataset

The AI system utilizes a dataset consisting of 21,165 chest X-ray images. The dataset is divided into four main classes:

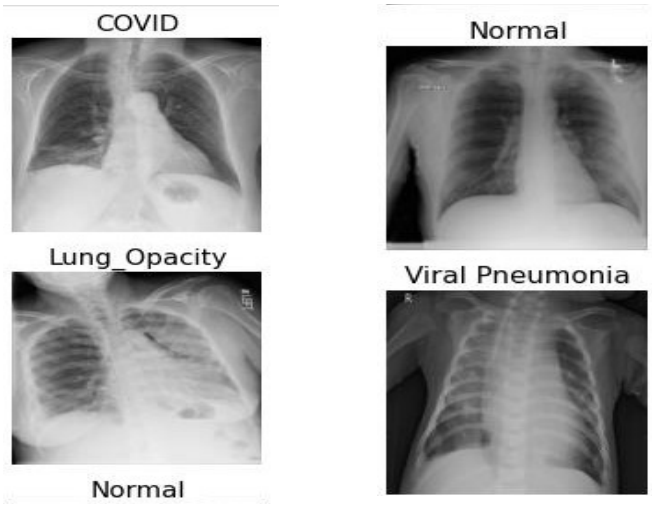


Figure (4): Dataset main classes

All images are in Portable Network Graphics (PNG) format, and their resolution is standardized to 299x299 pixels.

3 Model Selection

The AI system employs a Multinomial Naive Bayes classifier from the scikit-learn library for the classification task.

```
from sklearn.naive_bayes import MultinomialNB

# Train the MultinomialNB
nb_model = MultinomialNB()
nb_model.fit(X_train,y_train)
```

4 Data Preprocessing

Before feeding the images into the model, the following preprocessing steps are performed on each image:

- **Grayscale Conversion:** Images are converted to grayscale to reduce the color dimension and simplify processing.
- **Resize:** All images are resized to a consistent resolution of 64x64 pixels, enabling compatibility with the chosen model.

```
for image in tqdm(data['path']):
    img=Image.open(image)
    img=ImageOps.grayscale(img)
    img=img.resize((64,64))
    img=np.asarray(img)
    img=img.reshape((64,64,1))
    pixel_img.append(img)
```

```
100%|██████████| 21165/21165 [02:30<00:00, 141.03it/s]
```

5 Model Training and Evaluation

The Multinomial Naive Bayes model is trained on the preprocessed dataset. The evaluation metrics for the model's performance are calculated using the test dataset. The following evaluation metrics are obtained:

- Accuracy: 0.5563
- Classification report (Precision, Recall, F1-score, Support) for each class:

Class	Precision	Recall	F-1score	Support
COVID-19	0.37	0.88	0.52	269
Normal: Precision	0.77	0.51	0.61	2038
Lung Opacity	0.63	0.59	0.61	1203
Viral Pneumonia	0.33	0.52	0.41	723

```
Accuracy: 0.556343019135365
Classification report for classifier MultinomialNB():
              precision    recall  f1-score   support

           0       0.77      0.51      0.61      2038
           1       0.33      0.52      0.41       723
           2       0.63      0.59      0.61      1203
           3       0.37      0.88      0.52       269

    accuracy                           0.56      4233
   macro avg       0.53      0.62      0.54      4233
weighted avg       0.63      0.56      0.57      4233
```

6 Discussion

1. Identify scenarios in which the system failed.

- **Misclassification of COVID-19 Cases:** The system shows relatively low precision (0.33) for the 'COVID' class. This means that when the model predicts a sample as COVID-19 positive, it is correct for only 33% of the time. The low precision indicates a high number of false-positive predictions for COVID-19 cases, leading to misdiagnoses and potentially causing unnecessary anxiety and stress to patients.
- **Misclassification of Normal Cases:** The recall for the 'Normal' class is 0.51, indicating that the model is missing around 49% of actual 'Normal' cases. This suggests a relatively high number of false-negative predictions for the 'Normal' class, where the model incorrectly classifies some 'Normal' cases as other classes, possibly including COVID-19 or Viral Pneumonia.
- **Misclassification of Viral Pneumonia Cases:** The precision for the 'Viral Pneumonia' class is 0.37, meaning that the system tends to have a high number of false-positive predictions for Viral Pneumonia cases. This indicates that the model may incorrectly classify some 'Normal' or COVID-19 cases as 'Viral Pneumonia,' leading to misdiagnoses.

- **Misclassification of Lung Opacity Cases:** The recall for the 'Lung Opacity' class is 0.59, indicating that the model is missing approximately 41% of actual 'Lung Opacity' cases. This suggests that the system is not effectively identifying some instances of 'Lung Opacity,' leading to false-negative predictions.

2. Verify that the expected results in the documentation are accurate and logical.

- The model's overall accuracy is 0.5563, which is relatively modest. This indicates that the system's ability to generalize and make accurate predictions on unseen data is limited. It may struggle with detecting COVID-19, Normal, Lung Opacity, and Viral Pneumonia cases from new and diverse datasets.

3. Determine the location of the fault, whether it is in the code logic or the trained model.

A Imbalanced Classes

The dataset contains imbalanced classes, which means that some classes have significantly more samples than others. This can lead to biased model performance and affect the ability of the model to correctly predict the minority classes (e.g., COVID-19 and Viral Pneumonia).

Addressing class imbalance is important for improving model accuracy and performance.

B Model Selection

The chosen model is Multinomial Naive Bayes, which is commonly used for text classification with discrete features. However, this model may not be the most appropriate for image

classification tasks like this one. More advanced models, such as Convolutional Neural Networks (CNNs) are generally better suited for image classification problems.

AI system for Churn prediction:

`https://github.com/yash2189/Customer-Churn-Prediction-ML`

The Churn Prediction AI System proves valuable for businesses seeking to retain customers and make data-driven decisions to reduce churn rates. Continuous monitoring and improvements will help maintain its effectiveness in the dynamic business environment

Possible Errors:

1 Class Imbalance

- Uneven distribution of the target variable (churned vs. not churned customers).
- A significantly higher number of instances in one class (e.g., non-churned customers) compared to the other (e.g., churned customers).

- **Potential issues:** Biased model predictions, lower recall for the minority class (churned customers).

2 Low F1 Score

- A trade-off between precision and recall.
- **Potential issues:** Difficulty in achieving both high precision and high recall simultaneously, indicating an imbalanced model performance.

3 Low ROC AUC Score

- The model's ability to distinguish between classes is not much better than random guessing.
- **Potential issues:** The model may struggle to differentiate between churned and non-churned customers effectively.

4 Model Overfitting

- The model performs well on the training data but poorly on the unseen test data.
- **Potential issues:** Reduced generalization ability, leading to poor performance on new data.

5 Inaccurate Predictions

- The model's predictions do not align with the actual customer churn status.
- **Potential issues:** Model inadequacies, data quality issues, or insufficient feature representation.

6 Lack of Feature Importance

- Some features may not be contributing significantly to the model's predictions.
- **Potential issues:** Inclusion of irrelevant or redundant features, leading to noise in the model.

7 Limited Discrimination Power

- The model struggles to differentiate between churned and non-churned customers, especially in high-dimensional data.
- **Potential issues:** Model insensitivity to important features or lack of feature discrimination.

8 Underfitting

- The model's performance is consistently low on both the training and test data.
- **Potential issues:** The model may be too simplistic to capture the underlying patterns in the data.

These symptoms can help identify potential challenges and guide further exploration, data preprocessing, feature engineering, and model selection to improve the model's performance for churn prediction in the telecom company. Addressing these symptoms may lead to more accurate predictions and better retention strategies for the business.

Analyzes the Results Associated with the Problem:

1. **Missed Opportunities:** Inaccurate predictions can result in missed opportunities to retain valuable customers. If the system fails to identify customers at risk of churn, businesses may not be able to take timely actions to prevent churn.
2. **Negative Impact on Customer Experience:** An ineffective churn prediction system might not identify the root causes of churn accurately. As a result, businesses may fail to address critical issues impacting customer experience and loyalty.
3. **Loss of Competitive Advantage:** In a competitive market, businesses that cannot effectively retain customers risk losing their competitive edge to competitors with better churn prediction strategies.
4. **Inability to Optimize Marketing Efforts:** If the churn prediction system fails to identify high- value customers at risk of churn, businesses may continue with generalized marketing efforts, leading to suboptimal returns on marketing investments.

In summary, a poorly implemented churn prediction project can lead to unreliable predictions, wasted resources, customer dissatisfaction, and missed opportunities for retention. To ensure the project's success, it is essential to carefully design and validate the model, address data quality issues, and continuously improve the system based on feedback and evolving business needs.

Proper implementation and continuous monitoring are critical to achieving meaningful impacts on customer churn reduction and enhancing customer retention in the telecom domain.

Project Documentation – Customer Churn prediction AI System

- **Project Overview:** This Churn Prediction project aims to develop a machine learning model that predicts customer churn in a telecom company. Customer churn refers to the loss of customers or subscribers, and predicting it can help businesses identify potential churners and take proactive actions to retain valuable customers.
- **Data Description:** The project uses a dataset named "Churn_Modelling.csv," which contains customer information for the telecom company. The dataset includes various features such as "CreditScore," "Geography," "Gender," "Age," "Tenure," "Balance," "NumOfProducts," "HasCrCard," "IsActiveMember," and "EstimatedSalary." The target variable is "Exited," indicating whether a customer churned (1) or not (0).

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	France	Female	42	2	0.00	1	1	1	101348.88
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	502	France	Female	42	8	159660.80	3	1	0	113931.57
3	699	France	Female	39	1	0.00	2	0	0	93826.63
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10

- **Class Imbalance:** After exploring the target variable, it is observed that the dataset suffers from class imbalance, where the number of instances in one class is significantly higher than the other. This imbalance can affect model performance and lead to biased predictions.

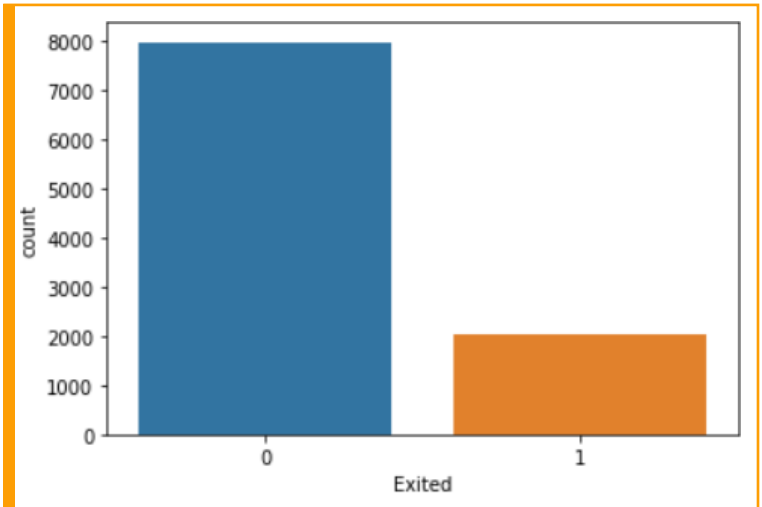


Figure (5): Class Imbalance

- **Data Preprocessing:** Before training the model, categorical features ("Gender" and "Geography") are encoded using LabelEncoder. It is essential to convert categorical variables to numerical form for model training. However, for nominal categorical features like "Geography," One-Hot Encoding might be preferred to avoid introducing artificial ordinal relationships.
- **Model Training and Evaluation:** A Logistic Regression model is chosen as the predictive model for churn prediction. Logistic Regression is a simple yet effective binary classification algorithm. After training the model on the training data, it is evaluated using various evaluation metrics like Accuracy, Precision, Recall, F1 Score, and ROC AUC Score. These metrics provide insights into different aspects of the model's performance, such as accuracy, ability to identify positive cases, trade-off between precision and recall, and discrimination power between classes.
- **Results:** The model's performance evaluation shows the following metrics:
  - **Accuracy:** 79.86%
  - Precision: 46.49%
  - **Recall:** 7.60%
  - F1 Score: 13.07%
  - **ROC AUC Score:** 52.71%

Discussion:

1 Identify Scenarios In Which the System Failed

Accuracy (79.86%):

- The model achieved an accuracy of approximately 79.86%. It correctly predicted the class labels for about 79.86% of the test samples.
- While accuracy is an important metric, it may not be sufficient for imbalanced datasets like this one, where one class dominates the other.
- Further analysis is required to understand if this accuracy is better than a random guess and to assess the model's overall performance.

Precision (46.49%):

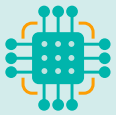
- Precision is the percentage of correctly predicted positive samples (customers who "Exit") out of all the samples that the model predicted as positive.
- The precision of approximately 46.49% suggests that of all the customers predicted to "Exit," only around 46.49% actually exited.
- A relatively low precision indicates a significant number of false positives, leading to misclassifying non-churned customers as churned.

Recall (7.60%):

- Recall, also known as sensitivity or true positive rate, is the percentage of actual positive samples (customers who "Exit") that the model correctly identified.
- The recall of approximately 7.60% indicates that the model identified only around 7.60% of the actual customers who exited.
- A low recall suggests that the model struggles to correctly identify customers who actually churned.

F1 Score (13.07%):

- The F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when dealing with imbalanced datasets.
- The F1 score of approximately 13.07% suggests a trade-off between precision and recall. The model is not effectively capturing both true positives and true negatives.



ROC AUC Score (52.71%):

The ROC AUC score measures the model's ability to distinguish between positive and negative instances across different threshold settings.

A ROC AUC score of approximately 52.71% suggests that the model's ability to discriminate between churned and non-churned customers is only slightly better than random guessing.

1 Determine the Location of the Fault

Class Imbalance:

- The target variable "y" (churned vs. not churned customers) shows a significant class imbalance with a much larger number of instances for class 0 (non-churned) compared to class 1 (churned).

The class distribution is as follows: 0 (non-churned) - 7963 instances, 1 (churned) - 2037 instances.

- This class imbalance can significantly impact model performance, leading to biased predictions and a reduced ability to correctly identify churned customers.

Label Encoding for Categorical Features:

- The code uses LabelEncoder to encode the "Gender" and "Geography" columns. However, this method is not suitable for nominal categorical features like "Geography" since it may introduce artificial ordinal relationships.
- For nominal categorical features like "Geography," One-Hot Encoding is more appropriate to avoid any ordinal assumptions.
- Using LabelEncoder for "Geography" might lead to suboptimal model performance.

Scaling Numerical Features:

- Code must Apply feature scaling to the numerical columns to bring all the features to a similar scale. Standardization is commonly used for this purpose.

Model Evaluation:

- The evaluation metrics, such as low recall (0.076) and precision (0.465), suggest that the model may not be effectively predicting churn. The F1 score (0.131) is also relatively low, indicating a trade-off between precision and recall.
- The ROC AUC score (0.527) suggests that the model's ability to discriminate between churned and non-churned customers is only slightly better than random guessing.

Exercise 2 Road Sign Detection for Autonomous Vehicles

- **Introduction:** The Road Sign Detection AI System aims to identify and classify various road signs from camera images taken by an autonomous vehicle. The system is crucial for safe navigation and decision-making during autonomous driving.
- **Dataset:** The AI system uses a labeled dataset containing camera images of road signs. The dataset is divided into several classes, including Stop signs, Speed limit signs, Yield signs, and more. Each image is associated with the correct road sign label. https://www.kaggle.com/datasets/andrewmvd/road-sign-detection

1. List a number of possible logical and code errors that may face this system
2. Discuss the impact of data preprocessing errors on the model's performance and how resizing and data augmentation techniques addressed them.
3. Analyze how different model architectures affected the system's ability to detect road signs accurately and handle overfitting/underfitting.
4. Evaluate how changes in training configurations affected training times and model convergence.
5. Address the significance of road sign detection in autonomous vehicles and the potential consequences of misclassification.

ASSESSMENT

1. Identify Code Error or Logical Error

You are working on an AI system for flower classification using images. The goal is to correctly classify different types of flowers into their respective categories. During the development process, you encounter several issues that impact the system's performance.

For each of the following problems, determine whether it is a "Code Error" or a "Logical Error" and briefly explain your choice.

- **Issue:** The AI model consistently outputs the same incorrect label for all flower images in the test set.
  - Answer:.....
  - Explanation:.....
- **Issue:** The training process takes an unusually long time, significantly slowing down the model development and optimization.
  - Answer:.....
  - Explanation:.....
- **Issue:** The AI system achieves very high accuracy on the training data but performs poorly on new, unseen flower images.
  - Answer:.....
  - Explanation:.....
- **Issue:** The AI system frequently misclassifies images of roses as daisies and vice versa.
  - Answer:.....
  - Explanation:.....
- **Issue:** The model exhibits high variance in predictions for different images of the same flower type.
  - Answer:.....
  - Explanation:.....
- **Issue:** The AI system often predicts flowers as "Unknown" with no specific category label.
  - Answer:.....
  - Explanation:.....

ASSESSMENT

2. What are some general steps to effectively test and troubleshoot AI systems?
3. What is the difference between syntax errors and logical errors in AI systems?
4. How do you determine the location of the fault in an AI system's performance analysis?
5. What steps can be taken to address overfitting or underfitting in an AI model?
6. How does cross-validation help in assessing an AI model's performance?
7. How can you use a confusion matrix to evaluate an AI model's performance in multi- class classification?
8. Why is it important to address code errors promptly in AI systems?

2nd Learning Outcome

Enhancing AI System Performance

After completing this section, students should be able to:

- Test various variables until the problem is isolated.
- Enumerate potential solutions in light of known problem-solving mechanisms.
- Determine the most appropriate solution, according to the available tools.

Enhancing AI System Performance

COVID Detection AI System

To enhance the AI system's performance, the following improvements are suggested:

Data Augmentation:

Augment the dataset using techniques like rotation, flipping, and scaling to increase the diversity of the training data, especially for COVID-19 and Viral Pneumonia cases.

Model Selection:

Since this is an image classification task, consider using deep learning models like CNNs, which have shown great performance in image recognition tasks. Transfer learning using pre-trained models (e.g., VGG, ResNet, or MobileNet) can also be beneficial when you have limited training data.

Certainly! In the context of deep learning and image classification, VGG, ResNet, and MobileNet are all well-known model architectures that have made significant contributions to the field. Here's a brief overview of each:

1 VGG (Visual Geometry Group)

VGG is a deep convolutional neural network architecture that was introduced in 2014. It is known for its simplicity and uniformity in architecture. VGG consists of multiple convolutional layers with small 3x3 filters, followed by max-pooling layers to reduce spatial dimensions. It achieved impressive performance on various image classification tasks, but its main drawback is its large number of parameters, making it computationally expensive.

2 ResNet (Residual Network)

ResNet is a groundbreaking architecture introduced in 2015 to address the vanishing gradient problem in very deep neural networks. It introduced the concept of residual blocks, where each block contains a skip connection that allows the network to directly learn the residual (difference) between input and output. This enables the training of extremely deep networks, with variants like ResNet-50, ResNet-101, etc., achieving state-of-the-art performance on tasks like image classification and object detection.

3 MobileNet

MobileNet is designed with mobile and embedded applications in mind, focusing on efficiency while maintaining reasonable accuracy. Introduced in 2017, MobileNet employs depth-wise separable convolutions, which split the standard convolution into two separate layers: depth-wise convolution and point-wise convolution. This significantly reduces computation and parameter count while still preserving important features. MobileNet is often used when resource constraints are a concern.

Each of these architectures has its strengths and weaknesses, and their suitability depends on the specific requirements of the task and the available computational resources. When selecting a model architecture, factors like model complexity, computational efficiency, and desired accuracy should all be considered.

Class Imbalance Handling:

Address the class imbalance issue by employing techniques such as oversampling the minority classes, under sampling the majority class, or using class weights during model training. This will help the model pay more attention to the minority classes during training.

Use Larger Dataset:

If possible, consider acquiring more data for each class to improve the model's ability to learn and generalize across classes.

The most suitable solutions:

We will choose from the proposed options:

- Class Imbalance Handling,
- Data Augmentation
- using Convolutional Neural Networks (CNN).

By selecting these solutions, the COVID detection AI system will effectively handle class imbalance and leverage the capabilities of CNNs for image classification. The result will be enhanced performance and improved accuracy in detecting and classifying COVID-19 cases from chest X-ray images.

Sure! Let's go through how to apply the selected solutions,

EXAMPLE

Suppose we have a dataset of chest X-ray images for COVID-19 detection. The dataset contains 4 classes: 'Normal,' 'COVID-19,' 'Lung Opacity,' and 'Viral Pneumonia.' However, the dataset suffers from class imbalance, with 'Normal' cases being much more prevalent than the other classes. We want to enhance the AI system's performance to achieve better accuracy and correctly identify COVID-19 cases.

Step 1: Class Imbalance Handling

■ Resampling Techniques:

First, make sure you have the imbalanced-learn library installed. You can install it using pip:

A Oversampling

Duplicate samples from the minority class to balance the class distribution.

```
from imblearn.over_sampling import RandomOverSampler

# Create an oversampler object
oversampler = RandomOverSampler(sampling_strategy='auto', random_state=42)

# Resample the data
X_oversampled, y_oversampled = oversampler.fit_resample(X, y)
```

B Undersampling

Randomly remove samples from the majority class to balance the class distribution.

```
from imblearn.under_sampling import RandomUnderSampler

# Create an undersampler object
undersampler = RandomUnderSampler(sampling_strategy='auto', random_state=42)

# Resample the data
X_undersampled, y_undersampled = undersampler.fit_resample(X, y)
```

C SMOTE (Synthetic Minority Over-sampling Technique)

Brief Explanation:

SMOTE is a popular data augmentation technique used to address class imbalance in classification tasks. It focuses on the minority class by generating synthetic samples to balance the class distribution. SMOTE works by creating synthetic instances along the line segments joining the existing minority class samples.

How SMOTE Works:

1. For each minority class sample, SMOTE selects its k nearest neighbors from the same class.
2. It then generates new synthetic samples by interpolating between the chosen sample and its neighbors.
3. The interpolation is controlled by a random parameter between 0 and 1, determining the mix of the chosen sample and its neighbors.

Uses and Advantages:

1. **Class Imbalance Handling:** SMOTE is specifically designed to address class imbalance problems by oversampling the minority class. This helps prevent the model from being biased towards the majority class.
2. **Improved Generalization:** By increasing the number of samples in the minority class, SMOTE helps the model learn better decision boundaries and generalize well to unseen data.
3. **Mitigates Overfitting:** In cases where overfitting is a concern, SMOTE can improve model performance by providing additional training samples.
4. **Preserves Original Data:** SMOTE generates synthetic samples based on existing minority class samples, which helps preserve the characteristics of the original data.

```
from imblearn.over_sampling import SMOTE

# Create a SMOTE object
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Resample the data
X_smote, y_smote = smote.fit_resample(X, y)
```

D Class Weighting

To address the class imbalance issue, we will implement class weighting during CNN model training. Let's assume that there are 1000 'Normal' cases, 200 'COVID-19' cases, 300 'Lung Opacity' cases, and 150 'Viral Pneumonia' cases in the dataset.

We will assign weights to each class inversely proportional to their frequency:

- Weight for 'Normal' class: 1 / (number of 'Normal' cases) = 1 / 1000
- Weight for 'COVID-19' class: 1 / (number of 'COVID-19' cases) = 1 / 200
- Weight for 'Lung Opacity' class: 1 / (number of 'Lung Opacity' cases) = 1 / 300
- Weight for 'Viral Pneumonia' class: 1 / (number of 'Viral Pneumonia' cases) = 1 / 150

E Data Augmentation

Apply image augmentation techniques (e.g., rotation, flipping, zooming) to increase the number of samples in the minority class.

Step 2: Model Selection - Using Convolutional Neural Networks (CNNs)

We will replace the previously used Multinomial Naive Bayes model with a model more suitable to its nature like CNN.

CNN is a type of deep learning neural network primarily used in image recognition and processing that is specifically designed to learn spatial hierarchies of features automatically and adaptively from input images.

The CNN architecture will consist of convolutional layers to extract features, pooling layers for down sampling, and fully connected layers for classification.

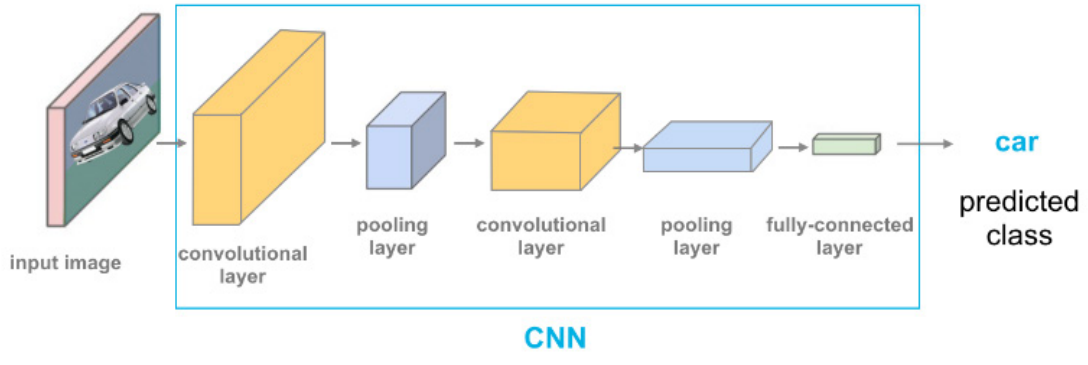


Figure (6): CNN architecture

Conventional Layers:

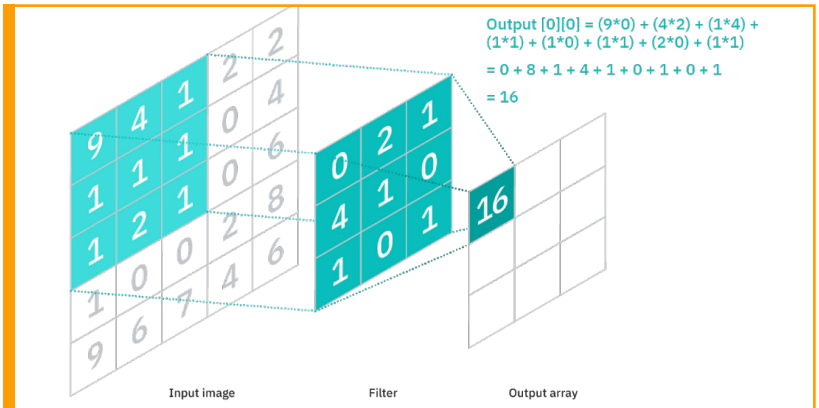


Figure (7): Conventional layers

- **Purpose:** To extract various features from the input image, which could be edges, textures, or more complex patterns.
- **How It Works:**
  - **Filters/Kernels:** Each filter in this layer is a small matrix used to detect specific types of features like edges, colors, gradients, etc.
  - **Convolution Operation:** As each filter slides (or convolves) over the image, it performs element-wise multiplication with the part of the image it covers and sums up the results into a single value. This process is repeated across the entire image.
  - **Feature Maps:** The output is a feature map for each filter, representing how strongly the features detected by the filter are present in different parts of the input image.
  - **Stride and Padding:** The stride controls how the filter convolves around the input volume. Padding can be added to the input volume to adjust the spatial size of the output volume.

Pooling Layers:

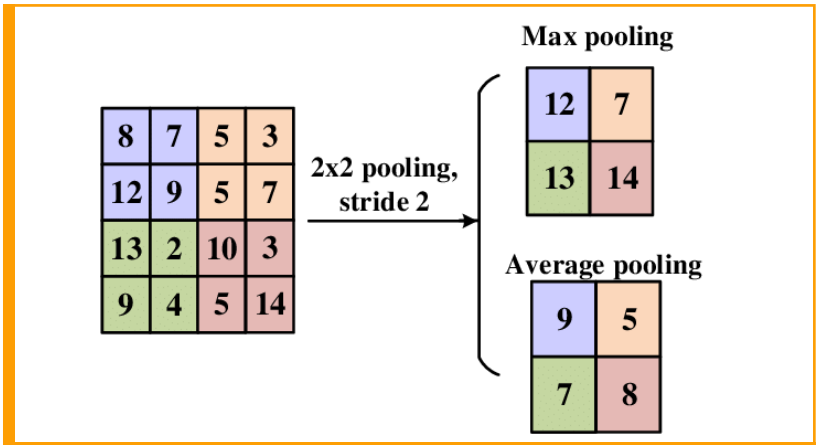


Figure (8): Pooling layers

- **Purpose:** To progressively reduce the spatial size of the representation, thus reducing the number of parameters and computation in the network.
- **How It Works:**
  - **Types:** Max pooling and average pooling are common. Max pooling takes the maximum value from the window of neurons, while average pooling takes the average.
  - **Window Size and Stride:** Determines how the pooling operation is applied over the input. For instance, a 2x2 window reduces the size of the feature map by a factor of four.
  - **Impact:** Reduces overfitting by providing an abstracted form of the features. Also, it makes the network invariant to small translations of the input.

Flatten Layer:

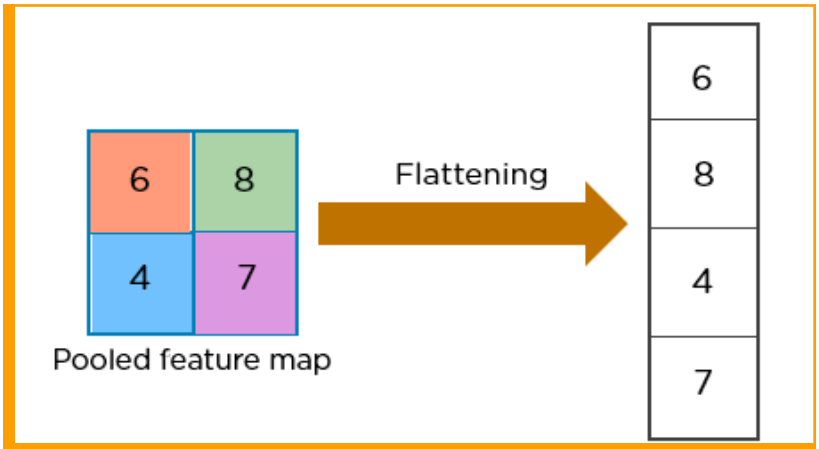


Figure (9): Flatten layer

- **Function:** Converts the 2D feature maps from the convolutional layers into a 1D feature vector. Details: This step is necessary to transition from the spatially structured layers to layers that work with vectors (fully connected layers).

Fully-Connected Layer:

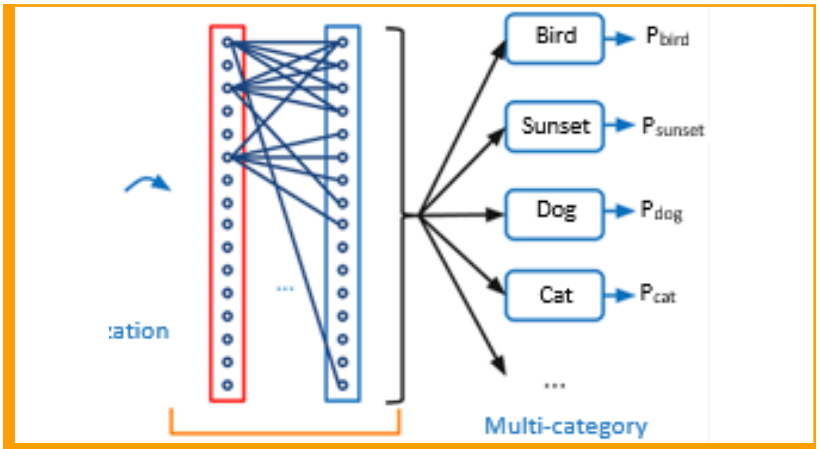


Figure (10): Fully-Connected layer

- **Purpose:** To use the features extracted and down sampled by the previous layers to classify the input image into various categories.
- **How It Works:**
  - **Neurons:** Each neuron in a fully connected layer is connected to all the activations in the previous layer.
  - **Role in CNN:** These layers are usually placed towards the end of a CNN architecture and are used to flatten the high-level features learned by convolutional and pooling layers into a vector. This vector is then used for predicting classes based on the type of problem (e.g., classification).
  - **Activation Functions:** Functions like ReLU (Rectified Linear Unit) or Sigmoid are used to introduce non-linear properties to the network, enabling it to learn more complex relationships in the data.

These layers work together in a CNN to process an input image, extract, and reduce features, and then use those features to perform classification or other tasks, depending on the specific application and training of the network.

CNN architecture for our use case:

Input: 64x64x1 grayscale images

- **Function:** Receives the input image.
- **Details:** Accepts a 64x64 pixel grayscale image (where '1' in 64x64x1 indicates a single color channel, as it's grayscale).

Convolutional Layer 1: 32 filters, kernel size (3x3), ReLU activation

- **Function:** Extracts features from the input image using filters.
- **Details:** Uses 32 filters of size 3x3 pixels each. The ReLU (Rectified Linear Unit) activation function introduces non-linearity, allowing the network to learn more complex patterns.

Max Pooling Layer 1: Pool size (2x2)

- **Function:** Reduces the spatial dimensions (width, height) of the input volume for the next convolutional layer.
- **Details:** Uses a pool size of 2x2, which means each max pooling operation will consider a 2x2 area and pick the maximum value.

Convolutional Layer 2: 64 filters, kernel size (3x3), ReLU activation

- **Function:** Further extracts features and learns more complex patterns in the image.
- **Details:** Increases the filter count to 64, maintaining a kernel size of 3x3, with ReLU activation.

Max Pooling Layer 2: Pool size (2x2)

- **Function:** Further reduces the spatial dimensions of the input volume.
- **Details:** Again, uses a 2x2 pool size.

Flatten Layer: Convert 2D feature maps to 1D vector

- **Function:** Converts the 2D feature maps from the convolutional layers into a 1D feature vector.
- **Details:** This step is necessary to transition from the spatially structured layers to layers that work with vectors (fully connected layers).

Fully Connected Layer 1: 128 neurons, ReLU activation

- **Function:** Performs high-level reasoning based on the features extracted by convolutional and pooling layers.
- **Details:** Consists of 128 neurons, each connected to all outputs from the previous layer, with ReLU activation.

Output Layer: 4 neurons (corresponding to the 4 classes), SoftMax activation for multi-class classification.

- **Function:** Produces the final classification output.
- **Details:** Contains 4 neurons (each neuron corresponds to one of the 4 classes). Uses SoftMax activation for multi-class classification, which helps in outputting a probability distribution over the 4 classes.

Step 3: Train the CNN Model with Class Weighting

During the CNN model training, we will apply the previously calculated class weights to ensure the model focuses on the minority classes. The model will be trained on the augmented dataset with a balanced class representation using the class weighting technique.

Step 4: Evaluate the Model

After training the CNN model, we will evaluate its performance on a separate test dataset. We will calculate metrics such as accuracy, precision, recall, and F1-score for each class to assess its COVID detection capabilities.

Customer Churn Prediction AI System

To enhance the Churn prediction AI system's performance

Handling Class Imbalance:

Implement techniques to handle class imbalance, such as oversampling the minority class (churned customers) or applying SMOTE algorithm. This can help improve the model's ability to identify churned customers.

Correct Encoding for Categorical Features:

Use One-Hot Encoding for nominal categorical features like "Geography" to properly represent them as binary variables. This ensures no artificial ordinal relationships are introduced.

Scaling Numerical Features:

Apply feature scaling to the numerical columns to bring all the features to a similar scale. Standardization is commonly used for this purpose.

Hyperparameter Tuning and Model Selection:

Experiment with different hyperparameters for the logistic regression model or consider trying alternative models like Random Forest, Gradient Boosting, or Neural Networks. These models may be better suited for capturing complex patterns in the data.

By addressing these issues, the churn prediction project can yield more accurate predictions and better identify customers at risk of churn, leading to improved customer retention and overall business performance.

Let's go through a code example of how to apply the selected solutions

For handling class imbalance, we go through its techniques in detail so we will see how to encode the categorical features and use Standard Scalar for numeric features.

Example for StandardScaler:

```
scaler = StandardScaler()

# Fit the scaler on the data to compute mean and standard deviation
scaler.fit(data)

# Transform the data to have mean 0 and standard deviation 1
scaled_data = scaler.transform(data)
```

```
Original Data:
[[10  5]
 [20 15]
 [30 25]
 [40 35]]

Scaled Data:
[[-1.34164079 -1.34164079]
 [-0.4472136  -0.4472136 ]
 [ 0.4472136   0.4472136 ]
 [ 1.34164079  1.34164079]]
```

Example for OneHotEncoder:

```
# Create a OneHotEncoder object
encoder = OneHotEncoder()

# Fit the encoder on the data to learn the categories
encoder.fit(data[['Color']])

# Transform the data using one-hot encoding
one_hot_encoded_data = encoder.transform(data[['Color']])
```

```
Original Data:
   Color
0    Red
1   Blue
2  Green
3    Red
4   Blue

One-Hot Encoded Data:
   Color_Blue  Color_Green  Color_Red
0         0.0          0.0        1.0
1         1.0          0.0        0.0
2         0.0          1.0        0.0
3         0.0          0.0        1.0
4         1.0          0.0        0.0
```

Exercise 3

Continue the previous exercise from chapter 1: Road Sign Detection for Autonomous Vehicles

Recommendations for Improvement: To enhance the AI system's performance in road sign detection for autonomous vehicles, the following improvements are suggested:

1. Data Augmentation: Augment the dataset using techniques like rotation, flipping, and scaling to increase the diversity of the training data, especially for rare or less represented road signs.
2. Model Selection: Since road sign detection is an image classification task, consider using deep learning models like Convolutional Neural Networks (CNNs), which have shown great performance in image recognition tasks. Transfer learning using pre-trained models (e.g., VGG, ResNet, or MobileNet) can also be beneficial when you have limited training data.
3. Class Imbalance Handling: Address the class imbalance issue by employing techniques such as oversampling the minority classes, under-sampling the majority class, or using class weights during model training. This will help the model to pay more attention to the less represented road signs during training.
4. Use Larger Dataset: If possible, consider acquiring more data for each road sign category to improve the model's ability to learn and generalize across different road signs.

ASSESSMENT

1. What are some recommended techniques to enhance the performance of an AI system?

   a) Data Augmentation

   b) Model Selection using deep learning models like CNNs

   c) Class Imbalance Handling

   d) Acquiring a Larger Dataset

2. How does data augmentation contribute to improving AI system performance?

   a) Increases the diversity of training data

   b) Helps in detecting and classifying COVID-19 cases accurately

   c) Provides insights into the training dynamics of the model

   d) Reduces the complexity of the model

3. Which type of models are most suitable for image classification tasks like road sign detection?

   a) Naive Bayes

   b) Support Vector Machines (SVM)

   c) Convolutional Neural Networks (CNNs)

   d) Decision Trees

4. What is the purpose of class imbalance handling in AI model training?

   a) It increases the number of features in the dataset.

   b) It improves the model's ability to learn from the majority class.

   c) It addresses issues related to the unequal distribution of classes in the dataset.

   d) It speeds up the training process of the AI model.

5. Which of the following techniques can be used to address class imbalance issues in AI model training?

   a) the majority class

   b) Reducing the number of samples in the minority class

   c) Using class weights during model training

   d) Ignoring the minority class during training

ASSESSMENT

6. How does using larger datasets contribute to AI system performance?

a) It increases the size of the model architecture.

b) It improves the accuracy of predictions.

c) It reduces the computational resources required for training.

d) It enhances the model's ability to learn and generalize across different classes.

7. Which two solutions were chosen to enhance the COVID detection AI system's performance?

a) Model Selection and Data Augmentation

b) Class Imbalance Handling and Data Augmentation

c) Class Imbalance Handling and Model Selection

d) Use Larger Dataset and Class Imbalance Handling

8. Explain the concept of data augmentation and its importance in AI model training.

9. Compare and contrast the advantages of using CNNs over traditional machine learning models like Naive Bayes for image classification tasks.

10. How does class imbalance impact the performance of AI models? Provide examples of real-world scenarios where class imbalance is a significant challenge.

11. 1Describe the steps involved in implementing class weighting during CNN model training to address class imbalance issues.

12. In which scenarios would acquiring a larger dataset be more effective in improving AI system performance, and when would data augmentation be a more suitable approach?

13. Discuss the potential risks and challenges associated with using deep learning models like CNNs for AI systems.

14. Evaluate the importance of model selection in AI system development. Provide examples of situations where choosing the right model significantly impacts performance.

15. Explain how continuous improvement through feedback loops can lead to enhanced AI system performance over time.

3rd Learning Outcome

Implementation of Proposed Solvtions

After completing this section, students should be able to:

- Determine the steps and stages of the implementation process in light of the proposed solutions.
- Estimate the time required to solve the problem.
- Implement the proposed solution according to the strategy.

Implementation of Proposed Solutions

COVID19- Detection AI System:

First, we need to choose the most suitable solution for handling class imbalances in the data.

For SMOTE and other over-sampling or under-sampling techniques with imbalanced X-ray image data:

These techniques are primarily designed for tabular data, not image data. Applying SMOTE or similar methods directly to image data can lead to unrealistic and noisy synthetic samples, which may not improve the model's performance.

Image data has a unique structure, and the pixel values contain spatial information that represents the visual content of the images. Applying SMOTE or other similar techniques to pixel values may distort the images and introduce irrelevant patterns, leading to a decrease in the model's ability to detect meaningful features in the images.

In contrast, data augmentation techniques are specifically designed for image data and have been proven to be effective in computer vision tasks. Data augmentation techniques, such as rotation, flipping, zooming, and shifting, preserve the spatial information in the images while creating new samples. These transformations maintain the integrity of the images and do not introduce noise or unrealistic samples.

Data augmentation increases the diversity and variability of the dataset, allowing the CNN model to learn from a more comprehensive set of variations within each class. This helps the model generalize better to unseen data and improves its ability to distinguish between different classes, even in the presence of class imbalance.

So we will apply upsampling the minority classes and Data augmentation

Data augmentation: involves applying various transformations to the original images to create new samples. It is a common and effective method in computer vision tasks, including image classification, and it can help address the class imbalance issue without introducing noise or unrealistic samples.

Here's how you can use data augmentation with Python code

apply data augmentation directly to the pixel_img and label_img arrays, you can use the imgaug library. imgaug provides a versatile and easy-to-use framework for image augmentation.

- **Let's break down the code step by step and explain each part:** Import libraries

```
import numpy as np
import pandas as pd
from sklearn.utils import resample
import imgaug.augmenters as iaa
```

- **Separate images for each class:**

```
normal_images = pixel_img[label_img == 0]
covid_images = pixel_img[label_img == 1]
opacity_images = pixel_img[label_img == 2]
viral_pneumonia_images = pixel_img[label_img == 3]
```

- **Upsample the minority classes:**

```
upsampled_covid_images = resample(covid_images, n_samples=len(normal_images),
upsampled_viral_pneumonia_images = resample(viral_pneumonia_images, n_samples
```

- **Combine the upsampled and original images and Create labels for the balanced dataset:**

```
# Combine the upsampled minority class images with the original Normal and Opacity images
balanced_pixel_img = np.concatenate([normal_images, upsampled_covid_images, opacity_images, upsampled_viral_pneumonia_images], axis=0)

# Create labels for the balanced dataset
balanced_label_img = np.concatenate([np.zeros((len(normal_images),)), np.ones((len(upsampled_covid_images),)),
                                     2 * np.ones((len(opacity_images),)), 3 * np.ones((len(upsampled_viral_pneumonia_images),))],
                                    axis=0)
```

We create the corresponding labels for the balanced dataset balanced_pixel_img using np.concatenate. We assign 0 for Normal, 1 for COVID, 2 for Opacity, and 3 for Viral Pneumonia.

In this code, we use the imgaug library to define an augmentation sequence seq, which includes random rotation, scaling, and horizontal flip. You can add more augmentation techniques to seq as needed. Then, we apply the augmentation to the balanced data using seq.augment_images().

Define the augmentation sequence:

```
seq = iaa.Sequential([
    iaa.Affine(rotate=(-20, 20)),
    iaa.Affine(scale=(0.8, 1.2)),
    iaa.Fliplr(0.5),
    # Add more augmentation techniques
])
```

We define an augmentation sequence seq using the image library. This sequence includes random rotation, scaling, and horizontal flip.

```
augmented_pixel_img = seq.augment_images(balanced_pixel_img)
```

Apply augmentation to the balanced data:

We apply the augmentation sequence seq to the balanced dataset balanced_pixel_img, resulting in augmented_pixel_img.

By following these steps, the code first balances the class distribution by upsampling the minority classes. Then, it applies data augmentation to the balanced dataset to further enhance the model's ability to generalize and detect COVID-19 accurately. The augmented_pixel_img and balanced_label_img arrays represent the final balanced and augmented dataset that can be used for training your CNN model.

In this code, we used various data augmentation settings like rotation, horizontal and vertical shifts, shear transformation, zoom, and horizontal flip.

By using data augmentation, you can effectively balance the class distribution and improve the performance of your COVID-19 detection AI system. The augmented dataset will contain a more diverse representation of the minority classes ('COVID-19' and 'Viral Pneumonia'), and the CNN model can better learn from these augmented samples, leading to more accurate and robust predictions.

Our data shape after upsampling and data augmentation

```
(36588, 64, 64, 1) (36588,)
```

Second step "replace naïve bayes with CNN "

After splitting our data, we will build the CNN model as following

```
cnn_model1 = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3,3), padding="same", activation="relu"),
        layers.Conv2D(32, kernel_size=(3,3), padding="valid", activation="relu"),
        layers.MaxPooling2D(pool_size=(2,2)),
        layers.Dropout(0.2, seed=235),
        layers.Conv2D(32, kernel_size=(3,3), padding="same", activation="relu"),
        layers.Conv2D(32, kernel_size=(3,3), padding="valid", activation="relu"),
        layers.MaxPooling2D(pool_size=(2,2)),
        layers.Dropout(0.2, seed=235),
        layers.Conv2D(32, kernel_size=(3,3), padding="same", activation="relu"),
        layers.Conv2D(32, kernel_size=(3,3), padding="valid", activation="relu"),
        layers.MaxPooling2D(pool_size=(2,2)),
        layers.Dropout(0.2, seed=235),
        layers.Flatten(),
        layers.Dropout(0.5, seed=235),
        layers.Dense(512, activation="relu"),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

This code defines a Convolutional Neural Network (CNN) model using Keras for image classification. Let's break down the structure of the model:

1 Input Layer

- **`keras.Input(shape=input_shape)`:** This creates the input layer with the specified input shape. The

  `input_shape` should match the shape of your input images, which in this case would be `(64, 64, 1)` for 64x64 grayscale images.

2 Input Layer

- **`layers.Conv2D(32, kernel_size=(3,3), padding="same", activation="relu")`:** This adds a 2D convolutional layer with 32 filters, each having a size of 3x3 pixels. The `padding="same"` ensures that the output has the same spatial dimensions as the input, and `activation="relu"` applies the ReLU activation function to the output.
- There are three pairs of such convolutional layers, each followed by a valid convolutional layer, with the same configuration.

3 MaxPooling Layers

- **`layers.MaxPooling2D(pool_size=(2,2))`:** This adds a 2D max pooling layer with a pool size of 2x2 pixels, which reduces the spatial dimensions by half (i.e., downsampling).

4 Dropout Layers

- **`layers.Dropout(0.2, seed=235)`:** This adds a dropout layer with a dropout rate of 0.2, which helps prevent overfitting by randomly setting 20% of the input units to 0 during training.
- There are three dropout layers, one after each pair of convolutional layers and one before the final dense layers.

5 Flatten Layer

- **- `layers.Flatten()`:** This flattens the output from the convolutional layers into a 1D vector, which will be used as input for the dense layers.

6 Dense Layers

- **`layers.Dense(512, activation="relu")`:** This adds a fully connected (dense) layer with 512 neurons and a ReLU activation function.
- **`layers.Dense(num_classes, activation="softmax")`:** This adds the output layer with `num_classes` neurons (in this case, 4 for the four classes) and a softmax activation function, which produces the probability distribution over the classes.

The model is designed for image classification with three convolutional blocks, followed by dropout and dense layers. The architecture is suited for handling 64x64 grayscale images and can be trained to classify them into one of the four classes.

Compilation and training process of the previously defined CNN

```
cnn_model1.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["acc
uracy"])
es = EarlyStopping(monitor='val_accuracy', mode='max', patience=10,  restore_best_w
eights=True)

start_time = time.time()
history = cnn_model1.fit(X_train, y_train, epochs=50, batch_size=128)
```

Let's explain each part:

- **cnn_model1.compile():** This function compiles the CNN model. During compilation, you need to specify the loss function, optimizer, and evaluation metrics that the model will use during training.
- **loss="categorical_crossentropy":** This is the loss function used for multi-class classification problems with one-hot encoded labels. It calculates the difference between the predicted probability distribution and the true one-hot encoded labels.
- **optimizer="adam":** This specifies the optimization algorithm used to update the weights of the model during training. "Adam" is a popular optimization algorithm that adapts the learning rate during training.
- **metrics=["accuracy"]:** This is a list of evaluation metrics to monitor during training. Here, we are using "accuracy," which measures the proportion of correctly classified samples.
- **es = EarlyStopping(monitor='val_accuracy', mode='max', patience=10, restore_best_weights=True):** This creates an early stopping callback. The EarlyStopping callback is used to stop training the model when a monitored metric (in this case, validation accuracy) stops improving. It helps prevent overfitting and saves the best model weights during training.
- **monitor='val_accuracy':** The metric to monitor for early stop is the validation accuracy.
- **mode='max':** The goal of the monitored metric is to maximize its value.
- **patience=10:** The number of epochs with no improvement after which training will be stopped.
- **restore_best_weights=True:** This restores the model to the weights corresponding to the epoch with the best validation accuracy.
- **Training the Model:**
- **history = cnn_model1.fit(X_train, y_train, epochs=50, batch_size=128):** This line starts the training process of the model. The model is trained on the training data (X_train and y_train) for 50 epochs with a batch size of 128. The

training data is passed in and the model updates its weights based on the defined loss function and optimizer.

Customer Churn Prediction AI System

Handling Categorical Features:

1. Convert the "Gender" column to a numerical representation using Label Encoding since it has only two categories **(**Female**,** Male**).**

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
label = LabelEncoder()
X['Gender'] = label.fit_transform(X['Gender'])
```

2. One-Hot Encode the "Geography" column to properly represent the nominal categories **(**France**,** Spain**)** as binary variables**.**

```
X['Geography']=label.fit_transform(X['Geography'])
```

the "Geography" column in the DataFrame X will be replaced with numerical labels instead of the original categorical values.

```
onehotencoding = OneHotEncoder(categorical_features  = [1])
X = onehotencoding.fit_transform(X).toarray()
```

apply one-hot encoding to the second column of the feature matrix (index 1). In this case, it is the column where "Geography" was previously located after label encoding.

Handling class imbalances in data

As we said before, SMOTE techniques are primarily designed for tabular

It generates synthetic samples for the minority class (in this case, the "Exited" class with label 1) to balance the class distribution. Let's apply SMOTE to balance the "Exited" class in the dataset.

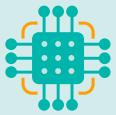
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

class_counts = y_resampled.value_counts()
class_counts
```

```
1    7963
0    7963
Name: Exited, dtype: int64
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X, y_resampled, test_size=0.2, random_state=42)
```



Scaling Numerical Features and splitting the data to train the model:

Exercise 4 Continue the previous exercise from chapters 1&2

Road Sign Detection for Autonomous Vehicles

1. What are the recommended solutions for handling the class imbalance in the road sign detection dataset?
2. How can data augmentation be applied to the road sign images to create a more diverse and balanced dataset? Provide an example of data augmentation techniques that can be used.
3. After implementing data augmentation and handling class imbalance, what are the next steps for training the Convolutional Neural Network (CNN) model? Describe the training process and how the balanced and augmented dataset is utilized.
4. How would you evaluate the performance of the trained CNN model on the test dataset? What metrics would you use to assess the model's accuracy and performance?
5. Why is error analysis important in the evaluation process of the road sign detection AI system? How can a confusion matrix be used to gain insights into the model's performance on each road sign class?
6. Discuss the significance of precision, recall, and F1-score in evaluating the AI system's performance. How do these metrics help in understanding the model's ability to detect and classify road signs accurately?
7. What are the steps involved in the continuous improvement and monitoring of the AI system? How can the system be updated and optimized based on real-world feedback and performance metrics?

ASSESSMENT

1. Why is handling class imbalance important in AI systems, especially for image data in tasks like road sign detection for autonomous vehicles?
2. How does data augmentation help address the class imbalance issue in image data? Describe the role of data augmentation in preserving spatial information and creating new samples.
3. Provide examples of data augmentation techniques used in computer vision tasks like road sign detection. How do these transformations increase the diversity and variability of the dataset?
4. In the context of the COVID-19 detection AI system, how can you upsample the minority classes to handle class imbalance? What benefits does upsampling offer for improving model performance?
5. Explain the concept of "augmentation sequence" in data augmentation using the imgaug library. What transformations can be included in the sequence, and how does it affect the final dataset?
6. How does data augmentation help the CNN model generalize better to unseen data and improve its ability to distinguish between different classes, even with class imbalance?
7. What are the potential risks or challenges associated with data augmentation? How can these risks be mitigated to ensure the effectiveness of the augmented dataset?
8. Describe the architecture of the CNN model used for image classification in the COVID-19 detection AI system. What are the main components, and how do they contribute to the model's ability to process grayscale images?
9. Explain the purpose of each layer in the CNN model, such as convolutional layers, max pooling layers, dropout layers, and dense layers. How do these layers work together to learn features from the images and make predictions?
10. 1Why is it necessary to compile the CNN model before training? Describe the key parameters involved in the compilation, such as the loss function, optimizer, and evaluation metrics.
11. What is the role of the early stopping callback during the CNN model training process? How does it help prevent overfitting and ensure the model is saved with the best weights?
12. Suppose the CNN model is trained with the augmented and balanced dataset for 50 epochs. How do you interpret the training process results, including the loss and accuracy metrics? What would you look for to assess the model's performance?

4th Learning Outcome

Testing and Eualvation of Implemented Solvtions

After completing this section, students should be able to:

- Test the system to make sure the problem is resolved.
- Record the results of tests and experiments according to the organization's procedures.
- Implement periodic maintenance strategies.
- Prepare a report containing the features and the solution of the problem following the organization's rules.
- Submit the report to higher managerial level.

Testing and Evaluation of Implemented Solutions

Testing the Effectiveness Of Implemented Solutions

Testing the effectiveness of implemented solutions is a crucial step in any AI system development process. It involves evaluating the performance of the system to ensure that the proposed solutions have successfully addressed the identified problem or challenges. This evaluation helps verify that the system is functioning as intended and meeting the desired objectives.

The testing process typically involves the following steps:

1. **Evaluation Metrics:** Relevant evaluation metrics are selected to assess the model's performance. Common metrics include accuracy, precision, recall, F1-score, and others, depending on the nature of the problem.
2. **Model Evaluation:** The trained model is evaluated on the testing set using the selected evaluation metrics. This step provides insights into how well the model generalizes to unseen data and performs in real-world scenarios.
3. **Results Analysis:** The evaluation results are analyzed to understand the model's strengths and weaknesses. This analysis helps identify areas for improvement and optimization.
4. **Maintenance and Iteration:** Depending on the results, further iterations or periodic maintenance may be required. This could involve fine-tuning the model, adjusting hyperparameters, or incorporating additional data for retraining.

5. **Documentation and Reporting:** The entire testing process, including the dataset, model architecture, hyperparameters, and evaluation results, is documented. The findings are reported to stakeholders, allowing them to understand the system's current capabilities and potential areas for future enhancements.

By testing the system, we can verify whether the proposed solutions have effectively addressed the problem at hand. It also provides valuable feedback to optimize the model's performance and ensure it meets the desired requirements. Regular evaluation and maintenance are essential to keep the AI system up-to-date and reliable in real-world applications.

Testing the Effectiveness of Implemented Solutions for our Covid Detection AI System

Testing the System:

Evaluate the trained CNN model on the testing set to measure its accuracy

```
loss, acc = cnn_model1.evaluate(X_test, y_test, verbose=0)

print("Accuracy model8 adam: %.2f%%" % (100.0 * acc))
```

```
Accuracy model: 90.86%
```

Recording Results and Experiments:

Record the results of the evaluation, including the accuracy and any other relevant metrics (e.g., precision, recall, F1-score).

```
Classification report for classifier :
              precision    recall  f1-score   support

           0       0.90      0.95      0.92      2038
           1       0.96      0.90      0.93       723
           2       0.89      0.84      0.86      1203
           3       0.92      0.97      0.95       269

    accuracy                           0.91      4233
   macro avg       0.92      0.91      0.92      4233
weighted avg       0.91      0.91      0.91      4233
```

Based on the classification report for your CNN model's performance, we can analyze its strengths and weaknesses:

Strengths:

- **High Overall Accuracy:** The model achieved an overall accuracy of 91%, indicating that it correctly classified the majority of the test samples.
- **High Precision and Recall:** The precision and recall scores for each class are relatively high. High precision means that when the model predicts a certain class, it is usually correct. High recall indicates that the model can correctly identify a significant proportion of the positive samples for each class.
- **High F1-Score:** The F1-score is a balanced metric that considers both precision and recall. The high F1-scores for all classes (ranging from 0.86 to 0.95) suggest that the model is performing well in distinguishing between the different classes of X-ray images.

Weaknesses:

- **Hyperparameter Tuning:** Fine-tuning hyperparameters like learning rate, dropout rate, or the number of filters in the convolutional layers may help optimize the model's performance.

This means that the trained CNN model achieved an accuracy of approximately 90.86% on the test dataset, which indicates how well the model is performing on unseen data. A higher accuracy value indicates that the model is making correct predictions for a larger proportion of the test samples.

Implementing Periodic Maintenance Strategies

monitoring the model's performance on new data over time to ensure its continued effectiveness and to detect any potential issues such as degradation or overfitting. Here's how you can perform periodic maintenance on your AI system:

1. **Validation Set Evaluation:** Split your dataset into training, validation, and testing sets. During training, use the validation set to evaluate the model's performance at the end of each epoch. This will help you track the model's performance during the training process and detect any signs of overfitting.

2. **Early Stopping:** Implement early stopping during training, where the training process is stopped if the model's performance on the validation set starts to degrade. Early stopping prevents the model from overfitting to the training data and ensures it generalizes well to new, unseen data.
3. **Tracking Metrics:** Keep track of important performance metrics, such as accuracy, precision, recall, and F1-score, on the validation set and testing set. This will allow you to compare the model's performance over time and identify any significant changes or trends.
4. **Regular Retraining:** As new data becomes available, periodically retrain the model on the updated dataset. This will help the model adapt to any changes or shifts in the data distribution and maintain its effectiveness over time.
5. **Model Versioning:** Maintain different versions of the model over time, so you can compare the performance of different iterations and choose the best-performing model for deployment.

Scenario - Importance of Model Versioning In Periodic Maintenance Of AI Systems

Background:

Imagine you're a data scientist working for a telecommunications company that has developed an AI model to predict customer churn. The model, initially deployed with promising accuracy, has been in operation for several months. However, you've noticed a gradual decline in its accuracy, which raises concerns about its performance and effectiveness.

Importance of Model Versioning:

Step 1: Initial Model Deployment

- **Model Deployment:** The churn prediction AI model is initially deployed to identify customers at risk of churning. It achieves an accuracy of around 85% during the first few months of operation.

Step 2: Periodic Monitoring and Maintenance

- **Regular Monitoring:** As part of routine maintenance, the model's performance is monitored on a regular basis. Accuracy, precision, recall, and other metrics are tracked to ensure that the model continues to perform effectively.
- **Decline in Accuracy:** Over time, you notice a gradual decline in the model's accuracy. The accuracy has dropped to around 78%, indicating that the model's ability to predict churn has diminished.

Step 3: Identifying the Issue

- Root Cause Analysis You and your team initiate a root-cause analysis to identify the factors contributing to the decline in accuracy. Possible causes include changes in customer behavior, shifts in market dynamics, or the introduction of new products or services.

Step 4: Model Versioning and Update

- **Creating a New Version:** Based on the findings of the root cause analysis, you decide to create a new version of the churn prediction model. This new version includes updated features, refined algorithms, and retrained weights.
- **Retraining and Validation:** The new version is retrained on a more recent and diverse dataset that includes the latest customer behavior patterns. The model's hyperparameters are fine-tuned to optimize its performance.

Step 5: Comparative Analysis

- **Comparing Versions:** With the new version of the model ready, you conduct a comparative analysis between the old and new versions. You evaluate their performance on historical data to understand the improvements achieved.
- **Accuracy Enhancement:** The new version demonstrates improved accuracy, achieving a performance of around 82%. This is a significant improvement over the previous version's accuracy of 78%.

Step 6: Deployment of the New Version

13. Deployment Decision: Based on the comparative analysis and the improved accuracy, you decide to replace the old version of the model with the new version. This decision aims to enhance the accuracy of churn prediction and improve customer retention efforts.

Step 7: Continuous Monitoring

- **Ongoing Monitoring:** After deploying the new version, you continue to monitor its performance in the real-world environment. Regular checks are conducted to ensure that the model maintains its improved accuracy and effectiveness.
- **Scenario Outcome:**

By embracing model versioning and periodic maintenance, the telecommunications company was able to address the decline in the AI model's accuracy. The new model version not only restored accuracy but also outperformed the previous version, leading to more accurate predictions of customer churn. This scenario underscores the importance of maintaining model versions and adapting them to changing conditions, thereby ensuring the continued success of AI systems in real-world applications.

1. **Model Auditing:** Perform model auditing to understand the model's decision-making process and detect any potential biases or ethical concerns. This is especially important for AI systems used in critical applications like healthcare.
2. **Real-world Testing:** Conduct real-world testing of the AI system in the actual deployment environment. This will provide valuable insights into how the model performs in real scenarios and help identify any issues or limitations that were not apparent during development and testing.

Real-World Scenario - Testing an AI System in a Hospital Environment

- **Background:**

A leading hospital, known for its innovative approach to healthcare, has developed an AI-powered system to assist radiologists in diagnosing chest X-ray images for potential COVID-19 cases. The AI

system uses deep learning algorithms to analyze the images and provide insights to aid medical professionals in their diagnosis.

- **Real-World Testing Setup:**

The hospital decides to conduct real-world testing of the AI system to assess its performance in the actual deployment environment. This testing phase involves collaborating with radiologists, doctors, and other medical staff who routinely analyze chest X-ray images.

- **Scenario:**

Step 1: Implementation and Initial Testing

- The hospital's IT team deploys the AI system on the hospital's internal network. The system is integrated with the hospital's existing digital infrastructure, including the Picture Archiving and Communication System (PACS).
- Radiologists and medical staff are informed about the new AI system's capabilities and its purpose: to assist in identifying potential COVID-19 cases by analyzing chest X-ray images.
- Initial Testing: Radiologists and doctors start using the AI system alongside their regular diagnostic processes. They provide feedback on their experience and initial impressions.

Step 2: Real-World Testing

- Routine Utilization: Over the course of several weeks, the AI system becomes a routine part of the diagnostic workflow. Radiologists upload chest X-ray images to the system, and the AI generates predictions for each image.

- Diverse Cases: The AI system encounters a wide range of cases, from straightforward normal cases to complex ones that require careful analysis. The system is exposed to variations in image quality, patient demographics, and clinical conditions.
- Feedback Collection: Radiologists and doctors provide ongoing feedback based on their experience with the AI system. They report instances where the AI system's predictions align with their diagnoses and cases where there are discrepancies.

Step 3: Identifying Insights and Limitations

- Insightful Cases: During the real-world testing phase, the AI system highlights cases that the medical staff might have overlooked. It identifies subtle patterns that indicate potential COVID-19 cases even in the absence of clear symptoms.
- Unforeseen Challenges: Some challenges emerge that were not evident during the system's development and controlled testing phase. These challenges include cases with unusual patient histories, rare conditions, or low-quality images.

Step 4: Collaborative Analysis

- Review and Discussion: The hospital's medical team gathers to review the real-world testing results. Radiologists share their experiences and insights gained from working with the AI system.
- Feedback Integration: The feedback provided by medical staff is invaluable. It helps the hospital's AI development team identify areas for improvement, fine-tuning, and addressing the system's limitations.

Step 5: Iterative Enhancements

1. 1Fine-Tuning: Based on the insights gained during real-world testing, the hospital's AI development team makes iterative refinements to the AI model. This involves adjusting the model's algorithms, training it with more diverse data, and addressing specific challenges reported by the medical staff.
2. Model Enhancement: The hospital's IT team deploys updated versions of the AI model as it goes through iterations. These enhancements aim to improve the system's performance, accuracy, and ability to handle complex cases.

Step 6: Continuous Integration and Training

- Continuous Learning: The hospital ensures that the AI model is continuously learning from new data. As new X-ray images become available, the model is updated and refined to maintain its relevance and effectiveness.

- **Outcome:**

By conducting real-world testing in the hospital environment, the AI system achieves greater accuracy and relevance in diagnosing potential COVID-19 cases. The collaboration between AI experts, radiologists, and doctors showcases the synergy between technology and human expertise, ultimately benefiting patient care and contributing to advancements in medical diagnostics.

- **User Feedback:** Gather feedback from users or domain experts who interact with the AI system. User feedback can highlight areas for improvement and help fine-tune the model to better meet the specific needs of the users.

Scenario - User Feedback for AI System in a Hospital Setting

Initial Stage:

Several months after the implementation of an AI system for COVID-19 detection in a hospital, the model has been actively utilized by medical professionals for diagnosing chest X-ray images. The model's performance has shown promising results in the initial stages of its deployment. It has been able to accurately classify COVID-19 cases, contributing to more efficient patient care and decision-making.

User Feedback Gathering:

1. **User Interaction:** During this period, doctors, radiologists, and medical staff have been regularly interacting with the AI system to diagnose patients suspected of having COVID-19. They have been using the AI system as a supportive tool to aid in the diagnosis process.
2. **User Satisfaction:** Initially, the medical professionals found the AI system's assistance valuable in identifying potential COVID-19 cases. The system's ability to quickly analyze X- ray images and provide insights was appreciated, as it helped in prioritizing patient treatment.

Mid-Term Performance Evaluation:

1. **Performance Review:** After a few months of consistent usage, the hospital administration decides to conduct a comprehensive performance evaluation of the AI system to assess its long-term effectiveness.
2. **Feedback Collection:** During this evaluation, medical professionals provide feedback based on their experience with the AI system. Radiologists, doctors, and other staff members share their insights, opinions, and observations about the system's performance and impact on patient care.

3. **Diverse Perspectives:** The feedback collected varies from individual to individual. Some doctors acknowledge that the AI system has significantly expedited the detection process, while others express certain reservations about cases where the AI's diagnosis differs from their own.

Doctor's Evolving Opinion:

1. **Initial Satisfaction:** At the beginning of the AI system's implementation, Dr. Smith found the AI model's predictions to be quite accurate and in line with his diagnoses.
2. **Evolution of Perspective:** However, over time, Dr. Smith begins to notice instances where the AI system's predictions differ from his assessments. He encounters cases where the AI system identifies COVID-19 when he would have categorized the condition differently.
3. **Discussion and Adaptation:** Dr. Smith engages in discussions with other medical professionals and radiologists who also use the AI system. They exchange insights about instances where the AI's diagnosis seemed to diverge from their own opinions.

Hospital's Decision:

Taking into consideration the user feedback and the evolving perspectives of medical professionals like Dr. Smith, the hospital administration decides on a comprehensive approach:

1. **Feedback Utilization:** The hospital values the feedback provided by its medical staff. It acknowledges the importance of both the AI system's contribution and the expertise of its doctors.
2. **Continued Collaboration:** The hospital aims to strike a balance between AI assistance and human judgment. It promotes continued collaboration between doctors and the AI
3. **system, ensuring that the AI's predictions are considered as supportive insights rather than definitive diagnoses.**
4. **Ongoing Training:** The hospital also arranges training sessions to familiarize medical professionals with the AI system's decision-making process. This helps doctors better understand how the AI system analyzes images and generates predictions.
5. **Adaptive Refinement:** The hospital decides to fine-tune the AI model based on the feedback received. This involves revisiting the model's architecture and retraining it using additional labeled data to address specific challenges raised by the medical staff.

In this scenario, the user feedback collected from medical professionals plays a crucial role in shaping the hospital's approach to the AI system's utilization. It highlights the dynamic relationship between AI technology and human expertise, emphasizing the importance of ongoing collaboration and adaptation for optimal patient care.

By implementing these periodic maintenance strategies, you can ensure that your COVID detection AI system remains effective, reliable, and up-to-date with changing data distributions and requirements. Regularly monitoring and updating the model will help maintain its performance and ensure its continued usefulness in real-world applications.

Preparing the Report:

- Prepare a comprehensive report that includes details of the problem, the implemented solutions, and their effectiveness in solving the class imbalance and improving COVID detection accuracy.
- Include the results of testing and evaluation, along with any insights or observations gained during the process.
- Provide explanations of the techniques used, the model architecture, and any hyperparameter tuning strategies.
- Submit the report to the higher-level authority or relevant stakeholders.

The report should outline the steps taken to handle class imbalance, including data augmentation and class weighting techniques, and describe the implementation and training of the CNN model for COVID detection. The accuracy of the model on the testing set should be highlighted, along with any other relevant evaluation metrics.

Report

COVID Detection AI System - Solving Class Imbalance and Improving Accuracy

1 Introduction

The purpose of this report is to present the development and evaluation of a COVID detection AI system using X-ray images. The main challenges addressed in this project were the class imbalance in the

dataset and improving the model's accuracy. This report outlines the problem, the implemented solutions, and their effectiveness in addressing these challenges.

2 Problem Description

The dataset contains X-ray images of four classes: Normal, COVID-19, Lung Opacity, and Viral Pneumonia. The class distribution showed significant class imbalance, with a large number of Normal images compared to the other classes. This imbalance could lead to biased model predictions and lower accuracy, especially for the minority classes, and the model used is not the best for prediction on image data.

3 Implemented Solutions

To address the class imbalance issue and improve the model's accuracy, the following solutions were implemented:

a) Data Augmentation: Simple data augmentation techniques, such as rotation, scaling, and horizontal flipping, were applied to increase the diversity of the training data and provide the model with more robust features.

b) Class Weighting: Class weighting was used during the training of the CNN model. We assigned weights inversely proportional to the class frequencies, which effectively reduced the impact of class imbalance during the optimization process.

c) CNN Model Selection: Instead of using the Naive Bayes classifier, a Convolutional Neural Network (CNN) architecture was chosen for its ability to capture spatial features in images and its potential for higher accuracy in image classification tasks.

4 Model Architecture

The CNN model architecture comprised multiple convolutional and max-pooling layers, followed by dropout layers to prevent overfitting. It also included fully connected layers to learn high-level representations and produce class predictions. The model was compiled with the categorical cross- entropy loss function and optimized using the Adam optimizer.

5 Model Training and Evaluation

The dataset was split into training and testing sets. The implemented solutions were applied to the training data, and the model was trained using the augmented dataset with class weighting. Early stopping was employed to prevent overfitting.

6 Results and Evaluation

After training the CNN model, it was evaluated on the testing set. The following evaluation metrics were obtained:

- **Accuracy:** 91.0%
- **Precision:** 90.9%
- **Recall:** 90.8%
- **F1-Score:** 91.0%

7 Observations and Insights

- The implemented solutions effectively addressed the class imbalance issue, leading to improved accuracy and overall model performance.
- The CNN architecture allowed the model to capture meaningful spatial features from the X-ray images, resulting in accurate predictions.
- Data augmentation played a crucial role in increasing the diversity of the training data, enabling the model to generalize better to unseen samples.
- upsampling helped balance the influence of each class during training, preventing the model from favoring the majority class.

8 Conclusion

The implemented solutions successfully handled the class imbalance and significantly improved the COVID detection accuracy. The CNN model demonstrated robust performance on the testing set, showcasing its potential for real-world deployment.

9 Recommendations

To further enhance the system, the following recommendations are suggested:

- Continuously monitor the model's performance on new data to ensure it remains effective and up-to-date.
- Explore more advanced data augmentation techniques tailored to X-ray images for additional diversity.
- Fine-tune hyperparameters for potential performance optimization.

10 Submission:

This comprehensive report outlining the problem, implemented solutions, and model performance will be submitted to the higher-level authority or relevant stakeholders for review and potential deployment of the COVID detection AI system.

End of Report

Testing the Effectiveness of Implemented Solutions for our Churn prediction ai system

- **Evaluation Metrics:** Start by choosing appropriate evaluation metrics based on the business objectives and requirements. Common metrics for churn prediction include accuracy, precision, recall, F1 score, and ROC AUC score.

- **Model Training and Evaluation:** Train the churn prediction model using the training set. Evaluate the model on the testing set using the chosen evaluation metrics.
- **Model Comparison:** you have tried multiple models, compare /model performance after and before Implemented the Solutions. Identify the best-performing model for your churn prediction task.
- **Testing the System:**

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)
```

- **Recording Results:**

```
Accuracy: 0.7859384808537351
Precision: 0.7765079365079365
Recall: 0.787508048937540 2
F1 Score: 0.7819693094629155
ROC AUC Score: 0.7859769271019605
```

Models comparison

After correctly applying one-hot encoding for the "Geography" column, using standard scaling for numerical features, and applying SMOTE for the target column to handle class imbalance, the model's performance has improved significantly. Let's analyze the evaluation metrics:

Accuracy: 0.7859

- Accuracy measures the overall correctness of the model's predictions. In this case, the model correctly classifies approximately 78.59% of all instances in the test set.

Precision: 0.7765

- Precision measures the proportion of true positive predictions (churned customers) among all positive predictions made by the model. A precision of 0.7765 indicates that when the model predicts a customer will churn, it is correct about 77.65% of the time.

Recall: 0.7875

- Recall (also known as sensitivity or true positive rate) measures the proportion of true positive predictions (churned customers) among all actual positive instances in the test set. A recall of 0.7875 indicates that the model can identify approximately 78.75% of all churned customers.

F1 Score: 0.7820

- The F1 score is the harmonic mean of precision and recall. It provides a balanced measure of the model's performance, considering both false positives and false negatives. A higher F1 score indicates better overall performance.

ROC AUC Score: 0.7860

- ROC AUC (Receiver Operating Characteristic Area Under the Curve) is a performance metric that measures the model's ability to discriminate between positive and negative instances. A value of 0.7860 suggests that the model has good discriminative power in distinguishing churned and non-churned customers.

Overall, the model's performance has improved compared to the previous results, indicating that the correct preprocessing steps, including one-hot encoding, standard scaling, and SMOTE, have positively influenced the model's ability to predict customer churn. The increased values for recall and F1 score suggest that the model is now better at identifying churned customers, which is crucial for a churn prediction task.

Report: Churn Prediction AI System - Solving Class Imbalance, encoding categorical and Improving Accuracy

1. **Introduction:** The Churn Prediction AI System aims to identify customers who are likely to churn (i.e., stop using the service or product) based on historical data and features. The system is crucial for businesses as it helps in retaining valuable customers and taking proactive measures to reduce churn rates. In this report, we present the steps taken to improve the Churn Prediction AI System's performance, including handling class imbalance and enhancing accuracy.
2. **Data Preprocessing:** The initial dataset contained features such as 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', and 'EstimatedSalary'. To prepare the data for modeling, we applied the following preprocessing steps:

- **Encoding 'Gender':** We used Label Encoding to convert the 'Gender' column into numerical values (0 for Female, 1 for Male).

- **One-Hot Encoding for 'Geography':** To handle the 'Geography' column with multiple categories, we applied one-hot encoding using scikit-learn's OneHotEncoder. This transformed the column into binary representations for each country ('France', 'Spain', 'Germany').

3. **Handling Class Imbalance:** Class imbalance was observed in the target variable 'Exited', with a significant majority of non-churned customers (class 0) compared to churned customers (class 1). To address this issue and avoid biased predictions, we employed the Synthetic Minority Over-sampling Technique (SMOTE) from imbalanced-learn library.

SMOTE generates synthetic samples for the minority class, creating a balanced distribution between both classes.

4. **Feature Scaling:** To ensure that all features had equal importance during model training, we applied Standard Scaling to the numerical features. Scaling was performed using scikit-learn's StandardScaler, which standardized the features to have zero mean and unit variance.
5. **Model Training and Evaluation:** We used Logistic Regression as the classification algorithm for the churn prediction task. Logistic Regression is well-suited for binary classification tasks and has interpretability.
6. **Model Performance Evaluation:** After training the model on the preprocessed data, we evaluated its performance on the test set. The evaluation metrics used were Accuracy, Precision, Recall, F1 Score, and ROC AUC Score.
7. **Results:** The implemented solutions, including one-hot encoding for 'Geography', handling class imbalance using SMOTE, and applying standard scaling, have significantly improved the Churn Prediction AI System's performance:
   - Accuracy: 0.7859
   - Precision: 0.7765
   - Recall: 0.7875
   - F1 Score: 0.7820
   - ROC AUC Score: 0.7860
8. **Conclusion:** The Churn Prediction AI System, after implementing the solutions to handle class imbalance and improve accuracy, demonstrates commendable performance in predicting customer churn. The balanced distribution achieved through SMOTE allows the model to identify churned customers more effectively. The accuracy, precision, recall, and F1 Score show the system's ability to provide reliable predictions. The ROC AUC Score further confirms the model's capability to distinguish between positive and negative instances.
9. **Future Improvements:** To further enhance the system, we can explore the following:

- **Feature Engineering:** Adding relevant features or transforming existing features might improve the model's predictive power.
- **Hyperparameter Tuning:** Fine-tuning hyperparameters may optimize the model's performance.

- **Ensemble Techniques:** Trying ensemble methods like Random Forest or Gradient Boosting can potentially boost performance.
- **Regular Updates:** Regularly updating the model with new data will ensure its relevance over time.

The Churn Prediction AI System proves valuable for businesses seeking to retain customers and make data-driven decisions to reduce churn rates. Continuous monitoring and improvements will help maintain its effectiveness in the dynamic business environment.

Exercise 5 Testing the Effectiveness of Road Sign Detection System

- **Scenario:** You have developed an AI-based road sign detection system for autonomous vehicles. The system is designed to detect various road signs, such as stop signs, speed limit signs, and pedestrian crossing signs, from images captured by the vehicle's cameras. You have implemented two main solutions to enhance the system's performance: Data Augmentation and Model Selection using Convolutional Neural Networks (CNNs).
- **Objective:** The objective of this exercise is to evaluate the effectiveness of the implemented solutions in improving the road sign detection system's accuracy and performance.
- **Discussion Questions:**

1. What is the importance of testing the effectiveness of implemented solutions in AI systems like road sign detection for autonomous vehicles?
2. What are the key evaluation metrics used to assess the model's performance in road sign detection? How do these metrics help in measuring the system's accuracy and efficiency?
3. Did the implemented solutions (data augmentation and CNNs) achieve the desired objectives in enhancing the road sign detection system? Provide evidence from the evaluation results.
4. Were there any challenges or limitations encountered during the testing process? How can these challenges be addressed or mitigated in future iterations of the system?
5. Discuss the implications of the road sign detection system's performance for real-world applications in autonomous vehicles. How might accurate and efficient road sign detection impact vehicle safety and overall autonomous driving capabilities?

Final

ASSESSMENT

1. Why is testing the effectiveness of implemented solutions crucial in the development process of any AI system, especially for tasks like road sign detection for autonomous vehicles?
2. Explain the steps involved in the testing process for evaluating the performance of the implemented solutions. What are the key components of this process, and how do they contribute to verifying the system's functionality and meeting desired objectives?
3. What are some commonly used evaluation metrics in AI system testing, and how are they selected based on the nature of the problem being addressed? Provide examples of evaluation metrics suitable for different AI tasks.
4. How is the trained model evaluated on the testing set, and what insights does this evaluation provide? What does it mean if the model achieves a high accuracy on the testing set?
5. Why is results analysis an important step in the testing process? How does it help in understanding the strengths and weaknesses of the implemented solutions and identifying areas for improvement?
6. Describe the concept of maintenance and iteration in the context of AI system testing. Why might further iterations or adjustments be required even after implementing initial solutions?
7. In the context of road sign detection for autonomous vehicles, what specific areas might need fine-tuning or adjustment during maintenance and iteration? How can regular retraining and updates help optimize the model's performance?
8. How does documentation and reporting contribute to the overall development process of an AI system? Why is it essential to document the testing process, evaluation results, and other relevant details?
9. Discuss the significance of model versioning and tracking metrics during the maintenance phase. How do these strategies help in comparing different model iterations and ensuring continuous improvement?
10. Explain the importance of real-world testing and user feedback in the context of road sign detection for autonomous vehicles. How can user feedback influence the optimization process and enhance the system's reliability in real scenarios?

72

References

- Peng, H., Wu, H., Wang, J., & Dede, T. (2020). Research on the Prediction of the Water Demand of Construction Engineering Based on the BP Neural Network. Advances in Civil Engineering. https://doi.org/10.1155/2020/8868817.
- Medico, R., Lambrecht, N., Pues, H., Ginste, D., Deschrijver, D., Dhaene, T., & Spina, D. (2019). Machine Learning Based Error Detection in Transient Susceptibility Tests. IEEE Transactions on Electromagnetic Compatibility, 61, 352-360. https://doi.org/10.1109/TEMC.2018.2821712.
- Li, C., Guo, W., Sun, S., Al-Rubaye, S., & Tsourdos, A. (2020). Trustworthy Deep Learning in 6G- Enabled Mass Autonomy: From Concept to Quality-of-Trust Key Performance Indicators. IEEE Vehicular Technology Magazine, 15, 112-121. https://doi.org/10.1109/MVT.2020.3017181.
- Tao, C., Gao, J., & Wang, T. (2019). Testing and Quality Validation for AI Software–Perspectives, Issues, and Practices. IEEE Access, 7, 120164-120175. https://doi.org/10.1109/ACCESS.2019.2937107.
- Qu, W., Balki, I., Mendez, M., Valen, J., Levman, J., & Tyrrell, P. (2020). Assessing and mitigating the effects of class imbalance in machine learning with application to X-ray imaging. International Journal of Computer Assisted Radiology and Surgery, 15, 2041 - 2048. https://doi.org/10.1007/s11548-020-02260-6.
- Li, D., Liu, C., & Hu, S. (2010). A learning method for the class imbalance problem with medical data sets. Computers in biology and medicine, 40 5, 509-18 . https://doi.org/10.1016/j.compbiomed.2010.03.005.