ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr/

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr/

Master of Science Thesis

Designing a Scalable and Reproducible Machine Learning Workflow

Student: Polychronou, Ioanna
Registration Number: mscaidl-0031

MSc Thesis Supervisor

Nikolaou, Grigorios
Lecturer

ATHENS-EGALEO, FEBRUARY 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr/

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr/

Μεταπτυχιακή Διπλωματική Εργασία

Σχεδιασμός μιας επεκτάσιμης και αναπαραγώγιμης ροής εργασιών μηχανικής μάθησης

Φοιτητής: Πολυχρόνου Ιωάννα
ΑΜ: mscaidl-0031

Επιβλέπων/ουσα Καθηγητής/τρια
Νικολάου Γρηγόριος
Λέκτορας Τμήματος ΜΒΣΠ

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΦΕΒΡΟΥΑΡΙΟΣ 2024

This MSc Thesis has been accepted, evaluated and graded by the following committee:

Supervisor	Member	Member
GRIGORIOS NIKOLAOU Digitally signed by GRIGORIOS NIKOLAOU Date: 2024.03.20 20:42:48 +02'00'	Digitally signed by Charalampos Patrikakis Date: 2024.03.20 16:31:37 +02'00'	Michail Feidakis Digitally signed by Michail Feidakis Date: 2024.03.16 08:43:03 +02'00'
Nikolaou, Grigorios	Patrikakis, Charalampos	Feidakis, Michalis
Lecturer	Professor	Laboratory Teaching Staff
Industrial Design & Production Engineering	Electrical & Electronic Engineering	Electrical & Electronic Engineering
University of West Attica	University of West Attica	University of West Attica

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/η κάτωθι υπογεγραμμένος/η Ιωάννα Πολυχρόνου του Κωνσταντίνου, με αριθμό μητρώου AIDL-0031 μεταπτυχιακός/ή φοιτητής/ήτρια του ΔΠΜΣ «Τεχνητή Νοημοσύνη και Βαθιά Μάθηση» του Τμήματος Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών και του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής, της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Η εργασία δεν έχει κατατεθεί στο πλαίσιο των απαιτήσεων για τη λήψη άλλου τίτλου σπουδών ή επαγγελματικής πιστοποίησης πλην του παρόντος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.
Επιθυμώ την απαγόρευση πρόσβασης στο πλήρες κείμενο της εργασίας μου μέχρι 23/02/2024 και έπειτα από αίτησή μου στη Βιβλιοθήκη και έγκριση του επιβλέποντος/ουσας καθηγητή/ήτριας.»
Ο/Η Δηλών/ούσα

Ιωάννα Πολυχρόνου

Declaration of the author of this MSc thesis

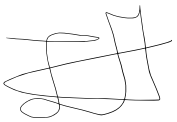
I, Ioanna Polychronou, Konstantinos with the following student registration number: AIDL-0031, postgraduate student of the MSc programme in "Artificial Intelligence and Deep Learning", which is organized by the Department of Electrical and Electronic Engineering and the Department of Industrial Design and Production Engineering of the Faculty of Engineering of the University of West Attica, hereby declare that:

I am the author of this MSc thesis and any help I may have received is clearly mentioned in the thesis. Additionally, all the sources I have used (e.g., to extract data, ideas, words or phrases) are cited with full reference to the corresponding authors, the publishing house or the journal; this also applies to the Internet sources that I have used. I also confirm that I have personally written this thesis and the intellectual property rights belong to myself and to the University of West Attica. This work has not been submitted for any other degree or professional qualification except as specified in it.

Any violations of my academic responsibilities, as stated above, constitutes substantial reason for the cancellation of the conferred MSc degree.

I wish to deny access to the full text of my MSc thesis until 23/02/2024, following my application to the Library of UNIWA and the approval from my supervisor.

The author
Ioanna Polychronou



(Signature)

Acknowledgments

I would like to express my deepest gratitude to my thesis supervisor, Grigorios Nikolaou, for his invaluable guidance, unwavering support, and insightful feedback throughout the entirety of this research endeavor. His expertise and commitment have been instrumental in shaping the direction and quality of this thesis.

I extend my sincere appreciation to Orfium for their support and collaborative involvement in this research. I am particularly grateful to my manager, George, whose leadership and mentorship have been instrumental in steering this project towards success. His encouragement and commitment to fostering a conducive working environment have greatly contributed to the positive outcomes of this thesis.

Special thanks go to Iasonas, the MLOps Tech Leader at Orfium, for their technical insights and continuous support. His guidance has been crucial in navigating the intricacies of the subject matter and ensuring the successful execution of the research.

I am indebted to Kostas, the Senior Data Scientist at Orfium, for providing me with the invaluable opportunity to delve into this captivating subject and for their continuous support throughout the entire process. Kostas' expertise and encouragement have played a pivotal role in the success of this thesis.

I am truly grateful to the entire team at Orfium for fostering an environment that encourages collaboration, learning, and innovation. Their collective efforts have significantly enriched my research experience.

Lastly, I would like to thank my family and friends for their unwavering support and understanding during the challenging periods of this academic journey. Their encouragement and belief in my capabilities have been a source of strength and motivation.

This thesis would not have been possible without the collective support, mentorship, and encouragement of these individuals and organizations. I am truly grateful for the opportunity to work on this subject and for the invaluable lessons learned throughout this research process.

Abstract

The advancement of machine learning (ML) applications necessitates the construction of end-to-end experimentation pipelines characterized by scalability, robustness, and reproducibility. This thesis aims to empower Data Scientists and ML Engineers, enabling seamless pipeline execution, free from unexpected obstacles such as downtime, hardware unavailability, OS conflicts, or dependency issues. To realize these objectives, an ideal ML workflow should exhibit robust execution in a highly available environment, revisitable reproducibility, minimal manual intervention through automation, easy extendability, and scalable capabilities to handle larger tasks concurrently. Moreover, facilitating these attributes, a comprehensive toolkit is essential, encompassing Containerization/Virtualization for consistent environment management, Monitoring experiments for provisioning necessary training information, Data/Model Tracking to trace model and data versions throughout the workflow, Scalable Object Storage for secure data and model storage, and a Workflow Engine for workflow automation, scheduling, and monitoring. This thesis explores the challenges inherent in designing scalable and reproducible ML workflows, detailing the utilization of the aforementioned toolkit to overcome these challenges. Furthermore, industrial case studies are presented to underscore the practical benefits of a scalable and reproducible ML workflow.

Keywords

Artificial intelligence, machine learning life cycle, machine learning operations, scalability, reproducibility

Περίληψη

Η πρόοδος των εφαρμογών μηχανικής μάθησης (Machine Learning) απαιτεί την δημιουργία ολοκληρωμένων ροών εργασίας (pipelines) μηχανικής μάθησης για πειραματικούς σκοπούς. Οι ροές αυτές χαρακτηρίζονται από επεκτασιμότητα (scalability), σταθερότητα (robustness) και αναπαραγωγιμότητα (reproducibility). Αυτή η διπλωματική εργασία στοχεύει να ενισχύσει τους Επιστήμονες Δεδομένων και τους Μηχανικούς Μηχανικής Μάθησης, προσφέροντας πληροφορίες για την απρόσκοπτη εκτέλεση ροών εργασίας χωρίς απροσδόκητα εμπόδια όπως διακοπές λειτουργίας, μη διαθεσιμότητα υλικού, εξάρτηση υλοποίησης και λειτουργικού συστήματος. Για να επιτευχθούν αυτοί οι στόχοι, η ιδανική ροή εργασίας θα πρέπει να παρουσιάζει σταθερότητα και ομαλή εκτέλεση σε ένα εξαιρετικά διαθέσιμο περιβάλλον, επαναληψιμότητα, ελάχιστη χειροκίνητη παρέμβαση μέσω αυτοματισμού, εύκολη δυνατότητα επέκτασης για τη διαχείριση μεγαλύτερων εργασιών ταυτόχρονα. Για τη επίτευξη αυτών των χαρακτηριστικών, είναι απαραίτητη μια ολοκληρωμένη συλλογή εργαλείων, η οποία περιλαμβάνει Containerization/Virtualization για την συνεπή διαχείριση του περιβάλλοντος εκτέλεσης των εργασιών, παρακολούθηση πειραμάτων για την παροχή των απαραίτητων πληροφοριών, Data/Model Tracking για την παρακολούθηση των δεδομένων και τον μοντέλων μηχανικής μάθησης, Scalable Object Storage για ασφαλή αποθήκευση δεδομένων και μοντέλων και μια μηχανή ροής εργασίας για αυτοματισμό, προγραμματισμό και παρακολούθηση της ροής εργασιών. Αυτή η διατριβή διερευνά τις προκλήσεις που υπάρχουν στο σχεδιασμό επεκτάσιμων και αναπαραγώγιμων ροών εργασίας μηχανικής μάθησης, περιγράφοντας λεπτομερώς τη χρήση των προαναφερθέντων εργαλείων για την αντιμετώπιση αυτών των προκλήσεων. Επιπλέον, αναπτύσσεται μια πραγματική μελέτη όπου υλοποιούνται οι παραπάνω τεχνικές προκειμένου να υπογραμμιστούν τα πρακτικά οφέλη μιας επεκτάσιμης και αναπαραγώγιμης ροής εργασίας μηχανικής μάθησης.

Λέξεις – κλειδιά

Τεχνητή νοημοσύνη, κύκλος ζωής μηχανικής μάθησης, λειτουργίες μηχανικής εκμάθησης, επεκτασιμότητα, αναπαραγωγιμότητα

Table of Contents

List of Tables

List of figures

Acronym Index

AD: Accuracy Difference
AUC-ROC: Area Under the Receiver Operating Characteristic Curve
Adam: Adaptive Moment Estimation
CI: Continuous Integration
CLI: Command-Line Interface
CNN: Convolutional Neural Network
CT: Continuous Training
DAG: Directed Acyclic Graph
DEVOPS: Development and Operations
DL: Deep Learning
DVC: Data Version Control
ELT: Extract Load Transform
ETL: Extract Transform Load
FFT: Fast Fourier Transform
FN: False Negatives
FP: False Positives
FPR: False Positive Rate
LSTM: Long Short-Term Memory
MAE: Mean Absolute Error
MIR: Musical Information Retrieval
MLOPS: Machine Learning Operations
ML: Machine Learning
MSE: Mean Squared Error
NLP: Natural Language Processing
NSGA-II: Non-dominated sorting genetic algorithm II
REST: Representational State Transfer
RNN: Recurrent Neural Network
R^2: Coefficient of Determination
RMSE: Root Mean Squared Error
RMSprop: Root Mean Square Propagation
SGD: Stochastic Gradient Descent
SST: Total Sum of Squares
SSR: Sum of Squared Residuals
TP: True Positives
TPE: Tree-Structured Parzen Estimator
TPR: True Positive Rate
UI: User Interface
WAV: Waveform Audio File

INTRODUCTION

Within the rapidly advancing landscape of technology, the integration of deep learning techniques into machine learning systems has become a cornerstone of innovation. This thesis delves into the intricacies of developing a Deep (or Machine) Learning Pipeline and explores the operational aspects of ML through the lens of Machine Learning Operations (MLOps), with a specific focus on addressing real-world challenges and opportunities. These advancements are particularly significant as they offer practical insights into managing and deploying ML models effectively.

This research is centered around unraveling the complexities involved in constructing a Deep Learning Pipeline and navigating the operational dimensions of ML with MLOps. A core concern is the rising intricacy of managing and deploying ML models, urging a closer examination of these processes. This exploration is exemplified through a use case—Music Genre Classification—which provides a practical lens to comprehend and apply the theoretical concepts discussed.

The overarching aim of this thesis is to provide a comprehensive understanding of the DL Pipeline and MLOps, emphasizing their pivotal roles in contemporary ML practices. The specific objectives include delving into data preparation techniques, exploring feature extraction nuances, investigating model training and evaluation processes, understanding MLOps workflows, evaluating MLOps tools, and applying these concepts to the real-world scenario of Music Genre Classification.

The methodology adopted in this thesis involves a thorough literature review to establish a theoretical foundation. Practical implementation is facilitated through a hands-on approach in developing and deploying a DL model for Music Genre Classification. This encompasses data preparation, feature extraction, model training, and evaluation, with the seamless integration of MLOps principles to ensure a holistic understanding of operational aspects.

The innovation within this thesis lies in its holistic approach to both the development and operationalization of DL models. By exploring MLOps, particularly within the context of a real-world use case, the thesis adds a practical dimension to theoretical discussions. The synthesis of these concepts fosters a profound understanding of the challenges and opportunities inherent in contemporary ML practices.

The thesis unfolds across three main chapters. Chapter 1 explores the fundamental stages of developing a DL model, emphasizing data preparation, feature extraction, model training, and evaluation. Chapter 2 investigates the operational facets of ML, highlighting the emergence of MLOps, its workflow, challenges in traditional ML workflows, and an overview of MLOps tools. Finally, Chapter 3 applies the discussed concepts to a real-world use case—Music Genre Classification—providing insights into the practical application of DL and MLOps in solving complex problems.

In pursuit of transparency and reproducibility, the entirety of the code implementation for this thesis project has been made openly accessible on GitHub. The repository, hosted at https://github.com/ioanna123/music-genre-mlops, provides comprehensive documentation and detailed instructions elucidating the purpose, functionality, and utilization of each technology employed. By embracing this open-access approach, stakeholders and interested parties can delve into the intricacies of the project, fostering collaboration, peer review, and further advancements in the field of music genre classification and MLOps.

1 CHAPTER 1: Deep Learning Pipeline

This section elaborates on the components of the DL pipeline that are systematically adopted by data scientists for purposes of research in developing ML models. The considered pipeline incorporates key phases including data preparation, feature extraction, model training, inference, and comprehensive evaluation. These stages collectively serve as building blocks towards development and holistic assessment of DL models providing a formalized and systematic manner in conducting exploration within the field of artificial intelligence and ML.

1.1 Data Preparation

Data preparation plays a seminal role in the attainment of optimal performance within the domain of DL. The careful processing and transformation of raw data at this crucial phase enable its conversion into a format suitable for efficient model training. Two crucial aspects are fundamental to the data preparation process:

1.1.1 Data Cleaning and Preprocessing

The initial and indispensable step comprises the correction of several data irregularities that may be impeding the effectiveness of DL models. Issues including the existence of missing numbers, outliers, and the requirement for data scaling are paramount among these anomalies. We carefully applied a variety of preprocessing techniques in order to demonstrate this requirement, using the example of image categorization. Notably, these methods included pixel normalization, data augmentation, and image scaling. Utilization of such methods was planned to ensure uniformity and integrity of the input image data [1].

The Figure 1 above illustrates the image before and after undergoing these preprocessing steps. The leftmost subfigure showcases the original input image [2]. Moving from left to right, the subsequent subfigures depict the normalized image, the image after data augmentation, and the scaled image, respectively. These preprocessing steps collectively contribute to improving the quality and suitability of the image for downstream computer vision tasks, thereby enhancing the overall performance and reliability of the associated ML models.

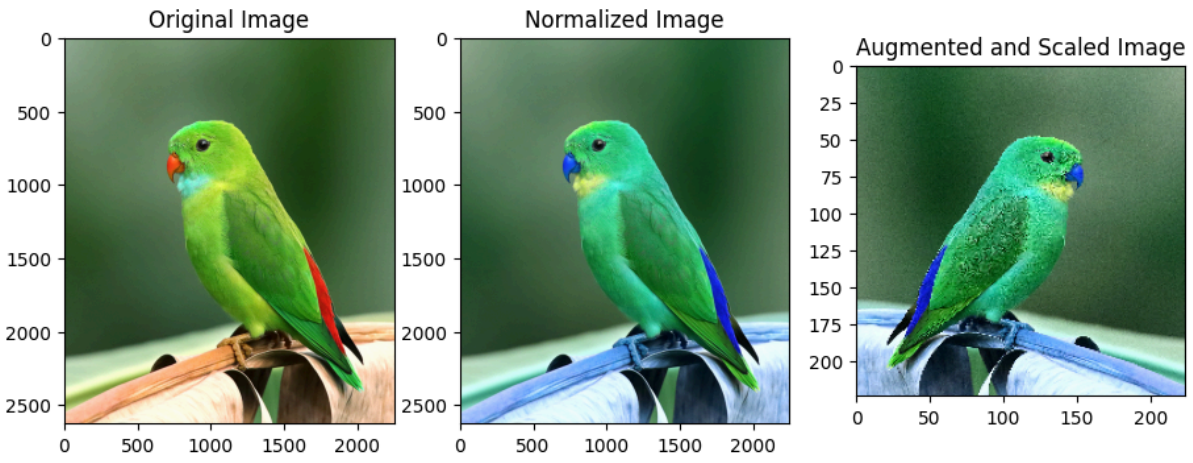


Figure 1 Image Preprocessing Steps [2]

1.1.2 Data Splitting

The second cardinal aspect of data preparation involves the partitioning of the dataset into three distinct subsets, which provides an essential foundation for DL model training and evaluation. The foundational component, the training set, acts as the crucible through which the model is developed and improved. The validation set is employed to examine model performance during training and to adjust model hyperparameters. Meanwhile, the test set is a really important step for evaluating the model's generalization ability outside the constraints of the training data [3].

1.2 Feature Extraction

Feature extraction constitutes a pivotal stage within the DL pipeline, of particular significance when addressing complex or structured data types. This phase serves to distill relevant information from the raw data, thereby affording the model a concise representation of salient characteristics. In applications related to Natural Language Processing (NLP), for instance, textual data is routinely transmuted into numerical vectors through methodologies such as word embedding, exemplified by Word2Vec and GloVe embeddings [4]. This transformation equips the model with the ability to operate upon and learn from textual data, while also facilitating meaningful semantic associations and predictive capabilities [4].

The spectrum of feature extraction techniques spans from fundamental numerical operations such as Fast Fourier Transform (FFT) for basic visual attributes to leveraging extensive pretrained models, exemplifying the evolution towards a modern foundation/downstream task paradigm in DL. This diverse range allows for the extraction of intricate and hierarchical features, empowering models to capture complexed information and excel in a variety of downstream tasks.

Similarly, when dealing with image data, feature extraction entails capturing essential visual attributes from raw images, and transforming them into numerical representations that can be processed by DL models. In Figure 2 below, we provide a visual representation of this concept. The left-hand side displays an original image, while the right-hand side showcases the extracted features, which serve as a compact representation of the image's distinctive characteristics. This transformation enables DL models to harness visual data effectively, enabling tasks such as object detection, image classification, and more.

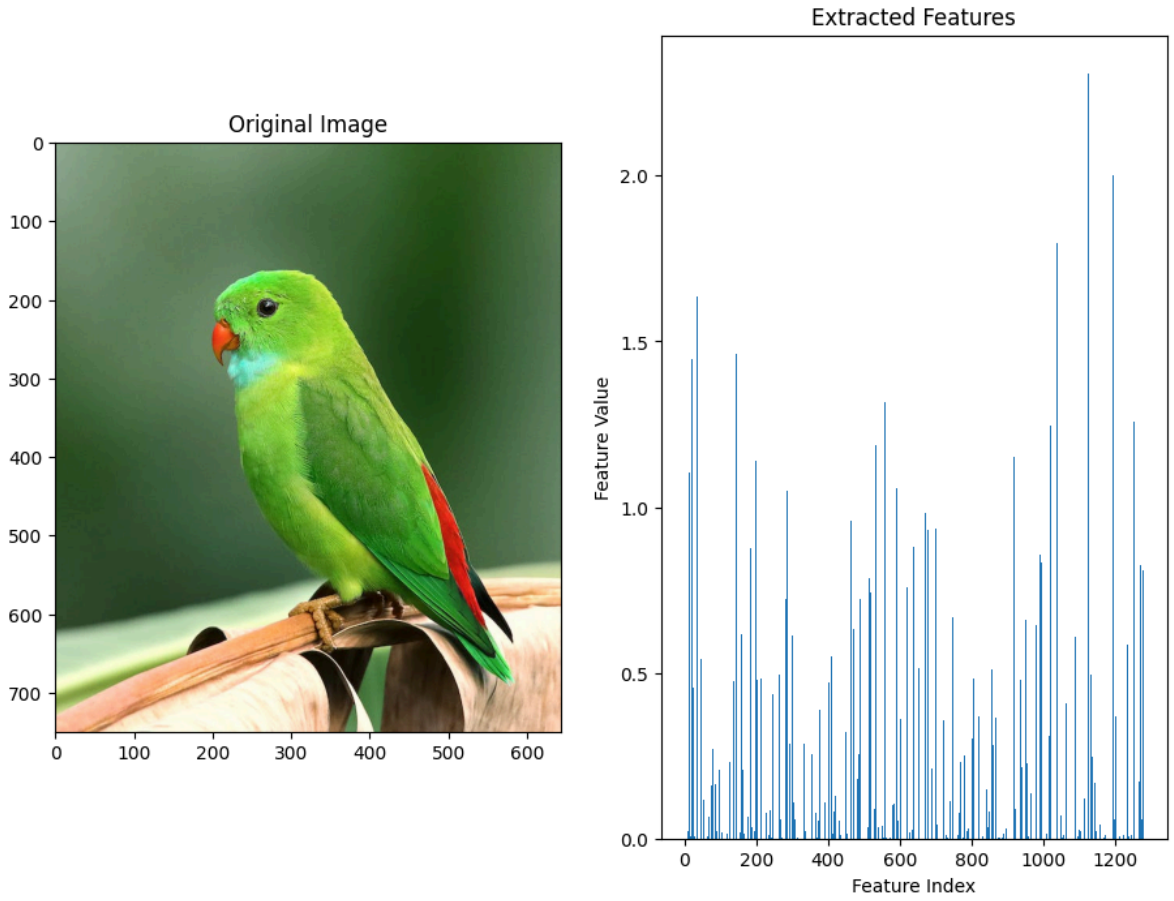


Figure 2 Illustration of Image Feature Extraction

1.3 Model Training

The process of model training serves as the cornerstone of the DL pipeline, wherein neural networks acquire the capacity to make predictions based on input data. The efficacy of this training phase is influenced by several key factors, encompassing the architectural configuration of the model, the selection of an appropriate optimization algorithm, and the fine-tuning of hyperparameters. Each of these facets plays an instrumental role in shaping the model's performance and its ability to generalize across diverse datasets.

For instance, a recurrent neural network (RNN) architecture enhanced with Long Short-Term Memory (LSTM) cells is used in the field of sentiment analysis to efficiently capture sequential dependencies in textual input [5]. The adaptive learning rate mechanism of the Adam optimizer, which accelerates convergence during training [6], serves as the foundation for its selection. To further improve training stability and effectiveness, changes to hyperparameters, particularly the learning rate—a crucial parameter driving gradient descent—as well as batch size adjustments, are highly beneficial.

In a different context, the use of a Convolutional Neural Network (CNN) architecture is crucial for image classification's effectiveness in hierarchical feature extraction [7]. The stochastic gradient descent (SGD) optimizer, a key technique for deep neural network training, serves as the foundation for learning from data. To achieve the best convergence rates and training robustness, hyperparameter fine-tuning, including learning rate and batch size optimization, is essential [8].

1.4 Model Evaluation

The assessment of DL model performance is paramount for evaluating their efficacy and generalization capabilities, a fundamental aspect in research and practical applications. The selection of appropriate evaluation metrics depends on the specific task at hand, which can include regression, classification, and other domains. Commonly employed evaluation metrics cover a broad spectrum and are each intended to address particular facets of model performance. To comprehensively assess the model's predictive capabilities in classification tasks, frequently used metrics include precision, accuracy, F1-score, recall, and the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) [9].

In the context of image classification, for instance, a suite of metrics is employed to evaluate the model's classification performance. Precision provides information about the model's capacity to generate precise positive predictions. Precision is defined as the ratio of true positive predictions to all positive forecasts. Recall, also referred to as the true positive rate, quantifies the percentage of real positive events that the model accurately recognized. When working with datasets that are unbalanced, the F1-score (equation 1), which is a harmonic mean of precision (equation 2) and recall (equation 3), offers a balanced evaluation of the model's performance. The confusion matrix is a useful tool that provides a thorough analysis of the model's predictions, including false positives, true positives, false negatives, and true negatives, allowing for a nuanced assessment of its behavior [10]. The following equation set illustrates the calculation of the aforementioned metric scores.

$$F1 = 2 * (precision * recall) / (precision + recall)$$

Equation 1: F1 score formula

$$Precision = TP/(TP + FP)$$

Equation 2: Precision score formula

$$Recall = TP/(TP + FN)$$

Equation 3: Recall score formula

On the other hand, evaluation measures for regression tasks are specifically designed to assess the predictive accuracy and deviation of the model's continuous predictions from the observed values. Metrics like mean absolute error (MAE) (equation 4), root mean square error (RMSE) (equation 5), mean squared error (MSE) (equation 6), and R-squared error (R2) (equation 7) are frequently employed and each offers a different perspective on how well the model performs in terms of prediction accuracy, error distribution, and explained variance [11]. Please refer to the set of equations presented below for a detailed description of the formulation of the regression's metric scores. The equations encapsulate the calculation process without the need for a visual representation.

$$RMSE = sqrt((1/n) * \Sigma (y_i - \hat{y}_i)^2)$$

Equation 4: RMSE metric formula

$$MSE = (1/n) * \Sigma (y_i - \hat{y}_i)^2$$
Equation 5: MSE metric formula

$$MAE = (1/n) * \Sigma |y_i - \hat{y}_i|$$
Equation 6: MAE metric formula

$$R^2 = 1 - (SSR/SST)$$
Equation 7: R^2 metric formula

1.5 Model Inference

In the DL pipeline, inference plays a pivotal role, representing the phase where a trained model applies its learned knowledge to make predictions or classifications on new, unseen data similar to the evaluation phase. During inference, input data is fed into the trained model, and the model produces output predictions based on the patterns it has learned during the training phase. This process is crucial for the deployment of DL models in real-world applications, enabling them to provide valuable insights, recognize patterns, or make decisions autonomously. Efficient and accurate inference is essential for the success of DL systems across various domains, from computer vision and natural language processing to speech recognition and beyond. Optimization of inference processes, including considerations for computational efficiency and real-time requirements, is an ongoing area of research to enhance the deployment and performance of DL models in practical applications.

2 CHAPTER 2: Understanding Machine Learning Operations: Challenges, Solutions, and Tools

This section focuses on the intricacies of deploying, managing, and optimizing ML models, addressing critical challenges, innovative solutions, and essential tools that characterize this sophisticated and dynamic field.

2.1 The Emergence of MLOps: Addressing Complexity in ML Systems

ML is clearly on the rise in modern industry, leading to innovation across a variety of applications and products [12]. However, the integration of ML introduces substantial complexity into the broader technological environment of ML-based software systems [13]. In contrast to conventional software, ML systems closely intertwine code and data, providing ongoing difficulties for system maintenance [12]. Collaboration among experts within divergent domains, development cycles, and tools further compounds this complexity [12]. MLOps, which advocates thorough automation across the ML system lifecycle, has emerged as a solution to these complex problems [1]. ML, development operations (DevOps), software engineering and data engineering are all combined by MLOps to guarantee the trustworthy and effective deployment of ML systems [14,15].

In projects dealing with massive datasets and the handling of large-scale models, the necessity for MLOps becomes increasingly apparent. The management and optimization of ML systems with enormous datasets and resource-intensive models pose unique challenges that require specialized solutions [16,17]. The automation aspect of MLOps becomes particularly crucial when dealing with large-scale projects, where manual interventions in model handling may be impractical [18]. By automating various stages of the ML lifecycle, MLOps enhances efficiency, reduces errors, and facilitates the management of intricate processes associated with extensive datasets and resource-intensive models [16,17,18].

Furthermore, the collaborative nature of MLOps (Figure 3), bringing together expertise from ML, DevOps, software engineering and data engineering, becomes particularly advantageous in projects dealing with significant data loads and complex model architectures. The synergy achieved through collaboration ensures a holistic understanding of the challenges and requirements, leading to more effective solutions [14,15]. This collaborative approach is crucial for addressing the diverse range of issues encountered in large-scale ML projects, from data preprocessing and feature engineering to big model training and deployment.

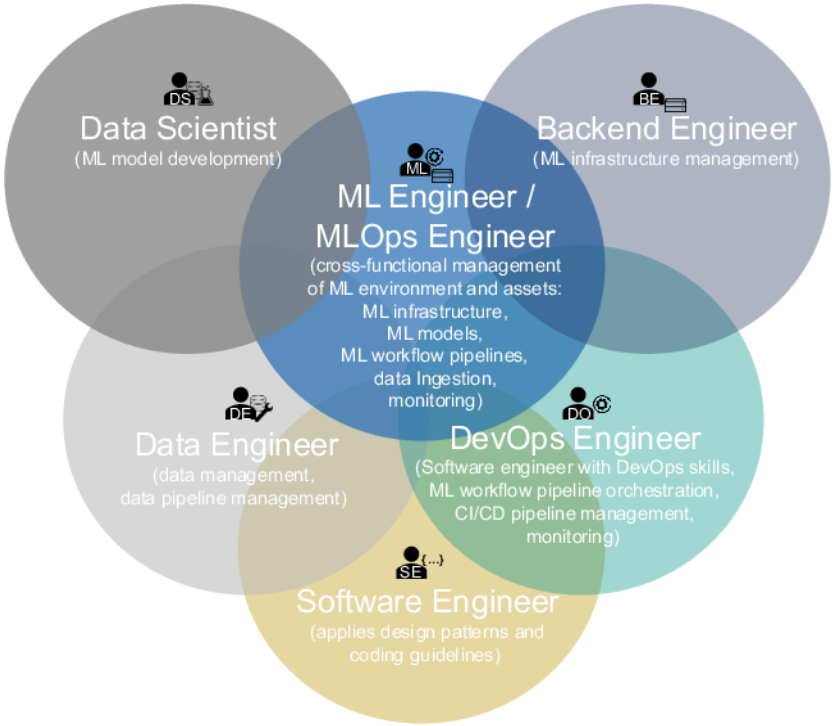


Figure 3 The roles and their overlaps that contribute to the MLOps paradigm [18]

2.2 Machine Learning Operations

The role of building an ML model is typically assigned to a data scientist, requently assisted by a domain expert. However, this role does not overlap with how the model enabler for generating business value. The operations team is in charge of deploying, operating, and overseeing the model in a production environment setting in a standard ML development lifecycle. An appropriate model is created by the data scientist and then given to the operations team for implementation.

There are various methods for delivering a trained model, with "Model as a Service" and "Embedded Model" being the two most popular ones [19]. Representational State Transfer (REST) API endpoints are used to access the model via the "Model as a Service" methodology, effectively placing the model on a web server. As a result, any application can interact with the model via REST API calls for by supplying input data through an API call. The web server can function locally or remotely. The "Embedded Model" strategy involves incorporating the model directly into an application before making it publicly accessible. In the context of the "Embedded Model" strategy, it implies that the application containing the model is accessible or usable by people outside the development or organizational context.

The choice between these deployment methods hinges on the nature of the end user's interaction with the model's output.

2.3 The MLOps Workflow

Constructing an ML pipeline is a labor-intensive task that often involves sequential development with loosely integrated tools. At its core, MLOps seeks to automate the ML pipeline; this idea was informed by the fusion of DevOps, data engineering, and ML techniques [20]. In essence, MLOps is a methodology for automating, managing, and

accelerating the operationalization of ML models (i.e., building, testing, and deploying) by incorporating DevOps practices into the ML domain.

MLOps introduces Continuous Training (CT) as a critical component in addition to incorporating Continuous Integration (CI) and Continuous Deployment (CD) from DevOps into ML workflows. The specific difficulties presented by ML are taken into account by CT, including the possible deterioration of model performance over time caused by processes like idea drift and data drift [21]. In order to guarantee consistent high-quality predictions, CT intends to automatically retrain and deploy models as necessary [22,23].

Since MLOps is a relatively new topic, no one-size-fits-all implementation method exists. There are different levels of automation. Level-0 designates the preliminary stage of contemplating ML process automation. The main objective is to ensure that the model receives continual training to support continuous delivery to production. Level 1 adds more automation components. Beyond Level 1, other models propose additional levels of maturity in MLOps implementation. Level 2 involves advanced automation techniques, such as automated model monitoring and retraining, aiming to enhance reliability and scalability. Level 3 focuses on continuous improvement, integrating automated hyperparameter tuning and feedback loops for ongoing optimization of ML processes. Finally, Level 4 represents full automation and autonomy, where MLOps processes are entirely automated with minimal human intervention, enabling self-learning systems and automated decision-making in model deployment and maintenance.

Risk managers and auditors are crucial to minimizing model-related risks and assuring project compliance throughout the MLOps project [21]. Figure 4 shows how different MLOps roles might affect the workflow at various phases. It sometimes takes several revisions and significant thought to assign roles to different stages. It's critical to define the roles at each stage for organizations new to MLOps.

The following are the primary responsibilities in the MLOps workflow:

- Data scientists work on the feasibility of the major objectives and model training, which is the foundation of ML systems, and are involved in many MLOps phases.
- Domain Experts are essential for creating domain-specific requirements and assessing outcomes in ML projects.
- Data stewards are in charge of providing the best circumstances for the usage of real-world data by ingesting it into ML pipelines.
- Data engineers convert unprocessed data into information that may be used by particular systems.
- Software engineers deploy and integrate ML solutions into applications, overseeing the entire process.
- MLOps engineers manage monitoring, continuous integration, including monitoring for optimal performance.

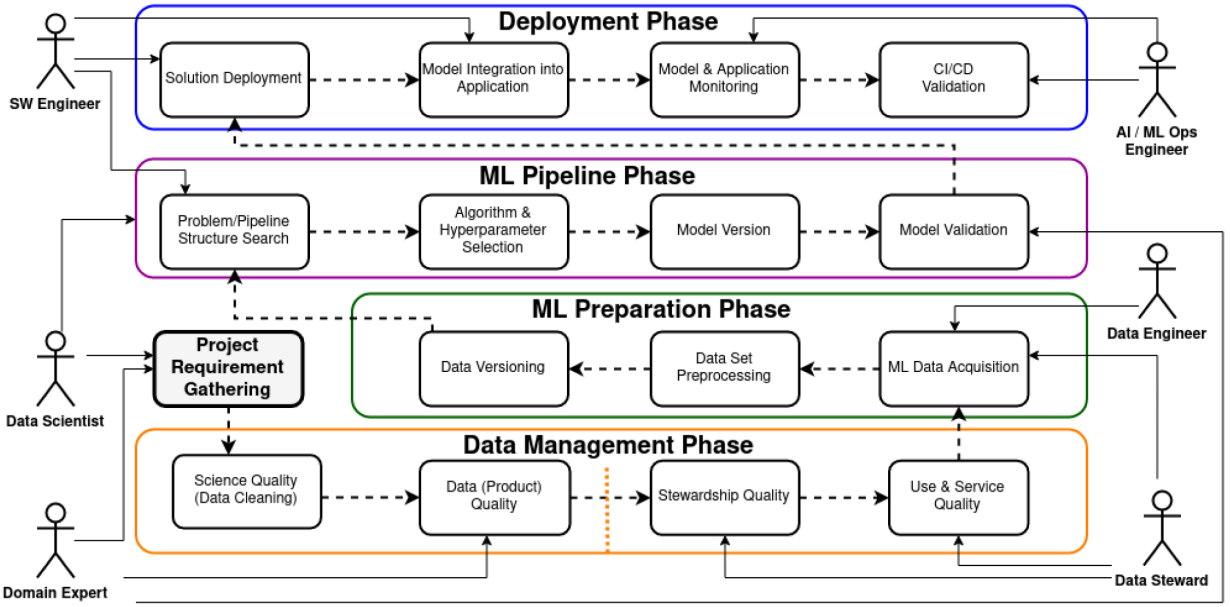


Figure 4 Common actors in MLOps and their responsibilities throughout the workflow [12]

2.4 Traditional ML Workflow Challenges

The field of ML has a unique set of issues that call for various methods, in contrast to traditional software development. The following are the main difficulties faced by the manual ML pipeline [12]:

- Metrics-Driven Process: ML largely relies on metrics to choose the best model, requiring several iterations of checks. Metrics must be manually tracked for each experiment, adding complexity.
- Manual logging: Logging is ineffective and error-prone, which results in inconsistent data.
- Data versioning: Since data is frequently not versioned, it can be difficult to replicate results as both data and code change over time.
- Lack of Model Versioning: It is frequently difficult to duplicate previous models since there is no structure in place for model versioning or model registry.
- Lack of testing and code reviews: Code reviews and testing are frequently absent in ML, in contrast to traditional software development, which results in little teamwork and code creation in solitary settings.
- Focus on a Single Model: Rather than taking into account the complete pipeline and tackling issues that degrade the overall pipeline performance, the manual approach typically produces a single model as the outcome.
- Collaboration issues: The inability to easily share models and artifacts across team members makes collaboration challenging.
- CI/CD Deficiency: By considering operations and development as independent branches, the process is made more difficult by the lack of CI and CD.
- Lack of repeatability: Experiments and their corresponding models are frequently left untracked, which prevents maintenance and repeatability.

- Limited Retraining: Due to model staleness or concept drift, deployed models cannot be automatically retrained, despite possible requirements. Model changes are sporadic as a result of this laborious approach.
- Cascade Effect: Without adequate versioning or documentation, changes to certain parameters might have a cascade effect and affect later pipeline stages.

2.5 Overview of MLOps Tooling

A wide range of tools have been developed in the field of MLOps to simplify and automate the ML workflow. The choice of these tools is based on the operational structure and the particular environment in which the machine-learning solution is being used [20]. This section provides an examination of the various requirements fulfilled by these tools. It is essential to note that different tools specialize in automating distinct phases of the ML workflow. Currently, there isn't a single open-source tool that we are aware of capable of fully automating the entire MLOps workflow. These criteria, such as scalability, compatibility with various machine learning frameworks, automation capabilities for different stages of the workflow (e.g., data preprocessing, model training, deployment, monitoring), support for version control, and security features, form a foundational framework for selecting different tools, intricately linked to the workflow introduced earlier. By identifying these requirements as key factors that MLOps tools must address, each tool can be assessed based on these criteria, facilitating the identification of tool combinations that collectively fulfill a spectrum of these demands.

2.5.1 Data Preprocessing

The quality of the input data that ML models use has a profound impact on their effectiveness. To make data suitable for use by ML algorithms, it must first go through a number of preprocessing stages. Specific requirements are imposed on ML projects during this preprocessing step.

Data Source Handling: It is necessary to employ connectors or interfaces for smooth data loading when integrating several data sources into a project's pipeline. Given the fundamental significance of data sources in any data science endeavor, skillful management of a variety of sources—from databases and files to network sockets and extensive services—is essential for the pipeline's integrity.

Data Transformations and Preprocessing: Strong support for preprocessing and data transformation is required since input data frequently arrive in a variety of disparate and incompatible forms. The preprocessing system must expand its ability to handle unstructured formats like audio, photos, and videos even though structured data, such as text or numerical values, is frequently present. In order to ensure that the ML project depends on thoroughly cleansed data, it is important to efficiently handle these various formats.

Data Management: A fundamental requirement of MLOps is the centralized management of datasets. It is essential to give consumers an easy-to-use visualization interface that may display different aspects of the gathered data (such as specific attribute value distributions or

variable changes over time). This method helps users quickly spot any potential problems or inaccuracies in the dataset.

Data Versioning: Similar to the significance of code versioning in DevOps methods, data versioning assumes the highest priority in MLOps. In ML nad DL projects, reproducibility depends on careful data versioning. Data versioning strategies can be divided into time-difference-based techniques and approaches prompted by Git. Techniques similar to Git can be employed to track immutable files, yet they are unable to capture nuanced differences between them.

Data Labeling: Data labeling appears as a necessary job in supervised learning scenarios. These datasets must be manually preprocessed, and a number of tools come in handy. When adding labels to multiple samples, they support actors like data scientists and domain experts. These labels play a crucial role in determining how the learning process is shaped, including the features that enable the model to make accurate predictions during its execution.

2.5.2 Model Training

ML model training is a challenging and resource-intensive process that is iterative in nature. ML offers several accurate solutions for a given problem, in contrast to older technologies, which frequently yield a single ideal solution [24]. Specifying requirements is necessary in order to address the difficulties experienced during model training that described below.

Selection of a Model Type: Configurability in training is achieved by allowing users to select several model types, such as decision trees or naive Bayes.

Hyperparameter Tuning: Configuring an algorithm's parameters, particularly its hyperparameters, becomes essential after choosing the algorithm. Automating this intricate process through various approaches helps alleviate the workload on data scientists.

Tracking: Robust tracking techniques are required by ML's iterative development and intrinsic approximation. In tracking, algorithmic information, hyperparameters, and performance indicators are automatically saved in a database. Collecting this meta-information simplifies the comparison of algorithms and performance metrics, which is the foundation of a model's quality.

Friendly to Integration: The integration of different frameworks into the MLOps workflow provides a variety of technical approaches to problem-solving due to the diversity of ML frameworks (such as Pytorch, Scikit-Learn, and TensorFlow).

Code versioning: Error source detection can be difficult because ML system failures sometimes lack unambiguous errors. Versioning is necessary for quickly returning to a previous functional state. MLOps uses code versioning, much like DevOps, and incorporates essential software engineering techniques like branches and rollbacks. Versioning technology is more in demand as applications become more complicated and there are more parties engaged. Code versioning encourages effective deployments and lessens the confusion that can arise during development.

2.5.3 Model Management

Effective model management is a critical pillar in the field of MLOps. The challenge of managing ML models grows increasingly difficult as businesses create a variety of them. In order to maintain consistent deployment of new models and to enable quick and painless rollback procedures in the event of failures, proper ML model management is essential. This section explores the particular requirements resulting from the difficulties in managing ML models.

Model Packaging: The technologies used are extremely comparable to how ML applications are deployed. As a result, trained model packing gains notable significance. Models may be expected to be shared across scientific groups, depending on the aspirations and objectives of MLOps projects. This demands thoughtful consideration be given to the packaging procedure.

Model Registry: The model registry's ability to save each model, including all of its versions and metadata, plays a crucial role in the industry. The model serving tool is unable to respond to model-related questions automatically in the absence of a full model registry. The process of registering models in a centralized registry, which assists in the coordination of model selection and deployment, is essential to this process. With this centralized method, productionizing previously stored models takes a great deal less time while improving visibility and governance.

2.5.4 Model Deployment

Each organization has a unique deployment strategy for ML models. The use of additional resources may be hampered by particular deployment restrictions. In the context of MLOps, taking care of this variety is essential.

Support for Different Deployment Patterns: Numerous deployment options are conceivable when taking into account the various physical places where predictions need to be carried out. Deploying models at the edge, inside container structures, or even clusters are viable alternatives to providing predictions as a service. Robust MLOps frameworks must be able to adapt to these various deployment patterns to ensure flexibility and efficacy across organizational landscapes.

2.5.5 Operations and Monitoring

Different monitoring metrics are crucial throughout the Operations and Monitoring Phase of MLOps, depending on how the MLOps project is structured and how it approaches a particular problem. The model's performance is closely related to the reliability of the input data and the system's overall infrastructure. In order to maintain the general health of the ML ecosystem, several observational features and techniques are essential. The following statement summarizes these elements.

System Monitoring: As a fundamental technique for preventing deployment-related errors, continuous monitoring of the deployment infrastructure, which includes aspects like system latency, throughput, and server load, is employed. As described below, a variety of metrics are used to track the entire system used for ML operations.

Throughput: The system's health can be assessed by the speed at which predictions are generated, typically measured as average predictions returned within a single second. A robust system is characterized by high throughput. Moreover, metrics related to throughput might be impacted by other factors such as total weekly hits, average hit size, or the frequency of errors [25]. Scale factors, such as the capacity to handle an increasing number of concurrent requests, contribute to throughput evaluations.

Disk utilization: It is essential to monitor the available disk space to prevent data loss and mitigate the risk of server downtime. Examining the metrics of the server's file system additionally supports the evaluation and understanding of the applications' health status [26].

CPU/GPU usage: Monitoring the usage of GPUs and CPUs aids in identifying instances where certain inputs lead to prolonged computation times. This data is valuable for refining the model to enhance user experience. When adjusting the capacity and configuration of the infrastructure, these resources can be resized to align with the necessary serving capabilities[25].

Serving latency: The metric assesses response time, a crucial factor for ensuring a positive user experience. To establish a comprehensive measurement, responses from all tiers must be considered. Besides the average time to respond to a request, other factors such as perceived page load time can impact the measurement [25]. Unlike throughput, which emphasizes speed and volume, serving latency prioritizes the timeliness and responsiveness of individual user interactions.

Input Data Monitoring: Failure to continuously monitor incoming application data can lead to oversight regarding error factors like data or concept drift. Such oversights can have significant impacts on subsequent model utilization. Various aspects of monitoring the respective datasets are outlined in the following list.

- **Data consistency**: This measure assess whether the production data meets the range that was predicted. Errors in data format or type can also be found in it. Another component of consistency can be assessed by comparing the quantity of records with previously determined states of the dataset [27].
- **Data drift:** This metric examines alterations in the data distribution, checking if the statistical structure of the input data (changes in input feature space) has shifted over time[12].
- **Data quality:** This metric evaluates how closely the model's training data matches the quality of the data itself. The performance of the model is directly impacted by changes in data quality. Therefore, it is crucial to maintain several aspects of data quality at every point in the data lifecycle [28].

- **Data schema:** A fundamental requirement of MLOps is the centralized management of datasets. It is essential to give consumers an easy-to-use visualization interface that may display different aspects of the gathered data (such as specific attribute value distributions or variable changes over time). This method helps users quickly spot any potential problems or inaccuracies in the dataset [27].
- **Outliers:** Many algorithms perform poorly on such data points because they are vulnerable to outliers. Outliers should be properly investigated because they might offer important insights about the use case. The measurement of data veracity is made possible by comparing incoming data against well-known sources or validating with mathematical formulas [27].

Model's Performance Monitoring: Model staleness is the occurrence when a model fails to operate as expected over time as patterns in the incoming data change significantly. Retraining the model becomes crucial in such circumstances. For the model to remain effective, performance must be closely monitored. The metrics shown below highlight numerous problems that need to be assessed both before and during production deployment.

- **Model's performance**: The application of this statistic depends on the nature of the current work. It helps assess the suitability of the current model. Examples of classification tasks include recall, precision, AUC score, false positive rate (FPR), true positive rate (TPR), and Confusion Matrix. Because of the potential for significant delays between output and accurate labeling, calculating these measures can be difficult. In general, the accuracy (right predictions), relative gains (in contrast to other models), and credibility (the veracity of the resulting predictions) of a model can be used to assess its performance [29].
- **Feature Importance Change:** A model's overall performance could not decrease, but the feature importance that underlies predictions might change. These changes might be a symptom of drift and might highlight the need for data collection during retraining.
- **Output Distribution:** An indication of probable model degradation is the difference in the distribution of projected outputs between training and production data. When such changes take place, metrics like the Population Stability Index or Divergence Index act as helpful indicators by sending out alerts. This statistic is crucial for evaluating the applicability of existing models to new data, especially in time-series datasets [29].
- **Concept Drift:** Concept drift, which implies that the patterns the model learned in the past are no longer true, is a primary cause of model performance degradation. The work in [30] is cited for a thorough investigation of pasive and active idea drift detection in distributed systems.
- **Bias/Fairness:** A trained model may have unintentional biases toward particular classes or groups. To find such biases, Accuracy Difference (AD) can be assessed for each class. Alternatively, the model can be evaluated on particular data subsets to get a more in-depth insight into how well it performs [12].

2.5.6 General Requirements

There are important elements to take into account while adopting tools, corresponding with common criteria applicable across many software domains, in addition to special requirements customized for ML. The first fundamental necessity is scalability, leading to the need for flexibility in processing power, storage, and service latency depending on the needs of the individual project. Second, the availability of APIs is essential since it makes it possible to programmatically control tools. Automating activities improves the robustness of the MLOps pipeline and lowers manual mistakes caused during tool usage and configuration. Additionally, it is essential to assess the maturity level of tools, taking into account elements like thorough testing and wide acceptance, in order to assure reliability in production situations and protect business objectives from potential errors.The need for user-friendliness should therefore not be underestimated, particularly in the numerous MLOps workflows that involve both technical and non-technical personnel. The process is more efficient overall because of self-explanatory features and intuitive interfaces that promote seamless collaboration.

2.6 MLOps Tools Overview and Comparison

In order to identify key indicators and support people in their decisions to adopt particular technologies, it is vital to provide a concise review of the tools that are currently accessible. In the parts that follow, we go in-depth on the critical evaluations of each tool, which are crucial for choosing tools to set up a successful MLOps workflow.

2.6.1 Experiment Management and Model Lifecycle Tools

Experiment management and model lifecycle tools, such as MLFlow, Comet ML, Weights & Biases, and Polyaxon, play a vital role in meeting the diverse needs of ML practitioners. These tools simplify crucial aspects of the ML lifecycle, offering capabilities such as experiment tracking, reproducibility assurance, model deployment, and comprehensive model management. Whether it's streamlined experiment display, complete ML platforms, or emphasis on teamwork, these tools collectively enhance collaboration, efficiency, and effectiveness in the dynamic landscape of ML.

- An open-source tool called **MLFlow** [31] was created to simplify crucial steps in the ML lifecycle. Although its primary use is as an experiment tracker, it may also be used to ensure reproducibility, deploy models, and keep track of model registries. Using a range of interfaces, including CLI, Python, R, Java, and REST API, users can efficiently manage ML experiments and model metadata.
- **Comet ML** [32] is a flexible framework for observing, contrasting, deciphering, and improving ML experiments and models. It meets the demands of people, teams, businesses, and academia and is compatible with many machine-learning frameworks, including Scikit-learn, TensorFlow, PyTorch, and HuggingFace. It makes experiment display and comparison easier while allowing for the analysis of data samples from many sources.

- **Weights & Biases** [33] offers a complete ML platform that includes model management, data and model versioning, hyperparameter optimization, experiment monitoring. Moreover, it allows logging for a variety of artifacts and provides visualization tools for a variety of data kinds. Monitoring ML trials is made easier by the single dashboard's easy integration with a variety of ML frameworks.
- A tool named **Polyaxon** [34] focuses on two things: first, it encourages teamwork within ML teams, and second, it concentrates on the full life cycle administration of ML projects. It supports code and data versioning, unlike MLflow, and deals with user management. The platform requires Kubernetes and is not entirely open-sourced. As a result, it is not entirely independent of architecture. Any job can be executed using its interface on a Kubernetes cluster. For small teams, the platform might not be necessary because of the longer setup time.

Tool	Features	Compatibility	Limitations
MLFlow	Simplifies ML lifecycle steps, acts as an experiment tracker, ensures reproducibility, deploys models, and manages model registries.	Compatible with various languages like Python, R, Java	Limited collaboration features, primarily focused on experiment tracking.
Comet ML	Flexible framework for observing, contrasting, deciphering, and improving ML experiments and models. Facilitates easy experiment display, comparison, and data analysis from multiple sources.	Compatible with various ML frameworks like Scikit-learn, PyTorch, TensorFlow, and HuggingFace	Complex setups migh require a learning curve for new users.
Weights & Biases	Complete ML platform offering model management, hyperparameter optimization, experiment monitoring, data, and model versioning. Allows logging for various artifacts and provides visualization tools.	Compatible with multiple ML tools and frameworks	Limited free tier, may require subscription for extensive usage.

Table 1 Experiment Management and Model Lifecycle Tools Comparison

2.6.2 Workflow management and orchestration tools

In the dynamic field of data science and ML, the efficient orchestration of workflows is crucial for streamlined operations. Various tools cater to diverse needs, each offering unique strengths. These tools contribute to tasks ranging from task dependency resolution and workflow visualization to parallel job execution. The landscape includes solutions that seamlessly integrate into the Python ecosystem, automate ML experiments, and optimize workflow design. Rooted in Python, some tools excel in developing reproducible and modular data science projects, while others focus on flexibility across on-premises and cloud environments.

This diverse array of tools collectively enhances the orchestration and management of ML workflows, contributing to a more efficient and effective workflow lifecycle.

- A platform for managing workflows called **Airflow** [43] includes features for managing and scheduling activities (represented as DAGs). In contrast to the kubeflow utility, airflow is intended to be integrated into the Python ecosystem rather than being exclusive to Kubernetes. It has a steep learning curve for novice users and is not designed to be intuitive. Python can be used to define tasks and orchestrate a straightforward pipeline. Additionally, integrating Airflow for ML applications that are already in progress is rather challenging.
- **Prefect** [35] is a modern data management tool made to keep track of, organize, and orchestrate activities between different applications. It is a lightweight, open-source tool that supports entire machine-learning pipelines. Users who choose Prefect Orion UI or Prefect Cloud for database operations can learn more about the state of their workflows, team collaboration, and orchestration.
- A powerful workflow management solution designed specifically for data science and ML applications is called **Metaflow** [36]. It enables data scientists to put more effort into model development and less effort into MLOps engineering. Metaflow automates ML experiments and data tracking and versioning while streamlining workflow design, scalability, and model deployment. It supports numerous Python packages, including Scikit-learn and TensorFlow, and several cloud service providers.
- The workflow orchestration tool **Kedro** [37], which has its roots in Python, emerges as the best choice for developing reproducible, modular data science projects. It integrates ideas from software engineering into MLg, encouraging modularity, concern separation, and version control. Kedro gives customers the ability to manage data, view and run pipelines, track experiments, deploy on single or distributed machines, and define dependencies. This promotes maintainable and reusable code.
- An easy-to-use tool for building ML Pipelines is **ZenML** [38]. It offers the flexibility to switch between on-premises and cloud settings quickly. It focuses on building pipelines utilizing a series of actions made possible by Apache Beam. It offers a user interface (UI) for comparing various pipelines in a single ZenML repository and visualizing pipelines. But at the moment, it doesn't support scheduling pipelines. It provides strong, out-of-the-box connectors with many different backends, including Google AI Platform, Sagemaker, Cortex, Dataflow, and Kubernetes.
- A particular tool for orchestrating ML workflows is called **Kubeflow** [39]. The tool's strong dependency with Kubernetes makes it more challenging to configure, which is one drawback. The major objective of the tool is to simplify interactions between the Kubernetes workflow management and the ML lifecycle.
- **Flyte** [40] platform focuses on orchestrating data processing workflows and ML. It offers a modular and adaptable design . It is directly based on Kubernetes and thus has all the advantages of containerization, but it is also challenging to configure. It employs "workflow" as a fundamental notion and "task" as the smallest computational unit. Prometheus [41] and Grafana [42] are integrated for monitoring. It can construct portable cross-cloud pipelines. Scala, Java, and Python all have SDKs.
- **MLRun** [55] represents an open-source orchestration tool for workflows and pipelines, offering an inclusive method for structuring machine learning pipelines. It

covers the entire process, starting from the initial development and model building to the eventual deployment of the entire pipeline in a production environment. MLRun utilizes an abstraction layer, seamlessly integrated with a diverse set of ML tools and plugins designed for working with features, models, and workflow deployment. It incorporates a feature and artifact store to manage the ingestion, processing, metadata, and storage of data across various repositories and technologies.

- **Argo** [56] is a container-native, robust, open-source workflow engine tailored for coordinating parallel tasks on Kubernetes. Developed as a Custom Resource Definition, Argo enables defining pipeline workflows with individual steps executed as containers. It facilitates modeling multi-step workflows using a directed acyclic graph (DAG) for dependency tracking. DAG is a graph structure with nodes connected by directed edges, representing dependencies between tasks or processes in a way that ensures a specific order of execution without forming cycles. Argo is particularly adept at handling resource-intensive tasks in ML and data science, optimizing time efficiency.
- **Luigi** [57] is an open-source Python package designed for efficient workflow orchestration, specifically for batch tasks. It simplifies the construction of complex pipelines and provides services for workflow management, dependency resolution, failure handling, visualization, and command line integration. Primarily tailored for long-running, intricate batch processes, Luigi manages time-consuming workflow management tasks, allowing users to focus on the actual tasks and their dependencies. It includes a toolbox with common project templates.

Tool	Key Features	Supported Technologies	Ease of Use
Prefect	Workflow tracking, team collaboration, orchestration.	Entire ML pipelines, Prefect Orion UI, Prefect Cloud	Moderate
Metaflow	Automated ML experiments, data tracking, versioning.	Python packages like Scikit-learn, TensorFlow, Various cloud service providers	High
Kedro	Pipeline management, experiment tracking, dependency definition.	Python, compatible with multiple backend systems, Single or distributed machines	Moderate
ZenML	Apache Beam integration, UI for pipeline comparison.	Google AI Platform, Sagemaker, Cortex, Dataflow, Kubernetes.	Moderate
Kubeflow	Simplified interaction between Kubernetes and ML lifecycle.	Kubernetes	Moderate
Flyte	Workflow and task-based approach, Prometheus and Grafana integration.	Kubernetes, supports Scala, Java, Python SDKs, Prometheus, Grafana	Moderate
MLRun	Workflow/pipeline orchestration, feature and artifact store,	Dask, kuberflow, Pytorch, Tensorflow, aws, amazon	High

	server-less service, centralized UI		
Argo	Orchestration, parallel job execution Kubernetes	Kubernetes	High
Luigi	Workflow orchestration, batch task optimization	Python	Moderate

Table 2 Workflow management and orchestration tools Comparison

Please be aware that scalability and simplicity of use are relative concepts that can change depending on the user's experience and particular use cases. Assessing each tool in light of the specific project needs and team skill levels is critical.

2.6.3 Tools for Data and Pipeline Versioning

In the field of data and pipeline versioning, diverse solutions cater to the intricate requirements of overseeing code, data, models, metadata, and pipelines. These tools offer functionalities such as version control, lineage tracking, continuous integration, and robust data quality testing. From open-source solutions tightly connected with Git, like Data Version Control (DVC), to cloud-based options such as Quilt data for AWS services, each tool brings unique strengths to data versioning and management. Whether automating conversions, conducting automated tests for data quality, or revolutionizing file management, these solutions collectively contribute to the comprehensive landscape of versioning in the context of ML projects.

- **Pachyderm** [44] automates data conversions and provides Kubernetes end-to-end pipelines, data versioning, and lineage tracking. Due to its adaptability, it can be scaled for both small teams (Community edition) and corporations (Enterprise edition) and can be integrated with a variety of data types and languages. Pachyderm enables data versioning similarly to Git by using repositories, commits, branches, files, history, and provenance.
- The versioning of code, data, models, metadata, and pipelines is made possible by Data Version Control, an open-source solution that is tightly connected with Git. **DVC** [45] provides tools for continuous integration and deployment, pipeline visualization, repeatability, data and model registries, experiment tracking and continuous integration for ML projects utilizing Continuous ML in addition to version control.
- When employing AWS services for the ML process, **Quilt data** [46] stands out as a highly suitable option for managing data versions. It not only provides robust data quality testing capabilities but also facilitates seamless data sharing across distinct projects. This tool comprises a Python client, a web catalog, and a backend system designed to efficiently handle datasets stored in S3 buckets.
- **Great Expectations** [47] serves as a robust tool for both data validation and profiling. It boasts seamless integration capabilities, allowing easy extension with various frameworks. This versatile tool offers seamless integration with numerous data sources such as MySQL, SQLAlchemy, and more. Its primary objective is to conduct

automated tests for data quality assessment, although it's important to note that it's specifically tailored for tabular data.

- Atlassian developed an extension for Git known as **Git-lfs** [48], which revolutionized file management by introducing the idea of tracking pointers instead of the physical files themselves. In contrast to DVC, Git-lfs necessitates the installation of a specific server. However, these servers do not possess the scalability that DVC offers. Git-lfs is primarily employed for storing sizable files. It's worth noting that when used in conjunction with GitHub, there's a file size limit of 2 GB, due to GitHub's restrictions on Git LFS.

Tool	Versioning	Data Quality Testing	Use Case
Pachyderm	Git-like (Repositories, Commits, Branches)	Not Specified	Pipelines, Data Versioning, Kubernetes
DVC	Git-like (Repositories, Commits, Branches)	Yes (Continuous Integration for ML Projects)	CI/CD, Deployment, Versioning
Quilt Data	Compatible with Git	Robust Data Quality Testing	Data Versioning, Data Quality Testing, Data Sharing
Great Expectations	Not specified	Yes (Automated Tests for Data Quality)	Data Validation, Profiling, Quality Assessment
Git-lfs	Compatible with Git	No	Large File Management

Table 3 Tools for Data and Pipeline Versioning Comparison

2.6.4 Tools for Model Deployment and Serving

In the landscape of model deployment and serving, various solutions address the complexities of deploying, managing, and maintaining ML models in production. These tools offer comprehensive solutions, encompassing aspects such as model experimentation, training, versioning, and scalability. With a focus on diverse frameworks and adaptable deployment options, these solutions contribute to the seamless integration of ML applications into production environments.

- By providing a complete solution for ML model experimentation, training, deployment, and maintenance, **TFX Serving** [49] makes it easier to use taught models as endpoints. It offers capabilities like batch request initiation, model versioning, and hardware efficiency, with a focus on TensorFlow models. It also supports scalability with Docker and Kubernetes.
- By offering Python-centric tools for production-ready APIs, **BentoML** [50] simplifies the deployment of ML applications. It features enhancements including adaptive

batching, parallel inference, and hardware acceleration. BentoML's interactive dashboard supports some ML frameworks, including Keras, ONNX, LightGBM, PyTorch, and Scikit-learn, and makes it easier to organize and monitor models.

- **Cortex** [51] is a flexible, multi-framework, open-source tool for serving and monitoring models. Scalability and flexibility are ensured by its integration with TorchServe, Docker, Kubernetes, TFX Serving, and other ML libraries. Multiple models can be deployed on a single API endpoint with Cortex, and auto-scaling is supported for improved API security and management.

Tool	Focus	Deployment Capabilities	Scalability	API Support	Flexibility
TFX Serving	TensorFlow models	Batch request initiation, model versioning, hardware efficiency	Docker, Kubernetes	TensorFlow	Specific to TensorFlow models
BentoML	Multiple frameworks	Adaptive batching, parallel inference, hardware acceleration	Supports scalability with multiple frameworks	Multiple frameworks (Keras, ONNX, etc.)	Supports various frameworks and Python-centric tools
Cortex	Multiple frameworks	Integration with various ML libraries, Docker, Kubernetes	Auto-scaling, supports Docker, Kubernetes	TorchServe, Docker, Kubernetes, TensorFlow Serving	Flexible, integrates with multiple ML libraries

Table 4 Tools for Model Deployment and Serving Comparison

2.6.5 MLOps Tools for Model Monitoring in Production

In the domain of MLOps, the critical aspect of model monitoring in production is addressed by various tools that provide comprehensive solutions for overseeing the entire lifecycle of ML models. These tools offer functionalities ranging from data and model quality checks to real-time monitoring of deployed services, interactive dashboards for performance evaluation, and proactive issue identification. By supporting end-to-end AI observability and providing intuitive user interfaces, these solutions contribute to the effective management and optimization of ML pipelines, ensuring the reliability and performance of models in real-world production scenarios.

- In **Evidently AI** [52], ML model lifecycle monitoring is supported by an open-source Python library, from development to validation and from validation to production. It checks the quality of the data and the models, looks for data drift and target drift, and measures the effectiveness of regression and classification. Evidently includes

real-time monitoring for deployed ML services, interactive dashboards for data drift and model performance and tests for batch model checks.

- A ML model monitoring tool with an intuitive user interface is **Fiddler AI** [53]. It makes it easier to explain models, debug them, analyze entire datasets, deploy scalable models, and monitor models' performance thoroughly. Performance tracking, data integrity checks, outlier identification, service metrics, and custom warnings for model or group performance issues in production are its main strengths.
- An end-to-end AI observability platform with autonomous monitoring and proactive problem-solving is provided by **Censius AI** [54]. It makes it possible to monitor the entire ML pipeline, explain predictions, and fix problems. Censius AI may be set up on-site or in the cloud and allows integration using Python or Java SDKs and REST APIs. Features include data explainability for different dataset types, real-time alerting, customized dashboards, and support for A/B tests.

Tool	**Monitoring Scope**	**Alerting Mechanisms**	**Data and Model Quality Checks**
Evidently	ML model lifecycle, from development to production	Tests for batch model checks	Yes
Fiddler AI	Model performance in production	Custom warnings for model or group performance	Data integrity checks
Censius AI	Entire ML pipeline	Real-time alerting	Explainability for different dataset types

Table 5 MLOps Tools for Model Monitoring in Production Comparison

3 CHAPTER 3: Use Case Implementation: Music Genre Classification

This chapter introduces the implementation of a music genre classification use case. Specifically, it defines the music genre classification problem, provides an overview of the dataset, and elaborates on the ML Pipeline that includes preprocessing, feature extraction, training, and evaluation. Additionally, it highlights the significance of MLOps in strengthening the robustness of the ML pipeline, utilizing various tools for enhanced effectiveness.

3.1 Music Genre Classification Problem Definition

Throughout human culture, music has remained an integral part, serving as a direct outlet for emotions, entertainment, and social interaction. As different eras are shaped by diverse social phenomena, fashion, and artistic influences, new musical genres emerge while older ones gradually fade away. The digitization of the music industry has led to significant transformations in music creation, reproduction, and distribution, streamlining many processes, but also creating new needs and challenges. One such challenge is the automated recognition of musical genres.

The classification of music into categories, such as genres, artists, or moods, remains an ongoing concern in musical information retrieval (MIR). In particular, the classification of musical genres has made significant progress in recent years, with numerous studies yielding remarkable results. At the same time, research and efforts to enhance the classification of musical subgenres are actively continuing, as distinguishing between similar categories is an even more significant barrier.

The choice of implementing a music genre classification system as a use case for this thesis is driven by its multifaceted relevance in the context of deep learning pipelines. Music genre classification not only encapsulates the complexity of audio signal processing but also serves as a compelling domain for exploring the intricacies of deep learning techniques. Moreover, by leveraging machine learning operations (MLOps), the thesis aims to highlight the significance of automating deep learning pipelines for efficient model development, deployment, and maintenance. This use case provides a practical avenue to delve into the challenges of managing and optimizing the entire lifecycle of deep learning models, from data preprocessing to model evaluation and deployment, thereby emphasizing the importance of MLOps in streamlining and scaling the development process.

This section focuses on developing a musical genre classification system capable of categorizing music into ten fundamental genres: hip-hop, country, disco, metal, reggae, blues, rock, classical, jazz, and pop. Deep learning techniques are used for this purpose. The initial step involves audio preprocessing, and then extracting valuable features using digital signal processing methods. These features are then fed into various neural network architectures. Ultimately, all experiments are meticulously evaluated to identify the optimal features and architectures for the classification system.

3.2 Understanding the Dataset for Music Genre Classification

A common dataset used for the music genre classification use case is the GTZAN set, originally introduced into the literature in 2002 by C.Tzanetaki and Fr. Cook [58]. This collection includes 1000 WAV audio files, each lasting 30 seconds. These audio files are

categorized into ten main music genres: rock, jazz, metal, pop, classical, reggae, blues, hip-hop, disco, and country. The dataset comes with an equal distribution of music tracks across the ten genres, with 100 tracks in each genre.

GTZAN is a relatively small data set especially for training deeper ML architectures.However, one of its strong points is that it is fully accessible to the general public. Due to its accessibility and balanced distribution of genres, it has been widely used and cited in the literature, with over 4000 publications using it as the primary evaluation dataset for problems classifying musical genres. However, subsequent investigations revealed some errors in the GTZAN dataset, including track repetitions and mislabeling [59]. These errors, while not rendering the dataset unusable, are worth considering during any evaluation involving that dataset.

3.3 Machine Learning Pipeline: A Methodical Approach for Music Genre Classification

This subsection describes the ML pipeline for music genre classification from a data scientist's perspective. Encompassing audio preprocessing, feature extraction, training, and evaluation methodologies, it provides a comprehensive exploration of the implemented approach.

3.3.1 Audio Preprocess

In audio preprocessing, the conversion of audio files to the WAV format is a crucial initial step, preserving original data integrity. This format, known for its uncompressed nature, aligns with seminal research emphasizing improved performance in music classification. Subsequently, a streamlined streaming function is introduced to handle extensive audio files, optimizing memory utilization. These preprocessing steps set the stage for the nuanced analyses and feature extraction integral to music genre classification.

Convert audio file to WAV

In the domain of audio analysis and processing, the conversion of audio files to the WAV format plays a pivotal role as an essential preliminary step. WAV (Waveform Audio File Format) serves as an uncompressed audio format, ensuring the preservation of the original audio data without any loss incurred during compression. This inherent characteristic is crucial for maintaining the accuracy and integrity of audio data during the preprocessing phase, thereby enhancing the reliability and precision of subsequent analysis and processing efforts. Additionally, the widespread prevalence and comprehensive support for the WAV format across diverse platforms make it an accessible and compatible choice, firmly establishing its utility for audio-related applications.

To underscore the importance of incorporating WAV conversion as a preprocessing step, seminal research conducted by Tzanetakis and Cook (2002) [58] highlights the numerous advantages of utilizing uncompressed audio formats, like WAV, in Music Information Retrieval (MIR) tasks. Their work demonstrates that this approach leads to improved performance and increased accuracy in music classification and genre recognition. In the present study, the initial preprocessing step involves converting audio files to the WAV format, if they initially exist in a different format. Despite the dataset already containing audio in

WAV format, it is crucial to ensure that any new audio file introduced to the pipeline adheres to the WAV format. This is achieved using the ffmpeg suite [60], which seamlessly converts the audio file to WAV, ensuring the preservation of the quality of the original audio data with minimal to zero loss during the conversion process.

Audio Streaming

In the field of music genre classification, employing a streaming function becomes crucial when managing extensive audio files. The challenge of processing entire audio files is further complicated by computational demands and memory constraints. To overcome these challenges, a stream function was developed that adopts a segmented approach to handle audio data in smaller units. Utilizing the librosa library [61], this function selectively loads designated segments of the audio file, allowing sequential processing and mitigating memory limitations. This preprocessing technique proves essential for the effective analysis and feature extraction from expansive music collections, facilitating the training of ML models. Despite the fact that the audio files in this dataset have short durations, it remains crucial to ensure that any new audio file introduced to the pipeline undergoes proper streaming. This precaution is imperative to prevent memory spikes during training or inference and enables parallelized processing, optimizing both computational efficiency and memory utilization.

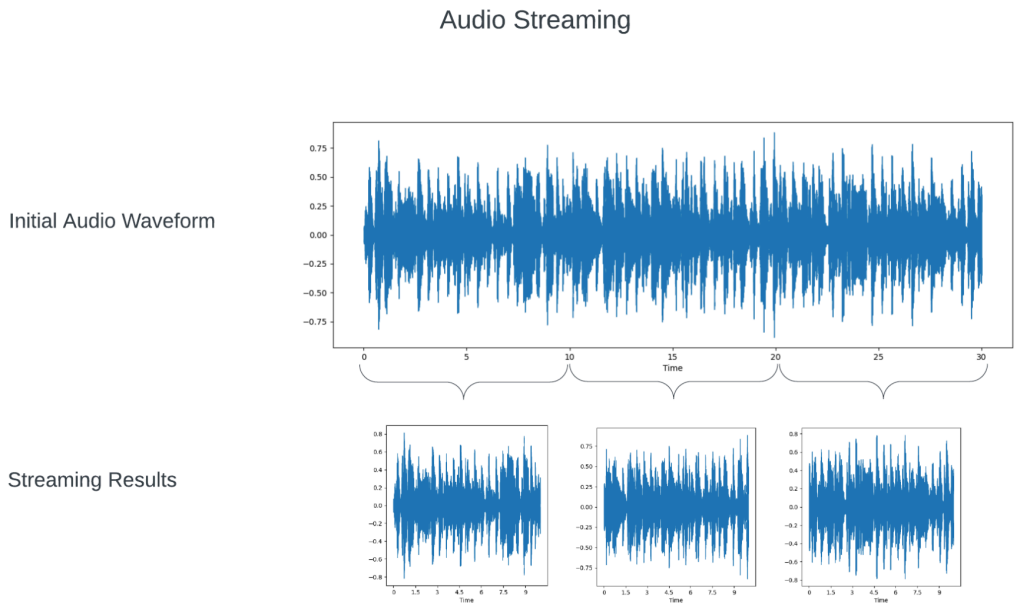


Figure 5 Audio waveform transformation

The Figure 5 above illustrates the audio waveform transformation through a streaming process with a window duration of 10 seconds. The initial audio waveform spans a duration of 30 seconds. As part of the streaming process, the audio is segmented into three distinct waveforms, each with a duration of 10 seconds. The first segmented waveform represents the audio content from 0 to 10 seconds, the second segmented waveform encapsulates the audio from 10 to 20 seconds and the third segmented waveform captures the audio content spanning 20 to 30 seconds.

3.3.2 Feature extraction

A critical stage in deep learning-based audio categorization is feature extraction, which enables the conversion of raw audio data into meaningful representations that machine and DL models can efficiently utilize. This procedure is essential for enhancing the precision and effectiveness of music genre classification systems, as it allows the identification of distinctive patterns and traits in audio data.

Representation of the Mel Spectrogram

In the initial step of feature extraction, the most common representation involves the utilization of the Mel spectrogram [73]. The Mel spectrogram concept offers a unique approach to audio representation, particularly well-suited for tasks like music genre classification. Unlike traditional spectrograms, which linearly map frequency to space, the Mel spectrogram incorporates the non-linear frequency perception of the human auditory system. This means that it emphasizes lower frequencies, which are more discernible to the human ear, while compressing higher frequencies. By doing so, the Mel spectrogram effectively captures the essence of how humans perceive sound, resulting in a representation that aligns more closely with our auditory experience.

This tailored representation proves invaluable for discerning distinct qualities within audio segments, especially in the context of music genre classification. By encoding audio streams into Mel spectrograms, we can effectively capture key spectral patterns, timbral information, and other crucial audio features that contribute to genre classification. This enables the model to recognize subtle nuances in the audio data that may be indicative of specific genres, ultimately enhancing the classification accuracy.

Compared to alternative feature extraction methods, such as raw audio waveforms or traditional spectrograms, the Mel spectrogram offers several advantages. Firstly, it provides a more compact representation of the audio data, facilitating faster computation and reducing the computational burden of subsequent processing stages. Additionally, its focus on perceptually relevant frequency bands enhances the discriminative power of the extracted features, leading to more robust and accurate classification results.

The primary step in feature extraction involves converting the input audio segment into a Mel spectrogram using the librosa.feature.melspectrogram [62] function. During this process, the audio segment is partitioned into short time frames, and the power of various frequency components is then mapped onto a Mel scale. This transformation is crucial for extracting essential spectral information needed for audio classification tasks [63,64]. This conversion effectively shifts the audio signal from the time domain to the time-frequency domain, laying the groundwork for subsequent feature extraction processes.

Conversion of Amplitude to dB

The second part of the feature extraction employs amplitude-to-decibel (dB) conversion to augment the interpretability and discriminative capabilities of the extracted Mel spectrogram. The process involves adjusting the amplitude values of the Mel spectrogram to a logarithmic dB scale, a transformation facilitated by the librosa.amplitude_to_db function. This conversion

enhances the visibility of lower amplitude components and provides a perceptually accurate representation aligned with human auditory perception [65].

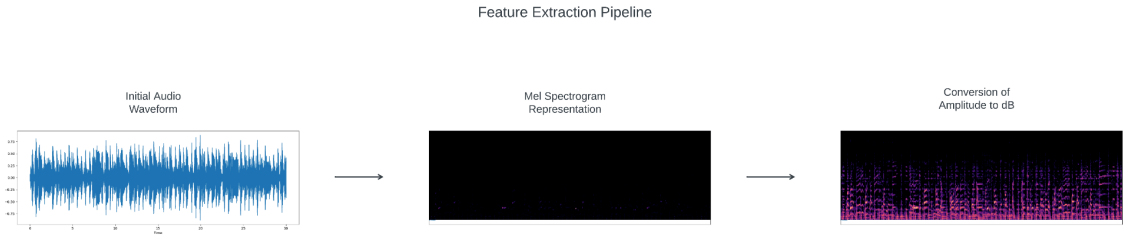


Figure 6 Feature Extraction workflow

The Figure 6 outlines the audio feature extraction workflow, starting with the initial audio waveform, transitioning to the Mel spectrogram for a time-frequency representation, and concluding with the conversion of amplitude to dB for enhanced interpretability. Each step contributes to a nuanced understanding of the audio content.

3.3.3 Deep Learning Training using Pre-Trained Models

This section presents the training process for DL models in audio classification, with the goal of achieving precise genre classification. It introduces the selected pre-trained models and outlines essential components of the training procedure to streamline and enhance effectiveness.

Transfer Learning Approach

This thesis prioritizes establishing a scalable and reproducible machine-learning workflow for music genre classification. To achieve this objective, the focus is on implementing a scalable and robust ML pipeline, using a transfer learning approach instead of custom model development. The strategic foundation for audio categorization is laid by leveraging pre-trained models, crafted with extensive datasets.

The selected pre-trained models include:

- Convolutional neural network (CNN) pioneer AlexNet was the first design to use DL for image recognition [66].
- DenseNet-121, a densely connected CNN, makes feature reuse between layers easier, improving feature propagation and convergence [67].
- ResNet-18 and ResNet-34 are residual networks with skip connections that mitigate vanishing gradient problems, enabling successful training of deeper networks [68].
- VGG is a deep CNN architecture known for its simplicity and ability to capture complex information through layered convolutional layers [69].

In the process of transfer learning for audio genre classification, the iterative optimization and hyperparameter-tuning of pre-trained models play a central role. Typically, this involves leveraging a pre-trained model, such as one trained on a large dataset for a related task, to

expedite convergence and facilitate the extraction of meaningful features from audio data, specifically Mel spectrograms in this context.

A key technique within transfer learning is to freeze the initial layers of the pre-trained model, which are responsible for capturing generic features like edges and textures, and only train the subsequent layers that form the classification head. By doing so, the model can adapt its learned features to the specific nuances of audio genre classification without overfitting to the limited dataset. This strategic approach allows the model to benefit from the knowledge gained during the pre-training phase while tailoring the final layers to the nuances of the target task. The harmonious integration of advanced DL algorithms with well-established architectural foundations in this transfer learning paradigm significantly enhances the robustness and precision of music genre classification.

Optimizer in Model Training

Optimizers play a pivotal role in training and converging ML models for music genre classification. These algorithms determine how the model adjusts its internal parameters during training, optimizing the learning process. Understanding the significance of optimizers is crucial, especially when building upon pre-trained models.

In large-scale training procedures, intricate architectures with numerous parameters, such as those found in ML and DL models, demand careful optimization. Optimizers become essential tools to efficiently navigate the complex parameter space, aiding in model convergence, preventing overfitting, and enhancing overall performance. This becomes particularly critical in the context of transfer learning, where adapting the model's weights based on gradients calculated during training is crucial. It is imperative to choose optimizers thoughtfully, even for a limited number of layers in the Transfer Learning paradigm, especially when dealing with a small dataset. The choice of optimizer becomes a key factor in achieving optimal performance and addressing the challenges posed by limited data.

This approach provides flexibility in optimizer selection, allowing users to parameterize the model training according to their preferences. The following optimizers are integrated into the implementation:

- Adam (Adaptive Moment Estimation): A popular optimizer that adapts the learning rates of each parameter individually, combining the advantages of two other optimizers, RMSprop and Momentum [6].
- Stochastic Gradient Descent (SGD): The classic optimization algorithm, which adjusts model parameters by considering the negative gradient of the loss function concerning each parameter [70].
- RMSprop (Root Mean Square Propagation): An adaptive learning rate optimization algorithm that adjusts the learning rates for each parameter based on the average of recent squared gradients [71].

This implementation offers users the opportunity to parameterize model training with the optimizer of their choice. By providing a range of optimizers, it enables experimentation and hyperparameter-tuning to achieve optimal performance in music genre classification tasks. This customization ensures adaptability to various data characteristics and specific objectives, contributing to the versatility of the proposed machine-learning workflow.

Criterion in Model Training

Loss functions, known as Criterions, form the backbone of training ML models. These functions gauge the dissimilarity between predicted and actual outputs, crucial for guiding the optimization process during learning process.

Given the intricate architecture of pre-trained CNN models utilized in this thesis, the transition to transfer learning with a limited dataset accentuates the significance of selecting appropriate loss functions. The criterion plays a crucial role in quantifying the divergence between predicted and true outputs, ensuring accurate learning in the context of transferring knowledge to a new task with limited data. A judiciously chosen criterion becomes instrumental in facilitating convergence and guarding against overfitting during the refinement of the model with the constraints posed by the limited dataset.

This approach introduces flexibility in the choice of loss functions, allowing users to parameterize model training according to their preferences. The following loss functions are integrated into the implementation:

- Cross Entropy Loss (cross_entropy): Widely adopted for classification tasks, cross-entropy measures the discrepancy between predicted and true class probabilities [6].
- Kullback-Leibler Divergence Loss (kldiv_loss): Evaluates the difference between two probability distributions, often applied in scenarios with probabilistic outputs [72].

This thesis implementation provides users with the flexibility to parameterize model training using the criterion of their preference. Offering a diverse set of loss functions empowers users to tailor the learning process to specific data characteristics and desired objectives. This adaptability ensures experimentation and exploration within the proposed ML workflow, contributing to the versatility of the model training process.

Framework Dilemma: PyTorch vs. TensorFlow

Choosing between PyTorch and TensorFlow for the implementation of this thesis involved a careful consideration of the strengths and weaknesses of both frameworks. Below is a concise comparison highlighting the distinctive features of each framework [74,75]:

TensorFlow stands as a widely acclaimed end-to-end open-source platform for ML, originally developed by Google's Brain team. It replaced Google's DistBelief framework and operates seamlessly across various execution platforms, including CPU, GPU, TPU, and mobile devices. TensorFlow offers robust support, and frequent releases, and is extensively used by global companies like Google, Uber, and Microsoft. Noteworthy components include TensorFlow Lite for edge-based ML.

Advantages	Disadvantages
Strong support and library management, ideal for production environments.	Computation speed lags compared to competitors.

TensorFlow Lite for optimized edge-based ML.	Increased dependency for code execution.
Data visualization tool (TensorBoard) for graphical representation.	Challenges with symbolic loops for indefinite sequences.
Compatibility with Keras for high-level functionality.	Limited GPU support initially.

Table 6 Advantages and Disadvantages of Tensorflow Framework

Introduced in 2016, **PyTorch** prioritizes usability and performance considerations, gaining popularity in the DL research community. Its imperative and Pythonic programming style supports the immediate execution of dynamic tensor computations, GPU acceleration, and automatic differentiation. PyTorch's user-friendly approach, dynamic computational graphs, and Python-centric design make it a go-to choice for researchers and developers alike.

Advantages	**Disadvantages**
Python-centric design for seamless integration.	Limited model serving in production compared to other frameworks.
Easier to learn with a syntax similar to conventional programming languages.	Lack of extensive monitoring and visualization interfaces.
Dynamic computational graphs for runtime adaptability.	Conversion is required for actual application deployment.
Robust community support and organized documentation.	Smaller documentation and developer community compared to TensorFlow.
Efficient debugging using Python debugging tools.	

Table 7 Advantages and Disadvantages of Pytorch Framework

Comparative Analysis:

- **Performance Comparison**: The Figure 7 illustrates the training speed of the two models under the utilization of 32-bit floats. The throughput metrics are expressed as images per second for the AlexNet, VGG-19, ResNet-50, and MobileNet models, tokens per second for the GNMTv2 model, and samples per second for the NCF

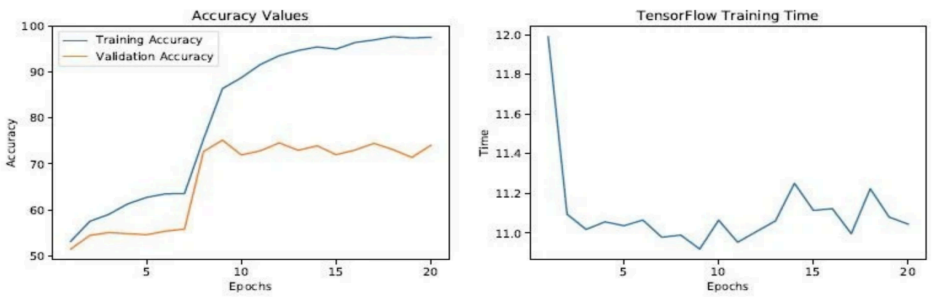
model. PyTorch demonstrates better performance in benchmark tests compared to TensorFlow, attributed to lower overhead [74].

Performance Benchmark: Throughput - Higher is better

Framework	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
TensorFlow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	9631 ± 1.3%	4.8e6 ± 2.9%
PyTorch	1547 ± 316	119 ± 1	212 ± 2	463 ± 17	15512 ± 4.8%	5.4e6 ± 3.4%

Figure 7 Performance Comparison between Pytorch and Tensorflow [74]

- **Accuracy**: Both frameworks (Table 8) yield similar accuracies, showcasing consistency in implementing neural networks accurately [75]
- **Training Time and Memory Usage**: TensorFlow exhibits lower memory usage but longer training times compared to PyTorch, with variations during model training (Table 8).

Tensorflow	

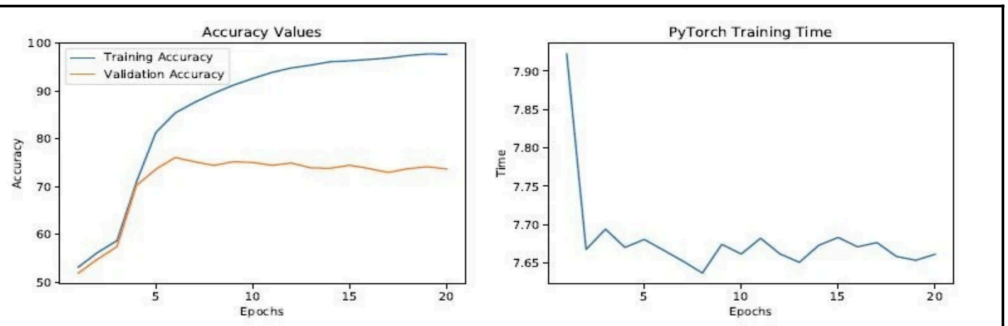
Pytorch	

Table 8 Accuracy and Time Comparison between Pythorch and Tensorflow

- **Ease of Use**: PyTorch's object-oriented style and straightforward data handling contribute to ease of use, while TensorFlow's low-level approach allows for greater customization.

The decision to opt for PyTorch in this thesis was driven by its user-friendly design, dynamic computational graphs, and imperative Pythonic programming style. The framework's active community support and integration capabilities align well with the research objectives, making it a suitable choice for building a scalable and reproducible machine-learning workflow for music genre classification.

3.3.4 Evaluation Metrics

Ensuring the robust evaluation of a music genre classification model is key to understanding its performance comprehensively. Precision, recall, and F1-score emerge as essential metrics, offering insights into the system's accuracy, sensitivity, and overall effectiveness in the multiclass classification context of music genre identification.

Refined Evaluation Metrics

- **Precision** signifies the percentage of accurately predicted positive instances among all instances predicted as positive. In the case of music genre classification, precision measures the algorithm's accuracy in identifying specific genres, minimizing false positives. This metric becomes crucial for assessing the reliability of predicted genres, preventing misclassification [76].
- **Recall**, also referred to as sensitivity or true positive rate, gauges the ratio of correctly predicted positive instances to the total number of actual positive instances. In music genre classification, recall showcases the system's ability to recognize relevant samples, ensuring no genre is overlooked. High recall indicates effective detection of specific genres while minimizing false negatives [77].
- The **F1-score**, a balanced measurement, represents the harmonic mean of precision and recall. It provides a comprehensive assessment of the classifier's overall effectiveness in classifying musical genres by considering both false positives and false negatives.

Its impartiality towards precision or recall makes it particularly valuable for datasets with uneven class distributions [10].

Significance in Music Genre Classification

In addressing genre imbalances and the exploration for high accuracy across various genres, Aucouturier and Pachet (2002) highlight the importance of balanced evaluation measures such as precision and recall [78]. Emphasizing precision and recall, Lee and Slaney (2009) stress their relevance when evaluating automatic music genre classification systems across multiple genres [79]. The F1-score, encapsulating the harmonic mean of recall and precision, emerges as a suitable measure for overall performance in classifying musical genres. Sokolova and Lapalme (2009) endorse the F1-score's significance as a balanced evaluation metric, particularly in handling unbalanced datasets and ensuring accurate categorization across numerous classes [10].

3.4 Machine Learning Operations: Streamlining Efficiency in Music Genre Classification

Within this section, we explore the vital domain of MLOps, underscoring their pivotal role in optimizing efficiency for music genre classification. We examine the application of MLOps tools to enhance and implement our ML pipeline. These tools contribute to the seamless orchestration and improved performance of our music genre classification model to achieve a scalable and reproducible DL Pipeline.

3.4.1 Data Versioning and Management

Efficient data versioning and administration play a pivotal role in upholding reproducibility and fostering collaboration within the dynamic realm of ML research. This subsection delves into the application of Data Version Control (DVC) together with Amazon Simple Storage Service (S3) for managing the GTZAN dataset. This use case revolves around the classification of musical genres, emphasizing the significance of robust data handling in ML endeavors.

At the core of connecting data-intensive projects with conventional version control systems is Data Version Control (DVC). By seamlessly integrating with Git, DVC establishes a unified platform for versioning both data and code. This adaptability makes it suitable for handling a variety of datasets, including audio files and images such as spectrograms. The decision to employ DVC for GTZAN dataset management stems from its compatibility with Git, providing a comprehensive versioning and collaboration solution. The collaborative features, traceability, and seamless scalability of DVC prove essential for handling large and diverse datasets in research settings. Integration with Amazon S3 offers researchers access to a scalable object storage service, ensuring effortless scalability, cost-effective data administration, and reliable data durability. Storing the GTZAN dataset in S3 safeguards the integrity and traceability of data over time.

Now, with DVC, metadata about the data is versioned alongside the source code, while the original data files are intentionally added to .gitignore (a configuration file used in Git to

specify files and directories that should be ignored when tracking changes in a repository), ensuring a clean version control history. The file snippet depicts the data stored in the remote s3 repository. The hash value of the added genres_original files (93c37d...) determines the cache path, as evidenced in the cache directory. If you inspect the genres_original.dvc, you'll find metadata specifying the hash, size, number of files, and path.

```
outs:
- md5: 93c37d996f5645354ebb96a940624b50.dir
  size: 1323989016
  nfiles: 1000
  path: genres_original
```

Table 9 genre_original.dvc yml file

This thorough approach to data management ensures not only versioning and traceability but also optimizes storage and transfer processes, promoting collaboration, and adherence to GitOps principles in data science projects. The lightweight and open-source nature of DVC eliminates the need for complex databases, fostering consistency through stable file names and enabling efficient data management. Adopting DVC for data versioning and management proves multifaceted, aligning with advanced CI/CD tools and Git workflows, thereby enriching collaboration and knowledge exchange in the research community.

3.4.2 Experiment Management and Model Lifecycle Tools

In this section, we investigate Experiment Management and Model Lifecycle Tools, introducing an approach to elevate the efficiency of our music genre classification methodologies. This segment intricately unveils three distinct ML Pipelines tailored for music genre classification. What distinguishes this exploration is the practical application through command-line interfaces (CLI) and Data Version Control (DVC).

Machine Learning Pipelines for Music Genre Classification

In pursuit of a comprehensive and scalable approach to music genre classification, three distinct ML pipelines have been developed to address various steps of the classification process. Each pipeline serves a unique purpose, collectively contributing to a robust and reproducible workflow.

1. **Feature-Based Music Genre Classification Pipeline**

This pipeline (Figure 8) focuses on training music genre classifiers using pre-extracted features, encompassing model selection, criterion optimization, and training orchestration. Its flexibility extends to employ diverse model architectures, including ResNet18, ResNet32, AlexNet, VGG, and DenseNet121, and it accommodates various optimization criteria such as Cross-Entropy, KL Divergence, and Smooth Loss, along with optimizers like Adam, SGD, and RMSprop.

The pipeline initiates by dynamically selecting the specified model architecture, criterion, and optimizer based on user inputs. Data loading and transformation are executed using torch and

torchvision, transforming raw images into a format suitable for model training. Subsequently, the dataset undergoes splitting into training, testing, and validation sets, ensuring robust evaluation.

The core of the pipeline involves training the selected model on the training dataset for a predefined number of epochs, guided by the chosen criterion and optimizer. Notably, in the context of transfer learning, this process entails training the last layer of the pre-trained architecture. Guided by the chosen criterion and optimizer, this targeted training approach optimizes the model's performance for the specific task at hand. The option to save the trained model provides flexibility for future use, while the calculation of evaluation scores, including F1-score, precision, and recall, furnishes a comprehensive understanding.

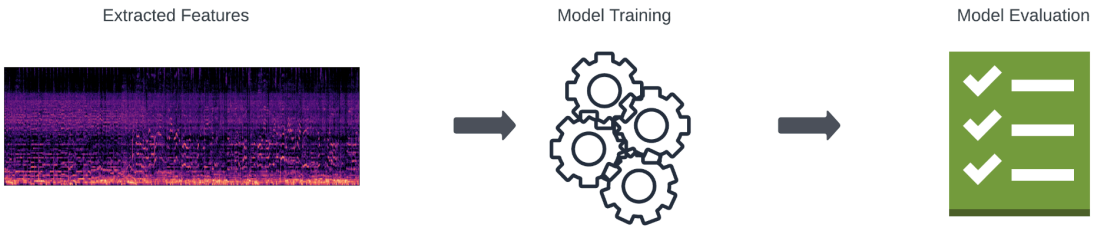


Figure 8 Feature-Based Music Genre Classification Pipeline

2. Audio Feature Extraction Pipeline

The second pipeline (Figure 9) tackles the essential task of extracting features from audio files. This process involves key parameters, including the audio file path, the destination path for storing features, and specifications for audio preprocessing and feature extraction. The pipeline begins with audio preprocessing, employing techniques defined by the audio preprocess factory to ensure compatibility of the audio data. Feature extraction, managed by the feature extraction factory, distills meaningful features like mel spectrograms from the preprocessed audio. This process includes labeling and saving, assigning genres to the audio features and storing them for subsequent model training.

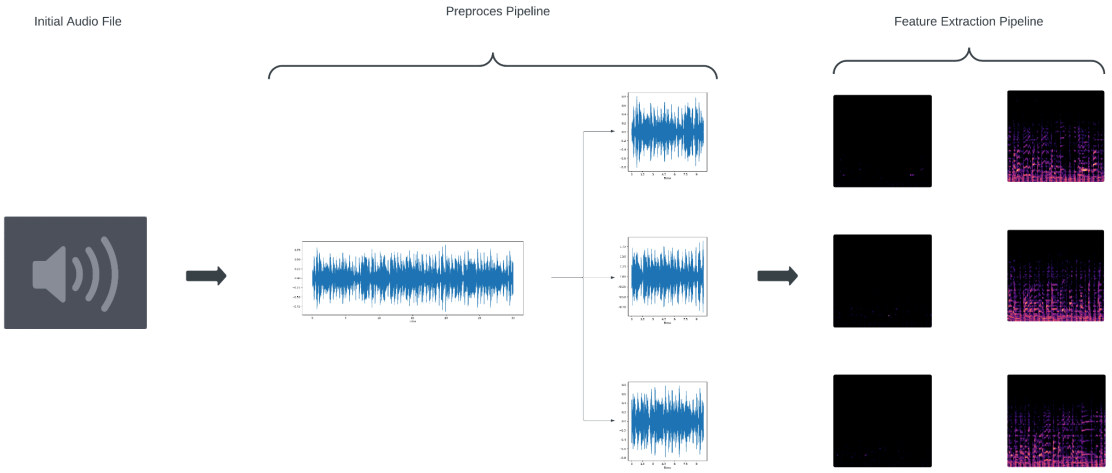


Figure 9 Audio Feature Extraction Pipeline

3. Audio-Based Music Genre Classification Pipeline

The third pipeline (Figure 10) seamlessly integrates the functionalities of the preceding two pipelines, presenting a unified approach to training models using audio files. This integration ensures a comprehensive workflow by merging the strengths of feature-based classification and audio feature extraction.

Commencing with the loading of pre-trained model architectures, akin to the first pipeline, the process advances to create features from audio files using the second pipeline. This involves a comprehensive procedure of audio preprocessing and feature extraction, establishing the groundwork for subsequent model training.

The final stages of this integrated pipeline encapsulate model training, evaluation, and optional saving functionalities. Evaluation metrics, including F1-score, recall, and precision, offer a nuanced assessment of the model's performance, validating the effectiveness of the integrated approach.

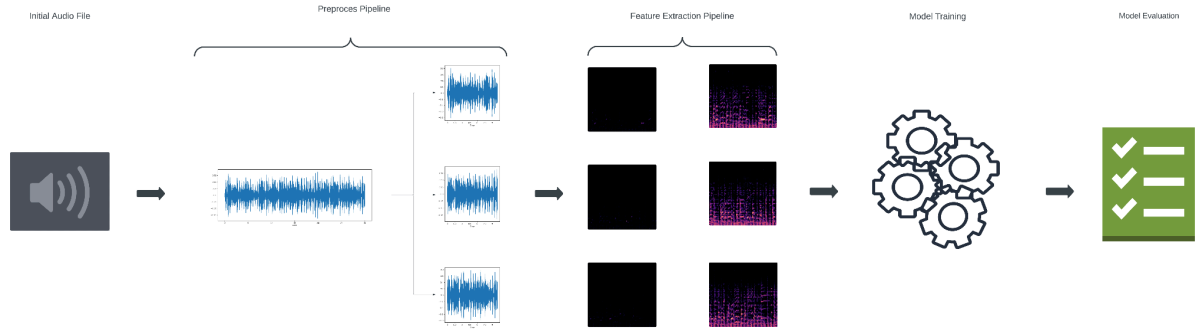


Figure 10 Audio-Based Music Genre Classification Pipeline

In unison, these precisely designed pipelines not only offer flexibility in model selection, feature extraction, and training but also contribute significantly to the creation of a scalable and reproducible ML workflow tailored for music genre classification.

Machine Learning Pipelines using Command Line Interface

In the territory of ML workflows, the Command-Line Interface (CLI) functions as a potent instrument for coordinating and carrying out intricate procedures. In the context of this thesis, the Click framework is utilized to construct a CLI that simplifies the implementation of music genre classification pipelines. In this scenario, CLI denotes a text-based interface enabling users to engage with the system by issuing commands.

A **CLI** is a text-based interface that allows users to interact with software and perform operations by entering commands into a terminal or console. CLIs offer a direct and efficient means of controlling applications and processes, making them a valuable tool for automating tasks and managing complex workflows [80].

Click is a Python package that simplifies the process of creating CLIs. It provides a clean and concise syntax for defining command-line options, arguments, and commands, making it easy to build robust and user-friendly interfaces [81]. The Click framework is particularly

well-suited for this approach due to its simplicity, extensibility, and seamless integration with Python.

Advantages of CLI and Click:

- Streamlined Execution: The CLI commands facilitate a streamlined execution of complex tasks, allowing researchers and practitioners to trigger the music genre classification pipelines with a single command. This simplicity enhances usability and reduces the barriers to entry.
- Reproducibility: By encapsulating the entire pipeline within CLI commands, reproducibility is inherently built into the workflow. The defined options and functionalities ensure that the experiments and evaluations can be replicated consistently.
- User-Friendly Interface: Click provides an intuitive and user-friendly interface for the CLI commands. The clean syntax and well-defined options make it easy for users to understand and leverage the capabilities of the music genre classification system.
- Flexibility and Extensibility: The modular nature of the CLI commands allows for extensibility and flexibility. Additional commands can be added to the CLI to accommodate future enhancements or variations in the music genre classification pipeline.

Implementation of CLI: Training CLI Command

In the context of music genre classification, a specific CLI command, *train_using_original_audios*, is developed using the Click framework. This command encapsulates the training pipeline for audio-based genre classification. Let's delve into the key components:

- Options: Various options are defined using @click.option decorators. These options include the model choice (--model), criterion (--criterion), optimizer (--optimizer), checkpoints path (--checkpoints_path), path to save featured images for training (--save_images_path), audio paths (--audio_paths), number of training epochs (--num_epoch), and others.
- Functionality: Integrated seamlessly within the comprehensive ML pipeline, this command plays a crucial role by invoking the `ml_entrypoints.train_tl_model_audio` function. This function, situated in the broader context of the extended pipeline, adeptly receives the specified options and takes charge of orchestrating the intricate training process for the music genre classification model. This seamless connection ensures that the training task operates harmoniously within the overarching pipeline, contributing to the model's evolution in a cohesive and integrated manner.
- Result Handling: The metrics generated during the training process are saved in a designated file (--path_to_save_metric). This ensures that the evaluation results are easily accessible and reproducible.

The incorporation of the Click framework for CLI commands in this thesis significantly contributes to the efficiency, reproducibility, and user-friendliness of the music genre classification workflows. CLI commands, driven by Click, serve as a pivotal interface for researchers and practitioners to interact with and explore the intricacies of ML processes.

Machine Learning Pipelines using DVC Pipelines

In the domain of music genre classification, the inclusion of DVC pipelines stands out as a fundamental element, ensuring the effectiveness, reproducibility, and structured arrangement of the ML process. This section delves into the reasoning for incorporating DVC pipelines within the framework of the thesis, providing insights into their practicality and advantages.

Advantages of DVC Pipelines [82]:

- Reproducibility: DVC pipelines offer a robust mechanism for capturing, versioning, and reproducing the entire workflow. This ensures that every stage of the music genre classification process, from audio preprocessing to model training, can be replicated precisely, contributing to the credibility of research findings.
- Structured Workflow: The structured nature of DVC pipelines allows for a clear delineation of each stage in the classification process. This organization not only enhances the readability of the workflow but also facilitates seamless coordination between different stages.
- Version Control Integration: DVC integrates seamlessly with Git, providing version control for both code and data. This integration ensures that changes in the codebase or dataset are systematically tracked, enabling precise control over the evolution of the ML models.
- Dependency Tracking: With DVC, dependencies between different stages of the workflow are explicitly defined. This ensures that any change in the input data or code triggers the necessary downstream steps, maintaining consistency in the workflow.

Implementation of DVC Pipelines: A Practical Example

The implementation of DVC pipelines in this thesis is exemplified through the provided code snippet. This snippet outlines the creation of a DVC pipeline with two distinct stages: feature_extraction and train.

- Feature Extraction Stage: This stage entails running a Python script to generate image features from audio files, aligning with the Audio Feature Extraction Pipeline discussed in the previous section. The dependencies, including the Python file serving as the pipeline entry point and the directory containing audio files, as well as the outputs, specifying the path for storing the features, are meticulously defined. This guarantees that modifications in the input data prompt the execution of this stage.
- Training Stage: The train stage employs the image features generated in the previous stage (Image_data) to train a ML model. This stage corresponds to the training pipeline from features mentioned in the previous section. The model, specified as resnet18, utilizes the specified criterion (cross_entropy), and optimizer (sdg). The checkpoints from the training process are stored in the checkpoints directory.

```
stages:
  feature_extraction:
    cmd: python3 __main__.py create-image-features-from-audio --path_with_audios_dir
Data/genres_original --path_to_image Image_data
    deps:
      - __main__.py
      - Data/genres_original
    outs:
      - Image_data
  train:
    cmd: python3 __main__.py train-using-image-features --model resnet18 --criterion
cross_entropy --optimizer sdg --checkpoints_path checkpoints --images_path Image_data
    deps:
      - __main__.py
      - Image_data
    outs:
      - checkpoints
```

Table 10 Implementation of a DVC Pipeline

The implemented pipeline, facilitated by the DVC framework, is the Audio-Based Music Genre Classification Pipeline as detailed in the preceding sub-section. This pipeline seamlessly integrates the feature extraction pipeline and the model training process, orchestrating a cohesive workflow for music genre classification.

By structuring the workflow using DVC pipelines, this thesis ensures a systematic and reproducible approach to music genre classification. The clear delineation of stages, version control integration, and dependency tracking collectively contribute to a robust and transparent ML workflow.

Experiment Management using MLflow

MLflow, designed for end-to-end ML lifecycle management, is rooted in the scientific principles of reproducibility and collaboration. It enables researchers and data scientists to systematically conduct, monitor, and compare experiments, fostering a scientific methodology that underpins advancements in ML [87]. The platform encapsulates four core components: Tracking, Projects, Models, and Registry, collectively offering a unified environment for seamless experimentation and deployment [88].

MLflow introduces a structured approach to experiment management. It allows the logging and tracking of metrics, parameters, and artifacts during model training, creating an accessible record of the experiment's evolution [89]. This not only facilitates a deeper understanding of model performance but also ensures reproducibility by capturing the entire context of an experiment, from hyperparameters to dependencies [90].

MLflow: A Comprehensive Evaluation in Contrast to Comet and Weights & Biases

In the field of experiment management and model lifecycle tools, MLflow emerges as a robust and versatile solution with distinctive features and advantages. MLflow distinguishes itself through its open-source nature and extensive customization capabilities. Unlike commercial platforms like Comet and Weights & Biases, MLflow's open-source status provides a transparent and adaptable framework that caters to a diverse range of machine-learning workflows [88]. Its language-agnostic and framework-agnostic design ensures flexibility and integration with popular frameworks, making it an inclusive choice for diverse ML projects [87].

MLflow's automatic logging functionality sets it apart in terms of efficiency. While Comet and Weights & Biases offer comprehensive tracking capabilities, MLflow automates the logging of metrics, parameters, and models without the need for explicit log statements [89]. This streamlined logging process facilitates seamless integration into existing codebases, requiring only minimal lines of code to initiate tracking [90].

MLflow's thriving community and widespread adoption in the industry contribute to its resilience and continuous improvement [91]. The platform enables teams of data scientists to share a single dashboard, and view, and compare results from numerous experiments carried out by different users [92]. However, it's essential to note that MLflow's collaboration features may not be as extensive as those in commercial platforms like Comet and Weights & Biases [32,33].

One of MLflow's major strengths lies in its holistic approach to machine learning lifecycle management. It goes beyond experiment tracking and offers tools for storing, versioning, packaging code into reproducible runs, and deploying models. This comprehensive suite of functionalities positions MLflow as a one-stop solution for the end-to-end ML lifecycle.

While MLflow boasts several advantages, it's crucial to acknowledge certain considerations. MLflow, being an open-source solution, requires organizations to maintain servers and infrastructure, posing potential challenges for smaller entities [87]. Additionally, its security features may necessitate additional configuration for secure handling of sensitive data, impacting the ease of sharing experiment results [87]. Collaborative features, while present, might not match the extensive offerings of commercial alternatives.

Implementation of MLflow in Music Genre Classification

In the context of music genre classification, MLflow emerges as a pivotal tool for monitoring and managing the training process. Given the intricacies of training models on diverse audio datasets, MLflow provides a unified interface to monitor losses, evaluate model performance, and systematically compare the efficacy of different algorithms and hyperparameters [91]. This becomes particularly crucial when dealing with complex pipelines involving feature extraction, model training, and evaluation.

Implementing music genre classification pipelines, MLflow is seamlessly integrated to monitor and log essential metrics, emphasizing loss values. The platform's ability to organize and version experiments aids in precisely tracking the evolution of models and facilitates the selection of the most performant configurations. This systematic logging of metrics ensures that each experiment is documented comprehensively, fostering transparency and reproducibility.

Beyond experiment tracking, MLflow plays a pivotal role in the broader spectrum of model lifecycle management. It offers a centralized repository for managing different versions of models, allowing for easy retrieval, deployment, and sharing across diverse environments. This streamlined lifecycle management is particularly beneficial in the dynamic landscape of music genre classification, where models may undergo frequent updates and improvements.

MLflow stands as a scientifically grounded and versatile tool in the complex landscape of ML experimentation. Its integration into the music genre classification pipelines provides a structured environment for experiment tracking and lifecycle management, aligning with the rigorous demands of scientific research. By focusing on model losses, MLflow becomes an invaluable companion in the quest for optimizing and understanding the intricacies of music genre classification models.

In the domain of experiment tracking and metrics visualization, MLflow proves to be an invaluable tool. Two key figures further illustrate the utility and functionality of MLflow within the context of this thesis implementation.

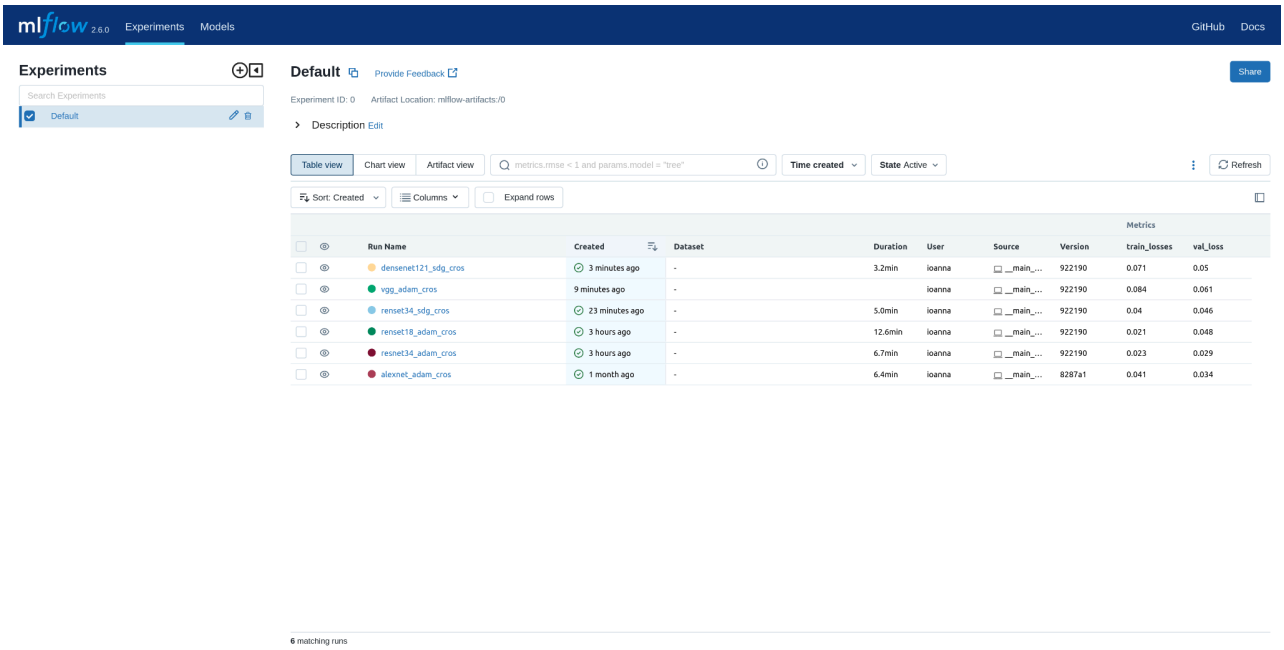


Figure 11 MLFlow experimental tracking

Figure 11 provides an overview of a sample list comprising models and experiments conducted throughout this thesis implementation. This Figure captures essential information about each experiment, serving as a comprehensive reference for analysis. Key details include the experiment's name, which acts as a unique identifier, the timestamp of its creation, the duration of the experiment, the user responsible for triggering the experiment, and crucial monitoring metrics such as training and validation losses. Additionally, information about the experiment's version and the source file used for execution is presented, contributing to a comprehensive understanding of each experiment's context and characteristics.

Figure 12 showcases a specific page within the MLflow user interface, offering an in-depth investigation into a particular run. This page is instrumental in visualizing the metrics logged during the training process of a specific experiment. In the depicted example, the metrics plotted pertain to the training and evaluation losses incurred during the training of the ResNet18 model. Noteworthy details include the utilization of the Adam optimizer and

Cross-Entropy criterion across 10 epochs. This granular view into individual runs allows for a detailed examination of the training dynamics, aiding in performance evaluation and model optimization.

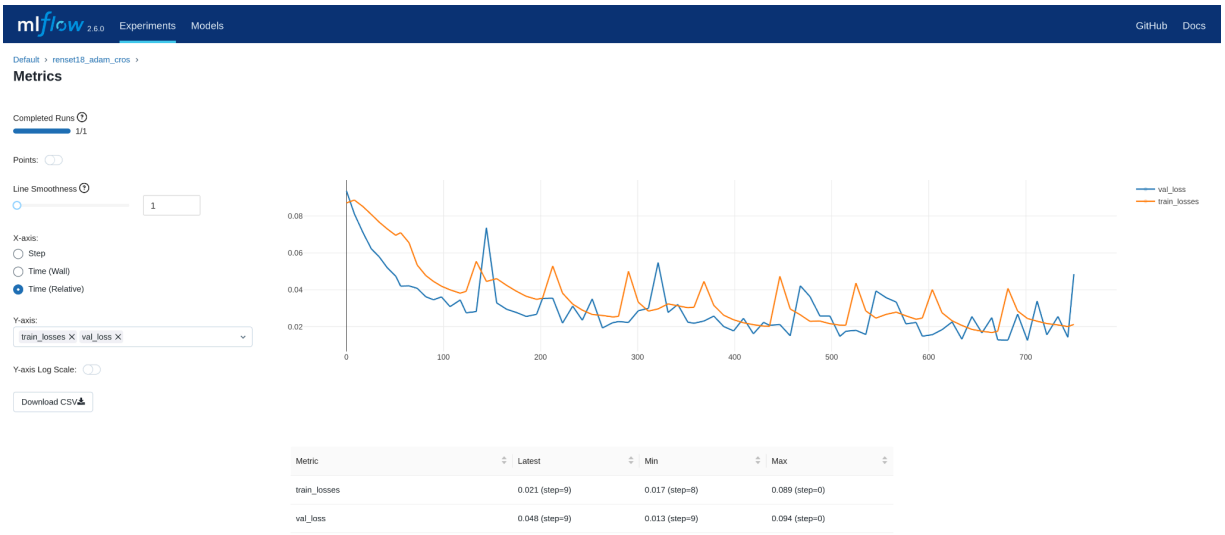


Figure 12 MLFlow experimental tracking, metrics monitoring

Together, these Figures underscore the significance of MLflow in experiment tracking and metrics visualization, providing a comprehensive and user-friendly interface for monitoring and analyzing the intricacies of DL model training.

- **Optimizing Hyperparameters in Machine Learning**

The optimization of hyperparameters stands as a pivotal undertaking in the domain of ML, significantly influencing model performance. This pursuit has undergone a notable evolution from traditional methodologies to more sophisticated techniques, with Optuna emerging as a forefront leader in hyperparameter optimization frameworks. Distinguished by a define-by-run API, efficient search and pruning strategies, and a versatile architecture, Optuna represents a pioneering effort in advancing the field [93].

Optimizing a ML model's hyperparameters is crucial for steering its learning trajectory and maximizing overall performance. This tailored calibration, specific to each model and dataset, focuses on identifying configurations that enhance the model's metrics, ultimately improving its predictive efficacy.

Traditional optimization approaches, exemplified by grid search and random search, while conceptually straightforward, exhibit limitations in scalability. As datasets and hyperparameter spaces burgeon, these methods face challenges in efficiency and efficacy, struggling to navigate expansive solution spaces efficiently. Heuristic methods, such as genetic algorithms and simulated annealing, offer promising alternatives to traditional optimization approaches. These techniques leverage intelligent search strategies to efficiently explore complex solution spaces, providing scalability advantages and enhanced adaptability in the face of large datasets and intricate hyperparameter configurations.

Optuna: Hyperparameter Optimization Framework

Optuna [46] heralds a paradigmatic shift in hyperparameter optimization through the incorporation of cutting-edge algorithms. The framework facilitates the implementation of diverse sampling and pruning strategies. Sampling strategies, including Tree-Structured Parzen Estimator (TPE) and Non-dominated sorting genetic algorithm II (NSGA-II), pivot towards the judicious selection of optimal parameter combinations. Pruning strategies, exemplified by Successive Halving, introduce the concept of early stopping, strategically discarding suboptimal trials. Operationalizing Optuna involves the definition of an objective function encapsulating the model's logic, training, and evaluation. The Study object within Optuna orchestrates the iterative optimization process, sequentially executing trials to discern the most fitting hyperparameters. Visualization tools embedded within Optuna offer insights into the optimization history, parameter importances, and intermediate values.

Implement Optuna in this thesis

In the context of a thesis, Optuna proves to be an invaluable asset. Its versatility allows for the dynamic definition of hyperparameter search spaces, accommodating the unique demands of different machine-learning models. The framework's seamless integration with optimization parameters such as batch size, optimizers, and criteria aligns with the varied requirements of thesis research. Whether the focus is on scalable distributed computing or lightweight experiments conducted through an interactive interface, Optuna's design principles and optimization algorithms make it an ideal companion for a thesis centered on hyperparameter tuning and model optimization.

- **Docker in the Context of Machine Learning Pipelines**

Docker has emerged as a pivotal tool in modern software development and deployment, offering a containerization platform that enhances the efficiency, consistency, and reproducibility of applications across diverse environments [83]. In ML workflows, Docker containers encapsulate an entire runtime environment, including dependencies, libraries, and configurations, ensuring seamless portability and execution of applications.

Docker's utility lies in its ability to encapsulate applications and their dependencies within isolated containers. This ensures that the software behaves consistently across various computing environments, mitigating the notorious problem of "it works on my machine." Docker eradicates compatibility issues by packaging the application along with its runtime dependencies, streamlining deployment and facilitating collaboration among researchers, developers, and data scientists [84].

Docker Implementation in Music Genre Classification Pipelines:

In the context of this thesis, Docker serves as a foundational element for managing the intricate processes involved in music genre classification pipelines. The provided Dockerfile employs a multi-stage building approach, starting with a base image and creating specialized stages for distinct functionalities.

The base image is established with essential tools and dependencies, providing a standardized foundation for subsequent stages. Additional stages, such as 'awscli' and 'dl_pipelines,' extend the functionality by incorporating tools like the AWS CLI and DVC. The 'awscli' stage retrieves and installs the AWS CLI, while the 'dl_pipelines' stage configures DVC and pulls essential files, ensuring a consistent environment.

The use of Docker simplifies the deployment and management of these pipelines. It encapsulates the entire application stack, guaranteeing that the environment required for execution is reproducible regardless of the underlying infrastructure. Moreover, Docker's containerization facilitates the isolation of dependencies, reducing conflicts and providing a standardized environment for pipeline execution.

In essence, Docker acts as an enabler for reproducibility and consistency in the execution of music genre classification pipelines. It encapsulates the complexities of dependencies and configurations, allowing for seamless deployment and collaboration, making it a valuable asset in the landscape of ML workflows [83,84].

- **Airflow in Machine Learning Workflow Orchestration**

Apache Airflow, a powerful open-source orchestration tool, plays a pivotal role in managing the intricacies of ML workflows. It provides a platform for defining, scheduling, and executing workflows, offering the flexibility to customize and scale complex data pipelines. The advantages of Airflow lie in its ability to manage dependencies, monitor workflow progress, and support dynamic scheduling, making it an ideal choice for orchestrating ML pipelines [43].

This open-source Python-based tool is adept at designing, scheduling, and monitoring data pipelines. Utilizing DAGs, Airflow provides a clear visualization of task relationships and execution orders. Airflow is particularly suited for batch pipelines, making it an excellent choice for music genre classification workflows. Its versatility shines in end-to-end ML pipelines, ETL/ELT data integration, automated report generation, and various DevOps tasks. With a focus on orchestrating Big Data workflows, Airflow ensures transparency and compatibility with major data platforms.

The architecture revolves around DAGs, allowing users to design tasks and relationships as Python scripts. The metadata database, scheduler, executor, workers, and web server collectively streamline orchestration and monitoring. Python scripting enhances accessibility, and the "workflow as code" philosophy offers unparalleled customization.

In navigating the landscape of workflow orchestration tools for music genre classification pipelines, various options offer specific strengths. Prefect, Metaflow, Kedro, ZenML, Kubeflow, Flyte, Airflow, MLRun, Argo, and Luigi each cater to distinct needs. However, for the specific requirements of orchestrating intricate music genre classification pipelines, Apache Airflow stands out. Its seamless integration with Python aligns with the existing workflow, providing a visual DAG-based representation for intuitive orchestration. Despite a learning curve, Airflow's versatility and compatibility within the Python ecosystem make it the optimal choice for enhancing transparency, maintainability, and orchestration capabilities.

In the context of this thesis, Airflow serves as the backbone for orchestrating music genre classification pipelines. Three distinct workflows are crafted to handle the diverse tasks involved in model training and feature extraction from audio data. Leveraging the flexibility of

Airflow, each workflow is meticulously designed to encapsulate specific stages, ensuring modularity and maintainability.

Implementation Approaches:

1. **Python Operator Approach:**

The initial implementation employs Python operators within Airflow, executing CLI commands for pipeline execution. This approach offers flexibility but requires careful management of package dependencies and data from DVC.

Click is a Python package that aids in the creation of command-line interfaces (CLIs) with minimal code, making it a valuable tool for managing and executing commands in a Python script. In the Python Operator Approach, Click commands are employed to encapsulate functionality into CLI commands, allowing for the easy execution of Python scripts within the Airflow pipeline. This approach enhances readability and maintainability, as the script logic is encapsulated within modular commands, and these commands can be easily triggered using the Python Operator.

Furthermore, by incorporating DVC, a version control system for ML projects, into the Python Operator Approach, the implementation ensures proper management of package dependencies and data. DVC enables the versioning and sharing of datasets, ensuring reproducibility in ML workflows. Within the Python Operator, Click commands can be configured to interact with DVC commands, facilitating tasks such as data versioning, retrieval, and storage. This integration streamlines the coordination of data and dependencies, enhancing the robustness and traceability of the ML pipeline.

2. **Docker Operator Approach:**

The Docker operator is harnessed for workflow execution, utilizing pre-built Docker images for corresponding CLI command execution. This approach simplifies dependency and data management during the image-building process. However, the workflows execute on the machine running Docker, posing challenges when steps demand diverse hardware requirements [85].

In this particular implementation, I adopted the Docker operator approach. Firstly, a Docker image was constructed based on a multi-stage Dockerfile. The implementation leverages a multi-stage Dockerfile for constructing Docker images in the Docker Operator Approach. This approach offers several benefits, including efficient dependency management, reduced image size, enhanced security, and isolation of environments. The initial stage, based on the python3.8 image, installs essential dependencies, while subsequent stages focus on specific tasks such as AWS CLI dependencies, DVC, and project requirements. By discarding unnecessary components in the final stage, the resulting Docker image is optimized, ensuring faster deployment, reduced storage requirements, and improved security. The isolation of environments in each stage contributes to a cleaner and more predictable build process, aligning with the efficiency and resource utilization goals of orchestrating ML workflows within Apache Airflow.

As illustrated in the Figure 13 below, this Docker image is employed in the Dockerized approach using Apache Airflow. AWS credentials are securely passed through environmental variables, facilitating the execution of the training command (Table 11):

```
python3 __main__.py train-using-image-features --model resnet18
--criterion cross_entropy --optimizer sdg --checkpoints_path
checkpoints --images_path Data/images_original
```

Table 11 Training command in python

This training task is scheduled to run daily for testing purposes, enabling the continual evaluation of the implementation's performance. Scheduling the training task to run daily is a strategic decision aimed at maintaining a dynamic and adaptive ML model. This frequent cadence enables continuous monitoring and real-time evaluation, ensuring the model's relevance and effectiveness in capturing evolving data patterns. The daily execution facilitates timely detection of anomalies, supports iterative model improvement through parameter tuning and experimentation, and aligns with project-specific requirements and business objectives.

While the daily execution proves beneficial for many scenarios, it's essential to acknowledge that in some real-world applications, the frequency of model training may not need to occur on a daily basis. Depending on the nature of the project and its requirements, the optimal time period for execution might be adjusted to align with specific needs—whether those involve capturing longer-term trends, accommodating periodic data updates, or addressing computational resource constraints. Striking a balance and tailoring the execution frequency to the unique dynamics of the project ensures a judicious use of resources while meeting the specific goals and operational demands of the ML workflow.

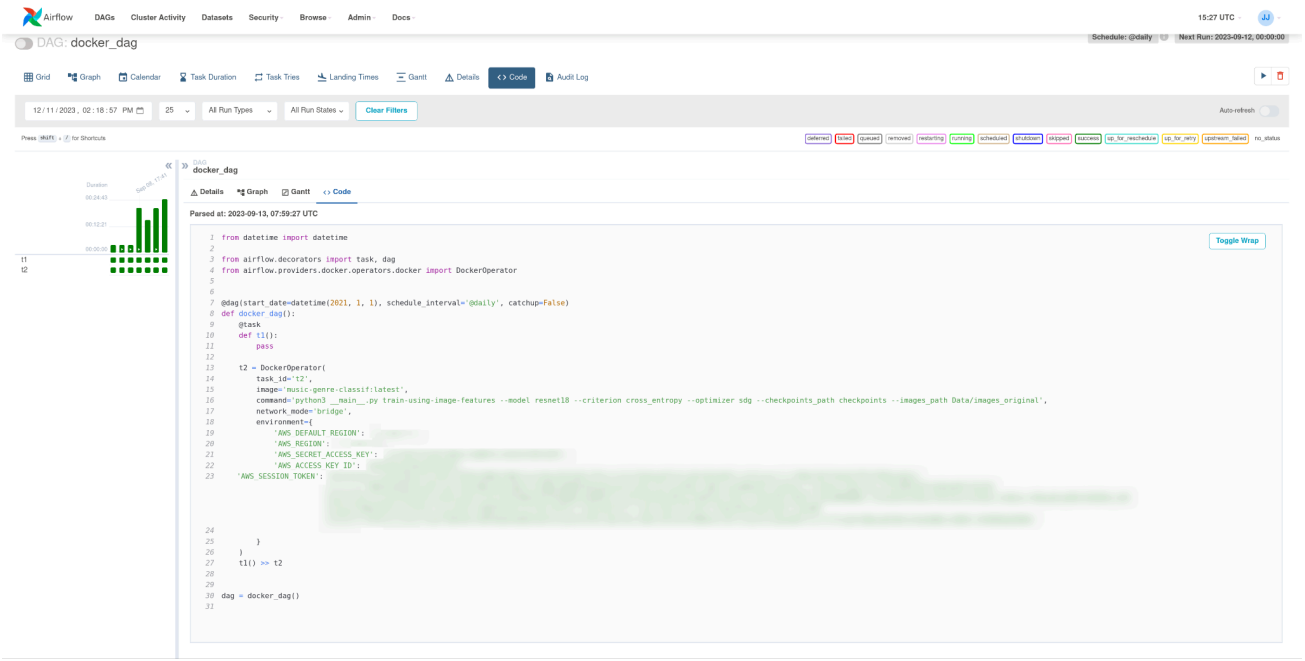


Figure 13 Airflow UI presents the training DAG

Furthermore, the implementation allows for the execution of multiple commands within the same deliverable. For instance, the following code snippet demonstrates the addition of the trained model to DVC and its subsequent versioning through Git:

bash -c 'dvc add checkpoints/resnet18_checkpoint.pt && dvc push && git add checkpoints/resnet18_checkpoint.pt.dvc'

This approach facilitates monitoring of model versioning and ensures robustness in tracking the evolution of the trained model over time.

3. **EC2 Operator Approach:**

To address challenges associated with hardware dependencies, the EC2 operator is introduced as a pivotal component in the workflow extension. This operator extends workflows by seamlessly integrating steps to create, run, and terminate infrastructure on Amazon EC2. Particularly in scenarios where preprocessing, feature extraction, and model training steps necessitate varying hardware resources (CPU, GPU), employing the corresponding hardware for each task emerges as a more efficient and cost-effective strategy [86].

Utilizing the same DL pipeline as employed in the Dockerized approach, this EC2 operator approach involves three key steps, as illustrated in Figure 14 below. The initial step involves the setup of the requisite infrastructure, where an EC2 machine is configured with a specified number of cores and memory tailored to support the DL pipeline. Subsequent to infrastructure creation, the execution of the DL pipeline is initiated. Once the execution concludes, the infrastructure is promptly terminated. This approach proves beneficial as it ensures the utilization of necessary resources exclusively during pipeline execution, promptly followed by their deletion.

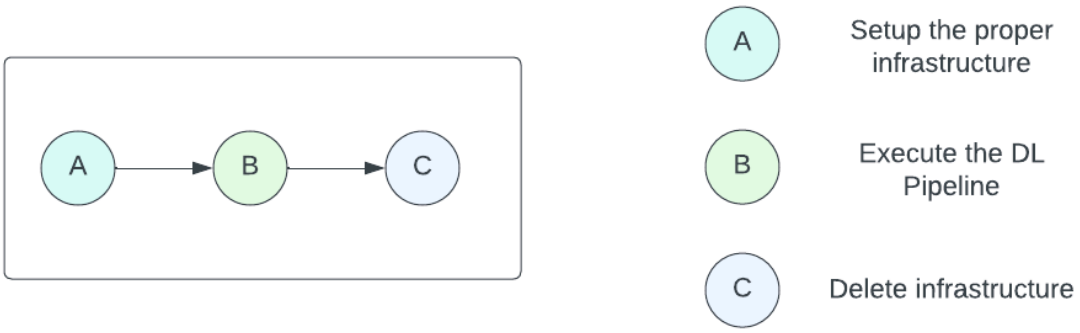


Figure 14 Basic workflow logic using EC2 Operator in Airflow

For further optimization, if the DL pipeline is splitted into discrete steps, such as preprocessing, feature extraction, and training (as illustrated in Figure 15), distinct resources can be allocated to maximize cost efficiency. The preprocessing step, involving lightweight tasks like audio conversion and stemming, contrasts with the heavier processes of feature extraction and training. Allocating different infrastructure for each step becomes more advantageous in this context, optimizing execution time and minimizing costs.

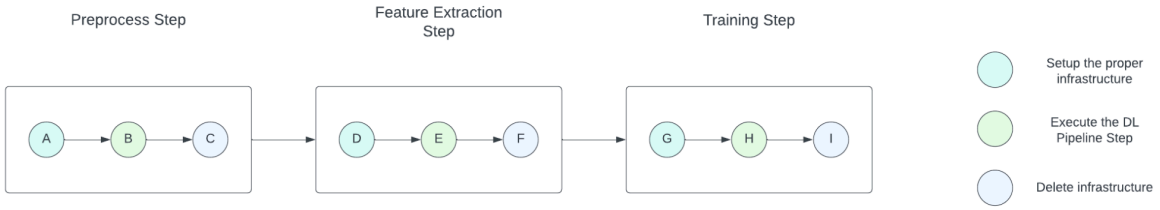


Figure 15 Advanced workflow logic using EC2 Operator in Airflow

In the domain of ML workflows, Apache Airflow stands out as a robust orchestrator, providing the flexibility required for tailoring complex pipelines to specific needs. Leveraging the Python, Docker, and EC2 operators within Airflow opens the door to a multitude of use cases.

For instance, consider a scenario where a data science team collaboratively develops and maintains a ML model. The Python Operator Approach allows for the seamless integration of custom Python scripts, enabling team members to contribute new pre-processing or feature engineering steps easily. This fosters collaboration and adaptability within the team, as each member can contribute to the pipeline without disrupting the overall workflow.

In another use case, the Docker Operator Approach proves indispensable when deploying ML workflows across diverse environments. Imagine a situation where the team utilizes a mix of on-premises servers and cloud platforms. The Docker Operator facilitates consistency by encapsulating dependencies within containers, ensuring that the same environment is replicated regardless of the underlying infrastructure. This is particularly beneficial for reproducibility and cross-team collaboration, as the entire workflow, including dependencies, is encapsulated in a portable Docker image.

Moreover, the EC2 Operator Approach shines in scenarios demanding dynamic resource allocation. Consider a situation where a DL model undergoes periodic training with varying data volumes. The EC2 Operator can be employed to dynamically provision Amazon EC2 instances with the requisite computing power for training tasks, optimizing resource utilization. This flexibility is especially advantageous in cost-conscious environments, as computational resources are scaled up or down based on the workload, ensuring efficient usage and cost-effectiveness.

In essence, the integration of Python, Docker, and EC2 operators in Apache Airflow not only streamlines ML workflows but also provides a versatile framework for collaboration, adaptability, and resource optimization. This trifecta of operators, orchestrated by Airflow, empowers data science teams to navigate the complexities of modern ML with efficiency and agility.

3.4.3 Model Deployment and Monitoring

Model Deployment and Monitoring explore the critical phases of transitioning deep learning models into production environments and ensuring their ongoing performance and reliability. This section explores the strategies and techniques employed in the productization of DL models, addressing key considerations in deploying models at scale. Additionally, it examines the methodologies for monitoring the performance and health of DL models in real-world

production settings, highlighting the importance of continuous monitoring for maintaining optimal functionality.

- **Utilizing ONNX for Enhanced Model Interoperability and Deployment**

Deep learning algorithms culminate in model files that encapsulate the relationships between input data and output predictions. These models, often stored in formats like .pth or .h5, may require portability for deployment across diverse environments.

ONNX [94] serves as a bridge between various machine learning libraries, including PyTorch, TensorFlow, MXNet, and Caffe, allowing models trained in one framework to be deployed using another. Its core function involves exporting models into the .onnx format, a serialized representation in a protobuf file. Currently, ONNX natively supports PyTorch, CNTK, MXNet, and Caffe2, with converters available for TensorFlow and CoreML.

Consider a scenario where a deep learning model is trained to predict food freshness from images. Initially developed in PyTorch, the model requires deployment in an iOS app using CoreML. With ONNX, the transition between frameworks becomes seamless. Exporting a model to ONNX is a straightforward process, as demonstrated by a single line of code in PyTorch. Subsequently, tools like ONNX-CoreML facilitate the integration of the pre-trained model into an iOS app, ensuring compatibility and ease of use.

ONNX plays a pivotal role in streamlining machine learning workflows, offering a standardized format for model interchangeability and deployment. By providing a common set of operators and facilitating hardware optimization, ONNX empowers developers to focus on model development without being constrained by inference time considerations. With support for various inference engines, ONNX enables high-performance deployment across a wide range of hardware devices.

ONNX Implementation in Music Genre Classification Pipelines:

In the context of music genre classification utilizing deep learning, the implementation of ONNX with a ResNet-34 model offers a streamlined approach to model conversion and deployment. The purpose of this thesis is not to identify the optimal model for this task but to present and implement the DL workflow.

Converting deep learning models from PyTorch to ONNX begins with loading the ResNet-34 model stored in AWS S3. Ensuring the model is in evaluation mode is essential, as certain operations behave differently during inference than training. With the model in evaluation mode and the dummy input defined, the next step involves specifying the input and output names for the exported ONNX model. Utilizing the `torch.onnx.export` function facilitates the conversion process, requiring parameters such as the model, dummy input, output file name, input names, output names, and the choice to export parameters. Following the conversion, utilizing the 'onnxruntime' Python package enables the creation of an inference session to make predictions with the ONNX model. This involves loading the ONNX model file and supplying input data to obtain the model output, which can then be used for inference tasks.

The conversion of the ResNet-34 model to ONNX format in this thesis is beneficial for several reasons. Firstly, ONNX enables interoperability between different deep learning frameworks, allowing for seamless integration of models trained in PyTorch into production systems using other frameworks. Additionally, ONNX facilitates hardware optimization, improving the efficiency and performance of the deployed model across various hardware platforms.

Furthermore, by converting the model to ONNX, future experimentation and deployment efforts can be conducted with greater flexibility and scalability, contributing to the overall efficiency and effectiveness of the music genre classification pipeline.

By implementing ONNX with a ResNet-34 model in the music genre classification pipeline, this thesis aims to demonstrate the practical application of ONNX for model interoperability and deployment, contributing to the broader understanding of leveraging ONNX in deep learning workflows.

- **Model and Data Monitoring with Evidently AI**

Evidently AI [52] is a powerful Python library designed for monitoring machine learning models in production environments, offering comprehensive testing and monitoring capabilities across various model types and data formats.

Evidently AI facilitates the generation of detailed reports to explore and debug the quality of machine learning models or data. Users can implement checks as part of their prediction pipelines using Test Suites, ensuring robustness and reliability in model predictions. Additionally, Evidently AI enables the deployment of live monitoring dashboards, allowing users to track how metrics change over time and establish a continuous monitoring process for both batch and real-time ML models.

One of the key functionalities of Evidently AI is its ability to detect and analyze changes in input feature distributions, commonly known as data drift. By utilizing the DataDriftPreset, users can visualize and analyze data drift over time, identifying potential shifts in the data distribution that may impact model performance. Evidently AI enables users to evaluate data drift in various scenarios, such as monitoring model performance without ground truth labels, debugging model quality decay, understanding historical data drift, and deciding on model retraining strategies. Evidently AI's Data Drift report provides detailed insights into changes in input data, applying suitable drift detection methods for numerical, categorical, or text features. Users can compare distributions between reference and current datasets, explore feature values, and assess dataset drift using statistical tests. The report offers aggregated visuals, including data drift summaries, tables, and distributions, enabling users to quickly identify drifting features and understand the overall impact on model performance.

Implementation of Evidently AI for Music Genre Classification

Evidently AI offers robust tools for evaluating and monitoring classification models, ideal for music genre classification tasks. Its pre-built Reports and Test Suites enable comprehensive analysis, whether in binary or multi-class scenarios. Users can continuously monitor model performance in real-world production, scheduling regular test suite runs to compare against predefined expectations and using visual reports for stakeholder communication. The test suite facilitates proactive model retraining by detecting declines in quality, ensuring model accuracy over time. Visual reports help identify areas for improvement, analyzing error-prone segments and refining model performance iteratively. Evidently AI simplifies result analysis across different stages, from training to A/B testing and deployment.

To implement Evidently AI for music genre classification, begin by ensuring you have already established the training pipeline for your classification model. Once this is in place, integrate Evidently AI into your project by installing the library using pip install evidently. Import the necessary modules and functions from Evidently to access its reporting and testing functionalities within your Python script. Utilize Evidently AI's pre-built Classification

Performance Report to assess your model's quality by providing the test set containing ground truth and predicted genre labels. Additionally, set up test suites within your pipeline to automate performance monitoring. Schedule regular runs of these test suites to evaluate accuracy and identify potential issues like performance drift. Finally, interpret the results of the generated reports and test suite outputs, analyzing key metrics such as accuracy, precision, recall, F1-score, and confusion matrices. Use these insights to make informed decisions regarding model retraining, performance improvement, and result analysis.

Classification Report

To produce a classification report tailored for music genre classification using Evidently, it is imperative to curate the model logs into a structured pandas DataFrame. Each entry within the DataFrame should encapsulate pertinent details, including the input features, predicted labels (or probabilities), and ground truth labels. Once this data preparation step is completed, one can seamlessly employ Evidently's ClassificationPreset to conduct a rigorous analysis of the classification model's efficacy (Figure 16).

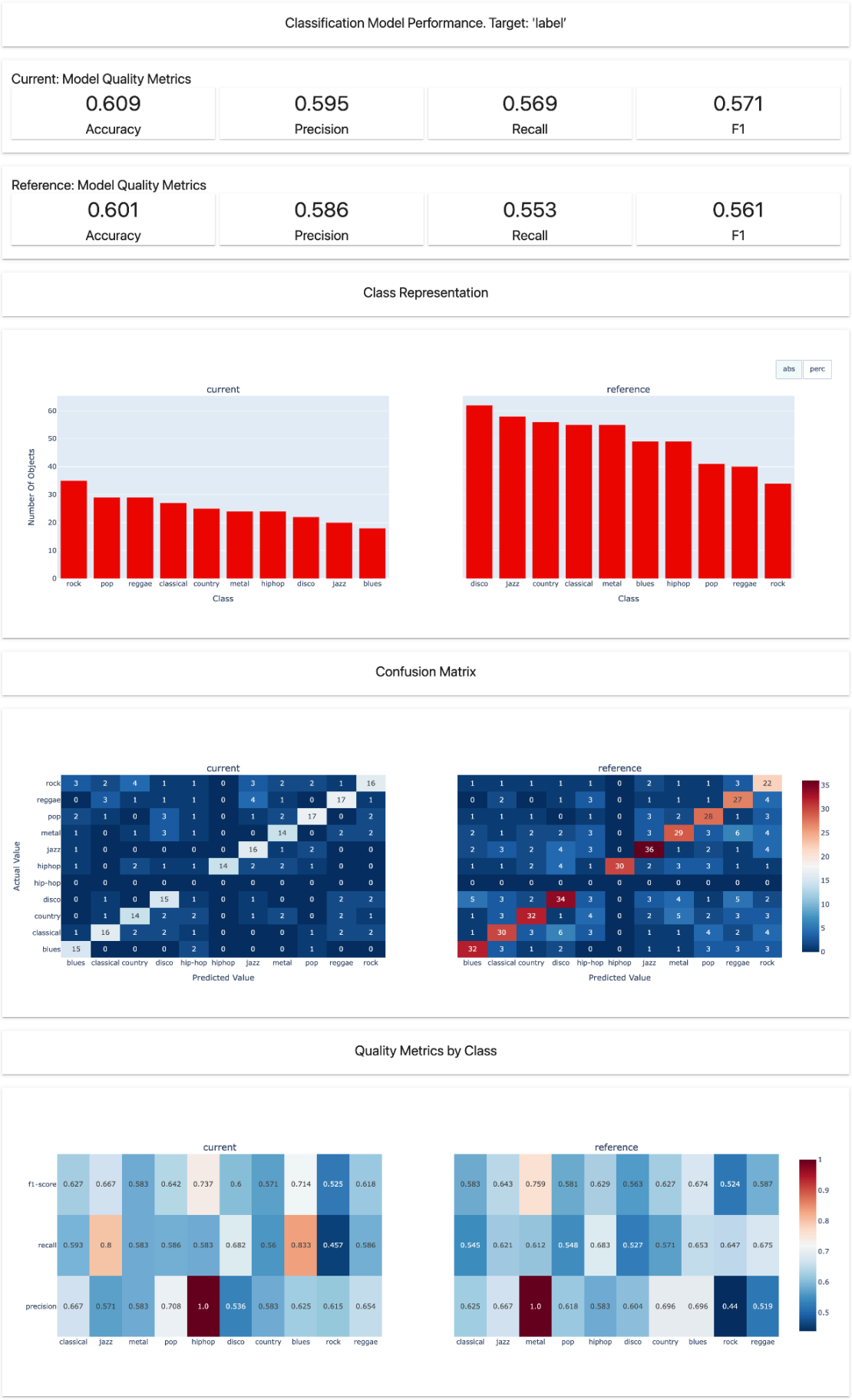


Figure 16 Classification Report for Music Genre Classification Using Evidently AI

The ensuing classification report, facilitated by Evidently, comprises a multifaceted array of components meticulously designed to furnish insights into the model's quality and performance in the context of music genre classification. Commencing with the Model Quality Summary Metrics, the report undertakes the computation of quintessential metrics such as Accuracy, Precision, Recall, F1-score, ROC AUC, and LogLoss. These metrics serve as pillars in discerning the overall performance of the classification model across diverse music genres. Furthermore, Evidently's provision of interactive visualizations endeavors to pinpoint areas of model fallibility, thereby fostering opportunities for refinement. Subsequently, the report unfolds to unveil Class Representation, offering a panoramic view of the distribution of musical compositions across various genres. Complementing this, the Confusion Matrix encapsulates the nuances of classification errors, elucidating their typology. Ultimately, Quality Metrics by Class empowers a granular examination of model performance, affording insights into its proficiency in discerning individual music genres. Through the amalgamation of these components, stakeholders can meticulously scrutinize the efficacy of the music genre classification model, thereby fostering informed decision-making and continuous improvement endeavors.

Data Drift Report

To analyze data drift in the context of music genre classification, Evidently offers a robust solution encapsulated within its Data Drift report. This tool is indispensable for detecting and exploring changes in the input data, providing critical insights into the evolving nature of the dataset over time. Leveraging suitable drift detection methods tailored for numerical, categorical, or text features, the Data Drift report furnishes a comprehensive overview of feature values and distributions across two datasets.

Figure 17 Data Drift Report for Music Genre Classification Using Evidently AI

At its core, the Data Drift report (Figure 17) operates on the premise of comparative analysis between a reference dataset and the current production data. By scrutinizing the input features' distributions, it discerns subtle variations indicative of data drift. Essential to this process is ensuring the datasets' schema alignment, with identical features and a consistent column mapping strategy. Through interactive visualizations, stakeholders can glean insights into the extent of data drift, represented in the Data Drift Summary, which encapsulates the share of drifting features and an aggregate Dataset Drift result. Furthermore, the Data Drift Table offers a detailed breakdown of drifting features, facilitating a granular examination of their characteristics and typology. Complemented by Data Distribution by Feature and Data Drift by Feature visualizations, stakeholders gain deeper insights into the evolving dataset's nuances, empowering proactive measures to mitigate the impacts of data drift on the music genre classification model's performance.

Data Quality Tests

In the domain of music genre classification, maintaining data integrity is crucial, and Evidently's Data Quality report (Figure 18) stands as a vital instrument for this task. Seamlessly integrated into the pipeline, this report enables thorough evaluations of data batches, even without a reference dataset, using the DataQualityTestPreset within a Test Suite. It furnishes users with comprehensive insights into descriptive statistics, uncovering crucial aspects such as missing data, duplicates, and features with minimal variation. The report's automated test condition generation, informed by predefined heuristics or provided reference datasets, ensures efficient identification and resolution of potential issues. Additionally, users can customize the assessment process by employing bespoke conditions, thereby optimizing data quality checks to align with project-specific requirements. Ultimately, the Data Quality report serves as a proactive safeguard, fortifying data integrity and fostering confidence in the music genre classification pipeline's performance.

Figure 18 Data Quality Tests for Musig Genre Classification

Data Stability Tests

To ensure the stability of data used in the music genre classification pipeline, integrating Evidently's Data Stability Test Suite (Figure 19) proves invaluable. By incorporating the DataStabilityTestPreset within a dedicated Test Suite, users can effectively assess the consistency of newly acquired data batches compared to previous ones. This preset facilitates the calculation of diverse dataset and feature statistics, enabling the detection of anomalies such as new categorical values, out-of-range values, or fluctuations in data volume. Leveraging a reference dataset, Evidently automatically generates test conditions, streamlining the evaluation process. Additionally, users have the flexibility to define custom conditions, tailoring stability checks to specific project needs. Ultimately, the Data Stability Test Suite acts as a proactive mechanism, ensuring the reliability and continuity of data inputs within the music genre classification pipeline.

Figure 19 Data Stability Tests for Music Genre Classification

4 CONCLUSIONS

In summary, this thesis has undertaken a rigorous exploration of the integration of DL techniques into ML systems, with a specific lens on the operational aspects highlighted by MLOps. The primary focus has been deciphering the intricacies of developing a DL Pipeline, with a specific emphasis on its real-world application in Music Genre Classification. Throughout the journey, the significance of MLOps, its tools, and their impact on research and industry practices has been underscored.

This work has delved into the fundamental stages of developing a DL Pipeline, from data preparation to model evaluation, highlighting the complexities that necessitate a systematic approach. The pivotal role of MLOps in addressing these intricacies becomes apparent, especially when considering its application in real-world scenarios.

Chapter 2 has shed light on the operational facets of ML, emphasizing the emergence of MLOps and the challenges in traditional ML workflows. This analysis underscores the need for a systematic and efficient approach that MLOps provides, making it particularly beneficial for both research and industry. In the context of research, the systematic workflows offered by MLOps ensure the reproducibility of experiments and the transparent management of experimental pipelines. This not only contributes to the credibility of research findings but also facilitates collaboration and knowledge sharing among researchers.

In the industrial landscape, the challenges of managing and deploying increasingly complex ML models are met with practical solutions through the adoption of MLOps. The selection and integration of MLOps tools ensure scalability, reproducibility, and efficient deployment of models in production environments. This is crucial for industries seeking to leverage ML for enhanced decision-making, process automation, and overall operational efficiency.

The Music Genre Classification use case in Chapter 3 serves as a tangible demonstration of the benefits of the proposed MLOps tools in both research and industry. By seamlessly integrating MLOps principles, this work emphasizes the reproducibility and scalability of the DL pipeline. In research, this translates to the ability to replicate experiments and validate results, fostering a culture of transparency and reliability. In industry, the MLOps-driven pipeline ensures a smooth transition from development to deployment, facilitating the integration of ML models into operational workflows.

In accordance with the specifications outlined in the abstract regarding an ideal ML workflow, the framework proposed in this thesis and exemplified through the Music Genre Classification case study demonstrates robust execution in a highly available environment. By leveraging MLOps principles, our framework ensures the reliability and availability of the DL pipeline, facilitating uninterrupted model training, evaluation, and deployment processes. Moreover, revisitable reproducibility is a cornerstone of our approach. Through meticulous version control, automated logging, and systematic documentation of experiments, our framework enables researchers and practitioners to revisit and reproduce results with ease, fostering transparency and reliability in experimentation. Furthermore, the integration of automation techniques minimizes manual intervention, ensuring streamlined workflows and reducing the risk of human error. This not only enhances efficiency but also allows for rapid iteration and experimentation. Additionally, our framework emphasizes easy extendability, enabling seamless integration of new data sources, algorithms, or model architectures. This flexibility ensures adaptability to evolving requirements and empowers users to continuously enhance

and refine their ML solutions. Lastly, the scalable capabilities of our framework enable it to handle larger tasks concurrently, accommodating the growing complexity and volume of data in modern ML applications. Through these features, our proposed MLOps framework and the Music Genre Classification case study embody the ideals of an ideal ML workflow, offering a systematic, efficient, and scalable approach to machine learning operations."

The innovation within this work lies not only in its theoretical discussions but also in its practical applications, emphasizing the broader implications of adopting MLOps in the research and industrial domains. By showcasing the benefits of MLOps tools in managing and deploying ML models, this thesis contributes to the broader conversation on the future of ML practices in both research and industry. The proposed MLOps system stands as a valuable asset, offering a systematic and efficient approach to the complexities of contemporary ML, making it indispensable for advancing research and promoting innovation in various industrial sectors.

Bibliography – References – Online sources

1. *Dai, H., Pears, N., Smith, W. a. P., & Duncan, C. A. (2019). Statistical modeling of craniofacial shape and texture. International Journal of Computer Vision, 128(2), 547–571. https://doi.org/10.1007/s11263-019-01260-7*
2. *Kamath, R. (Photographer). (2018, December 4). Bird image. [Photograph]. Pexels. https://www.pexels.com/photo/photo-of-perched-parakeet-1661179/*
3. *Nguyen, Q. H., Ly, H.-B., Ho, L. S., Al-Ansari, N., Le, H. V., Tran, V. Q., Prakash, I., & Pham, B. T. (2021). Influence of Data Splitting on Performance of Machine Learning Models in Prediction of Shear Strength of Soil. Mathematical Problems in Engineering, 2021, 1–15. https://doi.org/10.1155/2021/4832864*
4. *Mikolov, T., Ilya Sutskever, Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. Neural Information Processing Systems, 26, 3111–3119.*
5. *Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735*
6. *Zhang, Z. (2018, June). Improved adam optimizer for deep neural networks. In 2018 IEEE/ACM 26th international symposium on quality of service (IWQoS) (pp. 1-2). Ieee.*
7. *Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324. https://doi.org/10.1109/5.726791*
8. *Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning, volume 1.*
9. *Provost, F., & Fawcett, T. (2013). Data Science for Business: What You Need to Know About Data Mining and Data-Analytic Thinking. O'Reilly Media, Inc.*
10. *Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002*
11. *Willmott, C. J. (1981). On the validation of models. Physical geography, 2(2), 184-194.*
12. *Ruf, P., Madan, M., Reich, C., & Ould-Abdeslam, D. (2021). Demystifying mlops and presenting a recipe for the selection of open-source tools. Applied Sciences, 11(19), 8861.*
13. *Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 28). Curran Associates, Inc.: Cambridge, MA, USA.*
14. *Machine Learning Operations | International Journal of Information Technology Insights & Transformations [ISSN: 2581-5172 (online)]. (n.d.). Technology.eurekajournals.com. Retrieved March 7, 2024, from https://technology.eurekajournals.com/index.php/IJITIT/article/view/655*
15. *Raj, E., Westerlund, M., & Espinosa-Leal, L. (2021). Reliable Fleet Analytics for Edge IoT Solutions. ArXiv (Cornell University).*
16. *Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of Big Data, 2(1). https://doi.org/10.1186/s40537-014-0007-7*

17. *Hoffmann, D. S. (2022). Big Data and Deep Learning Models. Principia: An International Journal of Epistemology, 26(3). https://doi.org/10.5007/1808-1711.2022.e84419*
18. *Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine Learning Operations (MLOps): Overview, Definition, and Architecture. IEEE Access, 1–1. https://doi.org/10.1109/access.2023.3262138*
19. *Treveil, M., Omont, N., Clément Stenac, Lefevre, K., Phan, D., Joachim Zentici, Lavoillotte, A., Miyazaki, M., & Heidmann, L. (2020). Introducing MLOps. O'Reilly Media.*
20. *Raj, E. (2021). Engineering MLOps: Rapidly build, test, and manage production-ready machine learning life cycles at scale. Packt Publishing Ltd.*
21. *Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. IEEE transactions on knowledge and data engineering, 31(12), 2346-2363.*
22. *Baylor, D., Haas, K., Katsiapis, K., Leong, S., Liu, R., Menwald, C., Miao, H., Polyzotis, N., Trott, M., & Zinkevich, M. (2019). Continuous training for production ML in the TensorFlow Extended (TFX) platform. In Proceedings of the 2019 USENIX Conference on Operational Machine Learning (OpML 19), 51–53.*
23. *MLOps: Continuous delivery and automation pipelines in machine learning | Cloud Architecture Center. (n.d.). Google Cloud. Retrieved March 7, 2024, from https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning/*
24. *Mattson, P., Cheng, C., Coleman, C., Diamos, G., Micikevicius, P., Patterson, D., Tang, H., Wei, G. Y., Bailis, P., Bittorf, V., et al. (2019). Mlperf training benchmark. arXiv 2019, arXiv:1910.01500.*
25. *Shivakumar, S. K. (2020). Web performance monitoring and infrastructure planning. In Modern Web Performance Optimization: Methods, Tools, and Patterns to Speed Up Digital Platforms; Apress: Berkeley, CA, USA; pp. 175–212.*
26. *Sureddy, M. R., & Yallamula, P. (2020). A framework for monitoring data warehousing applications. International Research Journal of Engineering and Technology, 7, 7023–7029.*
27. *Sebastian-Coleman, L. (2013). Chapter 4—Data Quality and Measurement. In Measuring Data Quality for Ongoing Improvement; Sebastian-Coleman, L. (Ed.), MK Series on Business Intelligence; Morgan Kaufmann: Boston, MA, USA; pp. 39–53. doi:10.1016/B978-0-12-397033-6.00004-3.*
28. *Taleb, I., Serhani, M. A., & Dssouli, R. (2018). Big data quality: A survey. In Proceedings of the 2018 IEEE International Congress on Big Data (BigData Congress), San Francisco, CA, USA, 2–7 July 2018; pp. 166–173.*
29. *Ramasubramanian, K., & Singh, A. (2017). Machine learning model evaluation. In Machine Learning Using R; Apress: Berkeley, CA, USA; pp. 425–464.*
30. *Mehmood, H., Kostakos, P., Cortes, M., Anagnostopoulos, T., Pirttikangas, S., & Gilman, E. (2021). Concept Drift Adaptation Techniques in Distributed Environment for Real-World Data Streams. Smart Cities, 4(1), 349–371. https://doi.org/10.3390/smartcities4010021*
31. *MLflow - A platform for the machine learning lifecycle. (n.d.). MLflow. https://mlflow.org/*

32. *Comet Homepage. (n.d.). Comet. https://www.comet.com/site/*
33. *Weights & Biases – Developer tools for ML. (n.d.). Wandb.ai. https://wandb.ai/site*
34. *Open source Machine Learning at scale with Kubernetes - MLOps Lifecycle for data-scientists & machine-learning engineers - Model & Data Centric AI. (n.d.). Polyaxon. Retrieved March 7, 2024, from https://polyaxon.com/*
35. *Prefect - The New Standard in Dataflow Automation. (n.d.). Www.prefect.io. https://www.prefect.io/*
36. *Metaflow - a framework for real-life data science and ML. (2023, October 5). Www.metaflow.org. https://metaflow.org/*
37. *Kedro. (n.d.). Kedro. https://kedro.org/*
38. *MLOps framework for infrastructure agnostic ML pipelines. (n.d.). Www.zenml.io. Retrieved March 7, 2024, from https://www.zenml.io/home*
39. *Kubeflow. (n.d.). Kubeflow. https://www.kubeflow.org/*
40. *Flyte. (n.d.). Flyte. https://flyte.org/*
41. *Prometheus. (n.d.). Prometheus - Monitoring system & time series database. Prometheus.io. https://prometheus.io/*
42. *Grafana Labs. (2019). Grafana - The open platform for analytics and monitoring. Grafana Labs. https://grafana.com/*
43. *Apache Airflow Home. (n.d.). Apache Airflow. https://airflow.apache.org/*
44. *PachydermHome Page. (2022, July 6). Pachyderm. https://www.pachyderm.com/*
45. *Data Version Control · DVC. (n.d.). Data Version Control · DVC. https://dvc.org/*
46. *Quilt Data - Your data from instrument, to scientist, to filing. (n.d.). Www.quiltdata.com. Retrieved March 7, 2024, from https://quiltdata.com/*
47. *GX: a proactive, collaborative data quality platform. (n.d.). Www.greatexpectations.io. https://greatexpectations.io/*
48. *Git Large File Storage. (n.d.). Git Large File Storage. https://git-lfs.com/*
49. *Serving Models | TFX. (n.d.). TensorFlow. https://www.tensorflow.org/tfx/guide/serving*
50. *BentoML: Build, Ship, Scale AI Applications. (n.d.). Www.bentoml.com. Retrieved May 7, 2023, from https://www.bentoml.com/*
51. *Cortex | Internal Developer Portal. (n.d.). Www.cortex.io. Retrieved March 7, 2024, from https://www.cortex.io/*
52. *Evidently AI - Open-Source ML Monitoring and Observability. (n.d.). Www.evidentlyai.com. Retrieved December 7, 2024, from https://www.evidentlyai.com/*
53. *Fiddler AI. (n.d.). https://www.fiddler.ai/*
54. *Censius. (n.d.). https://censius.ai/*
55. *MLRun. (n.d.). https://www.mlrun.org/*
56. *Argo. (n.d.). https://argoproj.github.io/*
57. *Luigi. (n.d.). https://luigi.readthedocs.io/en/stable/workflows.html*
58. *Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. IEEE Transactions on Speech and Audio Processing, 10(5), 293–302.*
59. *Sturm, B. L. (2013). The GTZAN dataset: Its contents, its faults, its effects on evaluation, and its future use. arXiv [cs.SD].*
60. *FFmpeg. (2019). Ffmpeg.org. https://www.ffmpeg.org/*
61. *Librosa. (n.d.). Librosa.org. https://librosa.org/*
62. *librosa.feature.melspectrogram — librosa 0.8.0 documentation. (n.d.). Librosa.org. https://librosa.org/doc/main/generated/librosa.feature.melspectrogram.html*

63. *Logan, B. (2000, October). Mel frequency cepstral coefficients for music modeling. In ISMIR (Vol. 270, No. 1, p. 11).*
64. *Quatieri, T. F. (2002). Discrete-time speech signal processing: principles and practice. Pearson Education India.*
65. *Foote, J., & Cooper, M. (2001). Visualizing Musical Structure and Rhythm via Self-Similarity. ICMC, 1.*
66. *Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 25.*
67. *Huang, G., et al. (2017). Densely connected convolutional networks. Proceedings of the IEEE conference on computer vision and pattern recognition.*
68. *He, K., et al. (2016). Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition.*
69. *Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.*
70. *Bottou, L., et al. (2012). Stochastic gradient descent tricks. In Neural Networks: Tricks of the Trade (pp. 421–436). Springer.*
71. *Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4.2.*
72. *Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. The annals of mathematical statistics, 22(1), 79-86.*
73. *Meinard Müller. (2007). Information Retrieval for Music and Motion. https://doi.org/10.1007/978-3-540-74048-3*
74. *Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Luca Antiga, Alban Desmaison, Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Sasank Chilamkurthy, Steiner, B., Fang, L., & Bai, J. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. ArXiv (Cornell University), 32, 8026–8037.*
75. *SimmonsChance, & A HollidayMark. (2019). A comparison of two popular machine learning frameworks. Journal of Computing Sciences in Colleges. https://doi.org/10.5555/3381631.3381635.*
76. *David. (2011). Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. 2(1), 37–63.*
77. *Schütze, H., Manning, C. D., & Raghavan, P. (2008). Introduction to information retrieval (Vol. 39). Cambridge: Cambridge University Press.*
78. *Pachet, F., & Aucouturier, J.-J. (2004). Improving timbre similarity: How high is the sky. Journal of Negative Results in Speech and Audio Sciences, 1(1), 1-13.*
79. *Reed, J., & Lee, C.-H. (2009). On the importance of modeling temporal information in music tag annotation. 2009 IEEE International Conference on Acoustics, Speech and Signal Processing.*
80. *O'Reilly, A. (2018). Linux Pocket Guide: Essential Commands. O'Reilly Media.*
81. *Welcome to Click — Click Documentation (8.1.x). (n.d.). Click.palletsprojects.com. Retrieved February 7, 2024, from https://click.palletsprojects.com*
82. *Get Started: Data Pipelines. (n.d.). Data Version Control · DVC. Retrieved March 7, 2024, from https://dvc.org/doc/start/data-management/data-pipelines*

83. *Docker overview. (2020, April 9). Docker Documentation. https://docs.docker.com/get-started/overview/*
84. *Boettiger, C. (2015). An Introduction to Docker for Reproducible Research. ACM SIGOPS Operating Systems Review, 49(1), 71–79. https://doi.org/10.1145/2723872.2723882.*
85. *Airflow Docker Operator Documentation. https://airflow.apache.org/docs/apache-airflow-providers-docker/operators/docker/index.html.*
86. *Airflow EC2 Operator Documentation. https://airflow.apache.org/docs/apache-airflow-providers-amazon/operators/compute/aws_ec2_instance.html.*
87. *MLflow Documentation — MLflow 1.14.1 documentation. (n.d.). Www.mlflow.org. https://www.mlflow.org/docs/latest/index.html*
88. *MLflow: A Platform for Managing the End-to-End Machine Learning Lifecycle. (2024). Mlflow.org. https://www.mlflow.org*
89. *MLflow Tracking — MLflow 2.11.1 documentation. (n.d.). Www.mlflow.org. Retrieved March 8, 2024, from https://www.mlflow.org/docs/latest/tracking.html*
90. *Reproducibility and Collaboration with MLflow Projects. Www.mlflow.org. Retrieved March 8, 2024, from https://www.mlflow.org/docs/latest/projects.html*
91. *MLflow: An Open Platform for Research and Production. https://databricks.com/solutions/mlflow.*
92. *Managing Machine Learning Lifecycle with MLflow. Medium. (n.d.). Medium. Retrieved March 8, 2024, from https://towardsdatascience.com/managing-machine-learning-lifecycle-with-mlflow-5adc8885f98*
93. *Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19). Association for Computing Machinery, New York, NY, USA, 2623–2631. https://doi.org/10.1145/3292500.3330701.*
94. *ONNX | Home. (n.d.). Onnx.ai. https://onnx.ai/*