# Image-Based Music Genre Classification Using Convolutional Neural Networks

Juan Sebastian Gallego Villamarin

Dissertation written under the supervision of Professor Nicolo Bertani

Dissertation submitted in partial fulfillment of requirements for the MSc in Business Analytics, at the Universidade Católica Portuguesa
September 12th, 2023

# Abstract

**Title:** Image-Based Music Genre Classification Using Convolutional Neural Networks
**Author:** Juan Villamarin
**Keywords:** Convolutional Neural Networks, Music Genre Classification, Image-Based Classification, Deep Learning, Interpretability, Explainability

This dissertation presents a full investigation into the application of machine learning, specifically Convolutional Neural Networks (CNNs), in the music genre classification using vinyl album covers. Leveraging the latest advancements in deep learning and computer vision, the study introduces two models: one employing batch normalization and another utilizing Concept Whitening (CW) techniques to enhance model interpretability.

The main objectives are to evaluate these models' classification accuracy and interpretability. Using robust evaluation parameters, both models exhibit good accuracy rates in classifying music genres based on vinyl album covers. Notably, the concept whitening model adds another layer of interpretability, unraveling the black-box features found in standard neural networks. Empirical findings indicate that concept whitening enhances model interpretability and competes effectively in predictive performance.

This project serves in the pursuit of reliable and transparent image-based music genre classification systems. By comparing the two models on both accuracy and interpretability fronts, the study shines a light on the viability of incorporating concept whitening into standard CNN architectures for more explainable AI applications.

# Resumo

**Título:** Classificação de Géneros Musicais Baseada em Imagens Usando Redes Neuronais Convolucionais

**Autor:** Juan Villamarin

**Palavras-chave:** Redes Neuronais Convolucionais, Classificação de Géneros Musicais, Classificação Baseada em Imagens, Aprendizagem Profunda, Interpretabilidade, Explicabilidade

Esta dissertação apresenta uma investigação abrangente sobre a aplicação da aprendizagem de uma máquina, especificamente Redes Neuronais Convolucionais (CNNs), no campo da classificação de géneros musicais usando capas de álbuns de vinil. Aproveitando os avanços mais recentes em aprendizagem profunda e visão computacional, o estudo apresenta dois modelos: um que utiliza *Batch Normalization (BN)* e outro que emprega técnicas de *Concept Whitening (CW)* para aprimorar a interpretabilidade do modelo.

Os principais objetivos são avaliar a precisão na classificação desses modelos e a sua interpretabilidade. Usando parâmetros robustos de avaliação, ambos os modelos demonstram boas taxas de precisão na classificação de géneros musicais com base em capas de álbuns de vinil. Notavelmente, o modelo de *concept whitening* adiciona outra camada de interpretabilidade, desvendando as características de caixa-preta encontradas em redes neurais padrão. Resultados empíricos indicam que o *concept whitening* não apenas aprimora a interpretabilidade do modelo, mas também compete eficazmente em termos de desempenho preditivo.

Este projeto visa a criação de sistemas confiáveis e transparentes de classificação de géneros musicais baseados em imagens. Ao comparar os dois modelos em termos de precisão e interpretabilidade, o estudo destaca a viabilidade de incorporar o *concept whitening* em arquiteturas padrão de CNN para aplicações de IA mais explicáveis.

# Table of Contents


1. Introduction 8
   1.1. Research Objectives 9
   1.2. Significance of the Topic 10
2. Theoretical Discussion 12
   2.1. Literature Review 12
      2.1.1. Neural networks and Image Classification 12
      2.1.2. What is Deep Learning? 13
      2.1.3. The Mathematical Underpinnings: Convolutional Operations 13
      2.1.4. CNN Architecture: Building Blocks and Components 14
      2.1.5. Convolutional Layers 14
      2.1.6. Pooling Layers: Reducing Dimensionality 15
      2.1.7. Fully Connected Layers: The Decision Makers 16
      2.1.8. The Pioneering Leap: AlexNet 17
      2.1.9. From AlexNet to VGG to ResNet 18
      2.1.10. ResNet152: The Optimal Balance for Our Study 19
      2.1.11. Batch Normalization: Enhancing Training and Performance in ResNet152 20
      2.1.12. The Imperative of Interpretability in Machine Learning 21
      2.1.13. Interpretable Machine Learning: Principles and Challenges 22
      2.1.14. Concept Whitening: Bridging the Gap Between Performance and Interpretability 23
      2.1.15. The Versatility and Adaptability of Concept Whitening in Image Classification 25
      2.1.16. Future Horizons beyond CNNs: The Advent of Diffusion and Transformer Models 25
3. Methodology 27
   3.1. Data Collection 28
      3.1.1. Web Scraping 28
      3.1.2. Synthetic Data Generation 31
   3.2. Data Preparation 33
      3.2.1. Labeling Organization and Preprocessing 33
   3.3. Limitations and Constraints 35
4. Analyses and Results 36
   4.1. Install Required Packages 36

4.2. Dataset Organization and Data Generator 37
4.2.1. Directory Structure 37
4.2.2. Data Partitioning 38
4.2.3. Data Visualization 38
4.3. Constructing the Model Without Concept Whitening (bn_model) 41
4.4. Constructing the Model with Concept Whitening (cw_model) 44
4.5. Comparison Analysis 45
4.5.1. Model Without Concept Whitening (bn_model) 46
4.5.2. Model with Concept Whitening (cw_model) 47
4.5.3. Comparative Visualization 49
4.6. Deployment 49
4.6.1. Deployment of Model Without Concept Whitening (bn_model) 50
5. Main Conclusion and Limitations 52
6. Appendices 54
6.1. Glossary 54
6.2. Bibliography 57
6.3. Codebase 58

# Preface

First, a big thank you to my parents for always supporting my dreams. I am also grateful to my partner Daniela for her constant encouragement and discipline. Thanks to my friends at UCP for making this academic journey both fun and beautiful. A big thanks to Joren for the passion in predictive analytics and to Nuno for introducing interpretability. Special thanks to Nicolo Bertani for being such a great mentor and person, and to Filipa Reis for her unwavering support and care during my master's program. Thank you all for being part of this important chapter in my life, for your patience and love.

# 1. Introduction

This research explores the potential of convolutional neural networks (CNNs) for music genre classification using album cover artwork. The study compares standard CNNs to interpretable CNNs that apply concept whitening. Two TensorFlow models were developed in Python: a standard CNN with batch normalization and an interpretable CNN with an additional concept whitening layer.

The goal is to investigate if concept whitening can improve the interpretability of CNNs for music genre classification. Interpretability has become more relevant given recent advancements in large language models (LLMs) and growing concerns about AI risks. Transparent and explainable AI is critical for building trustworthy systems.

The study aims to contribute to improved music classification systems by evaluating concept whitening's effectiveness in CNNs. The results could provide insights into developing interpretable models that enhance music cataloging and recommendation. This may have significant implications for the music industry and listeners. More accurate genre classification could improve music search, discovery, and recommendations.

Overall, this work represents an ongoing effort to tap into CNNs' potential for music genre classification and stimulate creative insights. As a small step, refining these systems could achieve more accurate and efficient classification. The research aims to progress the development of transparent and trustworthy AI systems for music.

Significant challenges remain in improving the accuracy and interpretability of deep learning models for music. The subjective and nuanced nature of music genres makes classification inherently difficult. Expanding training datasets, optimizing model architectures, and integrating different data modalities like audio and lyrics could enhance performance. Testing variations in concept whitening techniques may also yield further improvements.

This research highlights the need for interdisciplinary collaboration between machine learning experts and musicologists to advance music classification AI. Domain expertise in music

theory and acoustics is essential for constructing robust systems. Human evaluation and feedback loops will be critical for iteratively refining model outputs. Ongoing advances in explainable AI and representation learning could hugely benefit this problem space.

While this work focuses specifically on album artwork and CNN models, the findings contribute to the broader mission of developing AI that can understand music holistically. Improving interpretability and transparency will be key to building trust in these systems. The insights from this research underscore the exciting possibilities at the intersection of music, culture, and artificial intelligence.

## 1.1. Research Objectives

The goal of this study is to build and test convolutional neural networks for music genre classification based on album artwork. The three main goals are as follows:

First, a standard CNN model will be developed leveraging batch normalization layers. Batch normalization has become a staple technique for stabilizing deep neural networks. Studying conventional CNN architecture provides a performance baseline for comparison.

Second, an interpretable CNN will be designed by integrating concept whitening layers. Concept whitening imposes meaningful constraints on features to enhance model transparency. Evaluating this novel approach is central for assessing the feasibility of interpretable networks.

Finally, the two models will be critically assessed in parallel experiments. Quantitative measurements and qualitative examinations of internal representations will be used to compare accuracy and explainability. This side-by-side comparison reveals the trade-offs between performance and interpretability.

These objectives work together to advance the state-of-the-art in deep learning for music classification. And more broadly, this research aims to enhance a comprehensive understanding of music through machine learning.

## 1.2. Significance of the Topic

In recent years, the growth of digital music libraries has led to increased interest in image-based music genre classification. This involves using album cover artwork to automatically classify music genres, which can enhance music recommendation systems and improve the overall user experience.

AlexNet was one of the first large CNN models to achieve breakthrough results on image classification using the ImageNet dataset. As noted by Krizhevsky et al. (2012), "the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don’t have"[7].The architecture of AlexNet incorporates several innovations that were crucial to its performance, including the non-saturating ReLU activation function, overlapping pooling, aggressive data augmentation, and dropout regularization features. These advancements allowed AlexNet to efficiently train on the large ImageNet dataset using the computational power of GPUs.

In this work, we leverage a similar CNN architecture to AlexNet, incorporating its key innovations. We train our model on album cover images labeled with music genres rather than the object categories of ImageNet. However, the underlying CNN architecture provides strong inductive biases and prior knowledge that allow the model to learn robust features, even when data is limited compared to the complexity of the task. The success of AlexNet on ImageNet classification provides inspiration for applying CNNs to music genre classification from artwork.

Convolutional neural networks (CNNs) have been shown to be highly effective in image classification tasks, making them a suitable choice for this type of research. As noted by Smith (2019), "CNNs have achieved state-of-the-art results across a variety of image classification benchmarks, including MNIST digit recognition, ImageNet classification, and CIFAR image labeling"[6]. By leveraging the power of CNNs, researchers can develop more accurate and efficient music genre classification systems that can help users discover new music and enjoy a more personalized listening experience.

One of the main advantages of using CNNs for music genre classification is their ability to learn and adapt to new data. By training a CNN with a large dataset of album cover artwork and their corresponding music genres, the network can learn to recognize patterns and features that are unique to each genre. This can help researchers develop more precise and accurate music genre classification systems.

In addition to improving music recommendation systems, image-based music genre classification can also have significant implications for music research and analysis. By analyzing the patterns and features that distinguish different music genres, researchers can gain a deeper understanding of the cultural and historical contexts that shape musical genres. This can help to uncover new insights into the evolution and development of different music genres over time.

Overall, image-based music genre classification is an exciting and rapidly evolving area of research that has the potential to revolutionize the way we discover and enjoy music. By leveraging the power of CNNs and other advanced machine learning techniques, researchers can develop more accurate and efficient music genre classification systems that can help us explore and appreciate the rich diversity of musical genres that exist around the world.

# 2. Theoretical Discussion

## 2.1. Literature Review

### 2.1.1. Neural networks and Image Classification

Neural networks are computational models inspired by the interconnected neurons in biological systems, such as the human brain. They serve as the backbone for a broad spectrum of machine learning applications, particularly in tasks that involve complex pattern recognition or decision-making (LeCun, Bengio, & Hinton, 2015).

In today's digital age, where visual content reigns supreme, the task of image classification has taken on unprecedented importance. Traditional machine learning algorithms, although effective in simpler scenarios, found themselves outmatched when it came to parsing the complex, high-dimensional data that images often present. It was against this backdrop that neural networks emerged, fundamentally altering the landscape of image classification.

In its most basic form, a neural network comprises an input layer, one or more hidden layers, and an output layer. Each layer consists of nodes, or neurons, connected to adjacent layers. These connections have associated weights, which are fine-tuned during the training phase to enable the model to make accurate predictions or classifications (Goodfellow, Bengio, & Courville, 2016).

Drawing inspiration from the human brain, neural networks serve as computational frameworks capable of learning patterns in data. These networks comprise interconnected layers of nodes, or "neurons," each contributing to the network's overall ability to classify images, among other tasks. However, it was the advent of deep neural networks—those with multiple hidden layers—that truly revolutionized image classification. These networks could learn hierarchical representations of images, capturing varying levels of complexity and abstraction.

### 2.1.2. What is Deep Learning?

Deep learning can be considered an extension of neural networks, specializing in algorithms that mimic the structure and function of the brain. Unlike traditional machine learning methods, deep learning models excel at learning from raw data, minimizing the need for manual feature extraction. Over the years, deep learning has made significant strides, advancing from basic perceptrons to complex architectures such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers. In the context of image classification, CNNs have been especially transformative, continuing the trajectory set by neural networks in revolutionizing this field. "Deep learning has evolved significantly over the years, from simple perceptrons in the 1960s to complex architectures like convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers. Among these, CNNs have been particularly impactful in the realm of image classification" (LeCun, Bengio, & Hinton, 2015).

Thus, deep learning, and more specifically CNNs, have become the go-to methods for complex tasks like image classification. But what enables these networks to perform so well? The answer lies in their mathematical foundations, particularly the concept of convolutional operations.

### 2.1.3. The Mathematical Underpinnings: Convolutional Operations

Convolution is a core operation in image processing that is particularly integral to CNNs. The basic idea involves the use of a filter or kernel that traverses the image. At each position, the filter performs element-wise multiplication with the section of the image it is currently on, and the results are summed up to form a single pixel in the output feature map. This procedure is repeated as the filter slides over the entire image, resulting in a complete feature map.

The feature map is a condensed representation of the input image, capturing essential components like edges, corners, or textures. These are the fundamental building blocks upon which more complex features can be recognized. Typically, a CNN will have multiple such filters, and consequently, multiple feature maps. These maps are stacked atop one another to form the convolutional layer in a CNN.

As data progresses through the layers of a CNN, each layer focuses on increasingly intricate features, building up a hierarchy of features at distinct levels of abstraction. This is what allows CNNs to excel at tasks like image recognition and classification. The convolution operation provides the mechanism for CNNs to learn these spatial hierarchies of features autonomously, without requiring handcrafted feature extraction techniques. This ability to automatically learn feature hierarchies makes convolution an indispensable operation in CNNs (LeCun, Bengio, & Hinton, 2015).

### 2.1.4. CNN Architecture: Building Blocks and Components

Convolution is the central element in Convolutional Neural Networks, enabling the network to learn hierarchical data patterns. In the initial layers, these patterns are straightforward—often capturing basic geometric forms like lines and corners. As the data advances through the network, the layers capture increasingly complex patterns, transitioning from simple geometric shapes to elaborate structures like faces. This hierarchical pattern recognition is a fundamental aspect of CNNs' effectiveness in tasks like image classification (Goodfellow, Bengio, & Courville, 2016).

The architecture of a CNN is tailored for image data and consists of three main layer types: convolutional layers, pooling layers, and fully connected layers. Convolutional layers are central to feature extraction. Pooling layers reduce the dimensionality of the data, making the network computationally more efficient. Finally, fully connected layers are used for classification tasks, mapping the extracted features to specific labels. This organizational structure is well-suited for handling grid-structured data and has been effective across a range of image recognition tasks (LeCun, Bengio, & Hinton, 2015).

### 2.1.5. Convolutional Layers

In Convolutional Neural Networks (CNNs), the convolutional layer serves as the first stage where raw image data enters the network. This differentiates them from fully connected layers that typically flatten the input data into vectors. The spatial organization of the input is

maintained in convolutional layers, a factor that is essential for image recognition applications.

Within a convolutional layer, there are several filters or kernels. These are matrices with learnable weights that move across the input image to generate feature maps. These feature maps are new representations of the input image, highlighting certain features. The filters can recognize local patterns within the image due to the use of small, overlapping receptive fields. In the beginning layers, simpler features like lines and edges are detected. As the data progresses through the network, later layers capture increasingly complex features, like shapes or objects. One characteristic of convolutional layers is the uniform application of filters across the input image. This uniformity ensures that the feature maps are sensitive to specific features no matter where they appear in the input, offering a degree of translation invariance (Goodfellow, Bengio, & Courville, 2016).

Convolutional layers also contribute to the efficiency of CNNs. The architecture allows for weight sharing and sparse connectivity and reduces the overall number of parameters that the network must learn. "CNNs can focus on local features with fewer parameters, making the model more efficient and less prone to overfitting" (Krizhevsky et al., 2012).

### 2.1.6. Pooling Layers: Reducing Dimensionality

Following convolutional layers, pooling layers come into play, aiding in dimensionality reduction. These layers serve to simplify the feature maps, thus enhancing computational efficiency and mitigating the risk of overfitting. Pooling layers achieve this by down sampling the feature maps, effectively reducing their size and complexity while retaining the essential features.

Max-pooling is the most generic form of pooling, which works by selecting the highest value in each localized region of the feature map. This method has the advantage of preserving the most salient features while eliminating less relevant information. As a result, the network gains some degree of translational invariance, meaning it becomes robust to minor changes in the position of features in the image.

Average pooling is another technique used for the same purpose, but it calculates the average value in each localized patch of the feature map. Although it also aids in dimensionality reduction, this technique could incorporate less relevant or noisy features, as it considers the average rather than the maximum value (Goodfellow, Bengio, & Courville, 2016).

The final stage in a typical CNN architecture is the fully connected layers.

### 2.1.7. Fully Connected Layers: The Decision Makers

Fully connected layers typically appear at the terminal part of the CNN architecture. They act as the decision-making body of the network. By the time the data reaches these layers, it has already been filtered and down sampled, preserving only the most significant features for the task at hand. The primary function of fully connected layers is to take these high-level features and combine them in a manner that allows the network to perform specific tasks, such as classification, object detection, or even more complex functions like image segmentation (Krizhevsky et al., 2012).

These layers are often accompanied by dropout layers or other regularization techniques to prevent overfitting, especially in deep networks. The fully connected layers can be seen as the stage where all the learned hierarchical features from the preceding layers are utilized for making a final high-level decision. They often employ activation functions like SoftMax for classification tasks, which provide probabilities for each class, aiding in interpretability (Goodfellow, Bengio, & Courville, 2016).

In sum, convolutional layers act as feature detectors, pooling layers introduce invariance while reducing computational load, and fully connected layers make the final decisions based on these features. Each layer type plays a crucial role in the network's ability to learn from complex, high-dimensional data, making CNNs an integral part of modern machine learning applications.

Effective in image recognition tasks, the power of CNNs significantly amplifies when multiple convolutional layers are stacked on top of each other. Each subsequent layer can

learn more complex features by combining simpler features learned by the previous layers. This hierarchical approach to learning features is a cornerstone of CNNs and is particularly effective for complex image recognition tasks (LeCun, Bengio, & Hinton, 2015).

Krizhevsky et al. (2012) demonstrated the efficacy of deep CNNs in their ImageNet classification model. The model, known as AlexNet, employed multiple convolutional layers to learn a hierarchy of features, which contributed to its groundbreaking performance on the ImageNet Large Scale Visual Recognition Challenge.

### 2.1.8. The Pioneering Leap: AlexNet

The introduction of AlexNet in 2012 by Krizhevsky, Sutskever, and Hinton was a watershed moment in the field of deep learning, particularly for computer vision. The architecture was groundbreaking not just because it was deeper than its predecessors, but also because it efficiently utilized convolutional and pooling layers to handle much larger and more complex image datasets, like ImageNet. For the first time, a deep learning model significantly outperformed traditional machine learning algorithms in a major competition, solidifying the role of CNNs in image recognition tasks (Krizhevsky, Sutskever, & Hinton, 2012).

AlexNet used the principles of hierarchical feature learning through its multiple convolutional layers, each followed by max-pooling. This helped in not just detecting basic features like edges and textures in the initial layers but also more complex, higher-level features in the deeper layers. Fully connected layers were then used for the final classification. The model also introduced the use of the Rectified Linear Unit (ReLU) activation function, which solved the vanishing gradient problem, enabling deeper networks to be trained more efficiently.

After the advent of AlexNet, the field saw an explosion of innovative architectures, each attempting to address specific challenges or improve upon the capabilities of previous models.

### 2.1.9. From AlexNet to VGG to ResNet

The introduction of AlexNet was a seminal moment, but it was just the beginning of a wave of innovations that would sweep the field. AlexNet was groundbreaking in its use of multiple

convolutional and pooling layers, arranged in a deep architecture. The model effectively utilized the Rectified Linear Unit (ReLU) activation function, solving the vanishing gradient problem and enabling deeper networks to be trained more efficiently (Krizhevsky, Sutskever, & Hinton, 2012).
While AlexNet laid the foundation for deep CNNs, it opened the door to questions about the role of depth and complexity in these networks. This led to the development of the VGG architecture, which took the idea of depth to a new level.

The VGG architecture, developed by the Visual Geometry Group, emphasized the importance of depth by employing architectures with up to 19 layers. Unlike the varied size of convolutional filters used in AlexNet, VGG adopted a simpler and more consistent design—using only 3x3 convolutional layers stacked on top of each other in increasing depth. This consistency made the architecture easier to understand and modify. It demonstrated that depth could significantly improve the network's performance, particularly in capturing hierarchical features at multiple scales (Goodfellow, Bengio, & Courville, 2016). VGG further validated the principles of hierarchical feature learning and spatial hierarchies in images, as initially outlined by LeCun, Bengio, & Hinton (2015).

The depth and simplicity of VGG made it a popular choice for various computer vision tasks. However, its depth also led to a significant increase in the number of parameters, making the network computationally expensive. This trade-off between performance and computational efficiency would become a recurring theme in the development of future architectures.

However, while VGG emphasized depth, it also highlighted the limitations of solely focusing on this aspect. Increasing depth improved performance but at the cost of computational efficiency and the risk of overfitting. This set the stage for architectures like ResNet, which introduced residual connections to enable even deeper networks without the associated increase in computational complexity (Goodfellow, Bengio, & Courville, 2016).

As the field progressed, the tug-of-war between depth and efficiency became more nuanced. ResNet152, for example, managed to achieve a good balance by using fewer parameters than VGG while delivering better performance, thanks to its residual connections. This balance

between depth and efficiency has been a critical factor in the choice of architecture for specific tasks, given the varying constraints of computational resources and dataset complexity.

### 2.1.10. ResNet152: The Optimal Balance for Our Study

Building on the understanding that the architecture of a CNN is critical in determining both its performance and computational efficiency, our study opted for ResNet152 as the architecture of choice. This decision was influenced by the nuanced balance that ResNet152 strikes between depth and efficiency, a critical consideration in modern deep learning applications.

ResNet152 extends the architectural innovations of its predecessors by introducing the concept of residual learning. Residual connections, or "skip connections," allow the output of one layer to bypass one or more intermediate layers and be summed with the output of a later layer. This mechanism aids in solving the vanishing gradient problem, enabling the network to learn effectively even when it is extremely deep. These residual connections perform an identity function over the activation of shallower layers, ensuring that the same activation is produced. This output is then added to the activation of the next layer, making the learning process more efficient (Goodfellow, Bengio, & Courville, 2016).

Moreover, ResNet152 has a notable computational advantage; it achieves better performance than VGG while using far fewer parameters. Specifically, VGG19 has approximately 138 million parameters, whereas ResNet152 only has 60 million. This lower number of parameters translates to greater computational efficiency, faster training times, and less susceptibility to overfitting (LeCun, Bengio, & Hinton, 2015).

To facilitate these residual connections, ResNet152 employs consistent dimensions for its convolutions throughout the network. It uses 3x3 convolutions uniformly to ensure that the dimensions match when outputs are summed. This consistency in convolutional dimensions allows for the smooth operation of the residual connections, further enhancing the model's efficiency.

Additionally, ResNet152 uses global average pooling instead of fully connected layers in the final stages of the network. This choice reduces the model's complexity and risk of overfitting, making it more robust to variations in input sizes and more adaptable to different tasks.

### 2.1.11. Batch Normalization: Enhancing Training and Performance in ResNet152

The utility of ResNet152 as an efficient and effective architecture for image recognition tasks is further amplified by the incorporation of batch normalization. Initially conceptualized by Ioffe and Szegedy in 2015, batch normalization aims to expedite the training of deep networks by mitigating the problem of internal covariate shift—helping the model to generalize better by normalizing the inputs (Ioffe & Szegedy, 2015).

Batch normalization offers several key advantages that align closely with the strengths of ResNet152. For instance, it enables the use of higher learning rates, thereby accelerating the often-time-consuming training process. This feature is especially advantageous for a robust yet computationally intensive model like ResNet152. Additionally, batch normalization serves a regularization function, which minimizes the necessity for other techniques like dropout that often add to the computational load. This is a perfect complement to ResNet152's innate efficiency and reduced risk of overfitting, owing to its use of global average pooling and fewer parameters compared to models like VGG19.

To deepen our understanding, the research by Santurkar et al. (2019) provides a nuanced view of batch normalization's effectiveness. They found that the technique does more than just adjust and scale activations; it fundamentally smoothens the optimization landscape, making it easier for learning algorithms to find optimal solutions. This has pronounced implications for the already complex architecture of ResNet152. The smoother landscape means that the intricacies of residual connections become easier to optimize, reinforcing ResNet152's utility for complex image recognition tasks.

In summary, the adoption of batch normalization is far from a trivial addition to the ResNet152 architecture; it is a strategic enhancement that transforms the model's training dynamics. When coupled with ResNet152's inherent strengths—like its residual connections

and reduced parameter count—the benefits of batch normalization make the architecture exceptionally well-suited for sophisticated image recognition tasks. Therefore, the integration of ResNet152 with batch normalization stands as a compelling reason for its selection in our study.

### 2.1.12. The Imperative of Interpretability in Machine Learning

As impactful as neural networks have been in transforming image classification, their complex architectures often serve as a double-edged sword. While they excel at learning intricate patterns in data, they are frequently criticized for their "black box" nature—highly accurate, but equally opaque. In applications where understanding the decision-making process is crucial, this lack of transparency can be a significant drawback. This brings us to the evolving discourse on interpretability in machine learning.

Interpretability refers to the extent to which a human can understand the reasoning behind a model's decisions. In simpler machine learning algorithms, the decision boundaries are often linear, and the logic is straightforward, making them easy to interpret. However, as we venture into the realms of deep neural networks and, more specifically, complex architectures like CNNs, the high dimensionality and non-linearity introduce levels of abstraction that are hard to untangle.

Yet, the demand for interpretability is not merely academic; it has real-world implications. Whether it is a healthcare professional trying to diagnose a disease from medical images, or a security system trying to distinguish between an intruder and a family member, the stakes are high. Understanding why a model made a specific classification not only provides insights into its strengths and weaknesses but also offers valuable context that can inform further action.

Therefore, while neural networks and deep learning have set new benchmarks in image classification, the conversation is incomplete without addressing the challenges and ongoing efforts in making these powerful models more interpretable.

### 2.1.13. Interpretable Machine Learning: Principles and Challenges

The appeal of machine learning models, particularly deep neural networks, rests on their unmatched performance across various tasks. However, as these models grow in complexity, so does the intricacy of their decision-making processes. This increasing opacity has prompted researchers to advocate for more interpretable machine learning models. Rudin et al. (2021) have delved deep into this topic, laying out the fundamental principles and challenges that shape the field of interpretable machine learning.

**Fundamental Principles of Interpretability**

Transparency: The inner workings of a model should be transparent to its users. This does not necessarily mean every user should understand every detail, but they should have access to a clear explanation of the model's decision logic at a level they can comprehend.
Justifiability: Every decision made by a machine learning model should be justifiable based on evidence. If a model classifies an image as a 'cat', there should be discernible features within the image that can be attributed to typical characteristics of a cat.
Understandability: Regardless of how advanced or sophisticated a model is, its decisions should be presented in a manner that is understandable to the end-user. This might involve visual aids, simplified explanations, or other methods tailored to the target audience.

**Grand Challenges in Achieving Interpretability**

Trade-off Between Performance and Interpretability: One of the primary challenges in interpretability is the perceived trade-off between model performance and its interpretability. While simpler models might be more transparent, they may not always deliver the best performance. Lack of Standardized Evaluation Metrics: Unlike accuracy or precision, interpretability is subjective and lacks standardized metrics. This makes it challenging to objectively compare the interpretability of different models.
Complexity of Real-world Data: Data in the real world is messy and often high-dimensional. Extracting interpretable features from such data is a significant challenge.

Diverse User Expectations: Different users have different expectations from interpretable models. A domain expert might want detailed insights, while a layperson might prefer a high-level overview. Catering to this diverse range of expectations is a challenge.
Rudin et al.'s insights underscore the importance of not just developing high-performing machine learning models but also ensuring these models are interpretable. Their work highlights the need for an integrated approach that considers both technical and human-centric aspects of interpretability.

In the context of image classification, as neural networks continue to dominate the field, it becomes crucial to integrate these principles of interpretability. Doing so not only makes the models more reliable but also fosters trust among users, paving the way for broader acceptance and application of these powerful tools in critical domains.

### 2.1.14. Concept Whitening: Bridging the Gap Between Performance and Interpretability

Concept Whitening is an advanced technique designed to improve the interpretability of deep neural networks that aligns closely with the principles of interpretability outlined by Rudin et al. (2021), namely Transparency, Justifiability, and Understandability. It operates by applying a "whitening" transformation to the activations of a specific layer within the neural network. This mathematical transformation aims to decorrelate or "whiten" these features, making them more independent of each other. By doing so, Concept Whitening allows each feature to represent distinct, meaningful aspects of the music album cover, such as color schemes indicative of a specific genre or icons that denote an artist's branding. By disentangling the high-level features, it provides a clearer pathway to understand what the neural network "sees" in a music album cover. For example, if an album is classified as "rock," Concept Whitening can help reveal whether it was the presence of electric guitars in the cover art or a specific color scheme that led to this classification.

In a seminal work, Chen et al. (2020) applied Concept Whitening to a ResNet-50 model trained on the ImageNet dataset. Their experiments demonstrated that the model's interpretability significantly improved without compromising its classification performance.

This finding is monumental, as it challenges the traditionally held view that there is an inherent trade-off between a model's performance and its interpretability. One of the most compelling advantages of Concept Whitening is its adaptability. It can be integrated into existing deep learning models with relative ease, making it a versatile tool for enhancing interpretability in music album image classification. This is particularly crucial in applications like recommendation engines, where understanding why a particular album cover has been classified under a specific genre can help in fine-tuning recommendations and avoiding algorithmic biases.

Chen et al. (2020) showed that this transformation improved model interpretability without affecting performance. While their study did not specifically focus on music album images, the technique's adaptability suggests that it could be just as effective in this domain, ensuring that labels like 'Rock,' 'Electronic,' or 'Hip Hop' assigned to album covers are transparently and justifiably made.

### 2.1.15. The Versatility and Adaptability of Concept Whitening in Image Classification

One of the most compelling advantages of Concept Whitening is its adaptability. It can be integrated into existing deep learning models with relative ease, making it a versatile tool for enhancing interpretability in music album image classification. This suggests that Concept Whitening can be retrofitted into already trained models, thereby saving computational costs and time; in applications like recommendation engines, where understanding why a particular album cover has been classified under a specific genre can help in fine-tuning recommendations and avoiding algorithmic biases.

The versatility of Concept Whitening becomes particularly valuable in specialized tasks like music album classification, where the interpretability of the model can significantly impact user trust and adoption. For instance, the reasons behind classifying a particular album cover as belonging to a specific genre need to be transparent to both the developers and the end-users. Concept Whitening can make this process more transparent, thereby aligning technological advancements with user-centric needs and ethical guidelines.

### 2.1.16. Future Horizons beyond CNNs: The Advent of Diffusion and Transformer Models

While Concept Whitening offers a promising avenue for enhancing model interpretability, the road ahead is not without challenges. Rudin et al. (2021) outlined several grand challenges, such as the lack of standardized metrics for interpretability and the inherent complexity of real-world data. Further research is needed to standardize Concept Whitening's implementation and assess its effectiveness across different neural architectures and varying domains.

It is worth noting that the field has moved beyond these architectures in recent years. The year 2023 marks an era where diffusion models and transformer models have started gaining attention for their capabilities in various image and data processing tasks. These new architectures have emerged as a natural extension to the problems that earlier models like ResNet152v2 aimed to solve but bring in more advanced mechanisms for data representation and feature extraction (Goodfellow, Bengio, & Courville, 2016).

Diffusion models, for example, have shown potential in generative tasks, and their application in image recognition is an area of active research. They operate by modeling the data distribution through a diffusion process and have proven to be effective in capturing complex data distributions. Though not directly comparable to CNNs, they offer a different paradigm for understanding and representing data.

Similarly, transformer models, originally designed for natural language processing tasks, have been adapted for image classification and have shown promising results. They have the advantage of capturing long-range dependencies in the data, something that traditional CNNs like ResNet152v2 struggle with. These models leverage self-attention mechanisms to weigh the importance of various parts of the input data, providing a more nuanced feature representation.

Considering these advancements, the choice of architecture—be it ResNet152v2, a diffusion model, or a transformer—would depend on the specific requirements of the task at hand,

including computational resources, the complexity of the data, and the need for interpretability among other factors (Krizhevsky, Sutskever, & Hinton, 2012).

# 3. Methodology

The methodology for this research aimed to assess the efficacy and interpretability of convolutional neural networks (CNNs) in music genre classification, using album artwork as the input data. The focus was on classifying vinyl album covers into rock, electronic, or hip-hop genres, utilizing a dataset of over 3000 images with dimensions of 300 by 300 pixels. These images were sourced from the Discogs marketplace and were supplemented with synthetic data generated using the Gretel.ai platform. Preprocessing involved resizing the images and normalizing pixel values.
Two CNN models were developed for the study. The first was based on a pre-trained ResNet152V2 architecture with added batch normalization layers. This model served as the control and was optimized for accuracy. The second model was an extension of the first but included concept whitening layers for increased interpretability. Both models were developed in Python using TensorFlow.

For evaluation, a mixed-methods approach was used that combined quantitative metrics such as accuracy and precision with qualitative methods, including visual inspection of the models' internal representations. The project was organized into modular Python scripts and later bundled into a Jupyter notebook:

- dataset_preparation.py for preprocessing
- model_without_whitening.py for the baseline model
- model_with_whitening.py for the concept whitening model
- comparison_analysis.py for evaluation

This structured methodology allowed for a comprehensive examination of the trade-offs between model performance and interpretability in the specific context of music genre classification based on album artwork. The concept whitening layers were implemented using resources pulled from the GitHub repository ConceptWhitening(zhiCHEN96) [13]. The entire

research process was documented in a Jupyter notebook, which, along with the Python scripts, was published on GitHub in a repository titled Image-Based-Music-Genre-Classification-Using-Convolutional-Neural-Networks https://github.com/juansgv/Image-Based-Music-Genre-Classification-Using-Convolutional-Neural-Networks

## 3.1. Data Collection

### 3.1.1. Web Scraping

In this research project, Discogs was selected as the primary source for data collection due to its extensive catalog of vinyl records, CDs, and other music formats. The platform's diverse range of album artwork across various genres made it an ideal choice. Python's BeautifulSoup package was utilized for web scraping, given its proficiency in parsing HTML and XML documents (ZenRows, 2020). Essential Python libraries such as 'requests' and 'urllib.request' were imported to facilitate the scraping process. Subsequently, BeautifulSoup was employed to access and parse the HTML content from specific Discogs URLs.
An example of such a URL is:

> https://www.discogs.com/sell/list?sort=listed%2Cdesc&limit=250&genre=Electronic&style=House&style=Techno&ships_from=United+States&format=Vinyl&format_desc=LP&format_desc=Album&page=1

The criterion for selecting album covers involved applying filters on Discogs' 'sell list.' These filters targeted LP album covers from the United States that were released between 1970 and 2012 and fell under the genres of Hip-hop, Rock, and Electronic. Once these filters were in place, the Python script executed the image downloads, saving them to a designated local directory. All scraping activities adhered to Discogs' terms of service and robots.txt file, ensuring responsible and ethical data collection.

The images downloaded from Discogs were saved in a 300x300 pixel format, contributing to a dataset of 2700 images. This dataset served as the foundational data for the training and

evaluation phases of the machine learning models. Organizing and preparing the dataset for subsequent processing ensured a smooth transition into the modeling stage.

The following Python code was utilized for the whole web scraping process:

**Web Scraping Code:**

```
import os
import requests
import urllib.request
from bs4 import BeautifulSoup

page = requests.get('URL').text  # 'URL' should be replaced with the actual URL
soup = BeautifulSoup(page, 'html.parser')
tags = [a.get('data-src') for a in soup.find_all('img')]
tags = [tag for tag in tags if tag is not None]

for tag in tags:
   try:
      urllib.request.urlretrieve(tag, os.path.basename(tag))
      print(f'Image downloaded: {tag}')
   except ValueError:
      print(f'Error downloading: {tag}')
```

**Sample Images:**

- **Electronic:**

*Figure 1. Scraped Electronic sample*

- **Rock:**

*Figure 2. Scraped Rock sample*

- **Hip-Hop:**

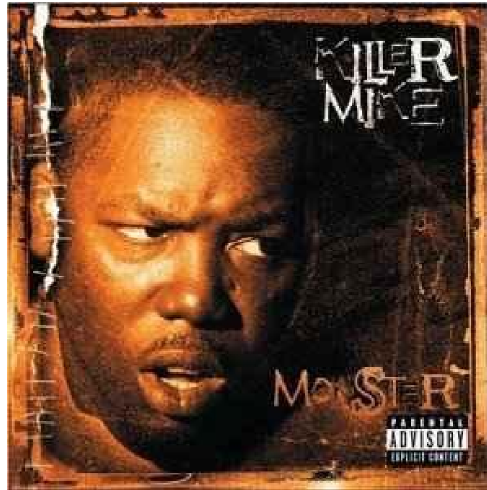


*Figure 3. Scraped Hip-hop sample*

### 3.1.2. Synthetic Data Generation

To augment the dataset and enhance model performance, 300 synthetic images were generated using the Gretel.ai platform. These images were equally distributed across the three genres—Electronic, Rock, and Hip-Hop—adding 100 images to each category. This increased the total dataset size to 3000 images.

Incorporating synthetic images into the dataset yielded several benefits. First, the additional variability in the data aided the model in generalizing to unseen examples. Second, the expanded dataset reduced the risk of the model overfitting during the training phase. This, in turn, had a positive impact on performance metrics like accuracy and precision. Additionally, generating synthetic images consumes fewer computational resources than the alternative of collecting more real-world images.

By merging real-world images scraped from the web with these synthetic images, the dataset has been made more robust and comprehensive. This strengthened dataset serves as a solid foundation for training and evaluating convolutional neural networks aimed at classifying music genres based on album artwork.

**Prompts for Gretel.ai**

- **Electronic:**

- Token name: Vinyl album cover
- Prompt: Vinyl album cover art of an electronic genre record
- Prompt keywords: front, photo-realistic

- **Rock:**

- Token name: Vinyl album cover
- Prompt: Vinyl album cover art of a rock genre record
- Prompt keywords: front, photo-realistic

- **Hip-Hop:**

- Token name: Vinyl album cover
- Prompt: Vinyl album cover art of a hip-hop genre record
- Prompt keywords: front, photo-realistic

**Sample Images**

- **Electronic:**

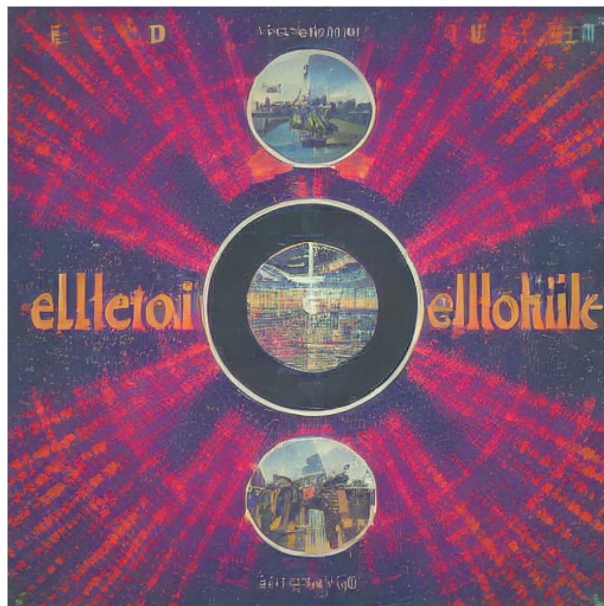

*Figure 4. Synthetic Electronic sample*

- **Rock:**

*Figure 5. Synthetic Rock sample*

- **Hip Hop:**

*Figure 6. Synthetic Hip-hop sample*

## 3.2. Data Preparation

### 3.2.1. Labeling Organization and Preprocessing

The organization and labeling of the dataset were key steps in its preparation for model training. The dataset comprised 3000 JPEG image files, neatly categorized into three genres: electronic, rock, and hip-hop. To maintain a balanced set, each category contained an equal number of images.

Each image in the dataset was tagged in a uniform, sequential numerical format. The first image was tagged as "image_1," the second as "image_2," and so on, up to "image_n," where 'n' represents the total number of images in the dataset.

For documentation and easy reference during model training, all image labels were systematically recorded in a CSV file. This file, named training_labels.csv, contained two columns: the first for the category or genre and the second for the image name. This organized approach to labeling and documentation ensured that the model training process could proceed without ambiguities or inconsistencies in the dataset.

| | image | category |
|---|---|---|
| 0 | image_1.jpeg | electronic |
| 1 | image_2.jpeg | electronic |
| 2 | image_3.jpeg | electronic |
| 3 | image_4.jpeg | electronic |
| 4 | image_5.jpeg | electronic |
| ... | ... | ... |
| 2455 | image_3316.jpeg | rock |
| 2456 | image_3317.jpeg | rock |
| 2457 | image_3318.jpeg | rock |
| 2458 | image_3319.jpeg | rock |
| 2459 | image_3320.jpeg | rock |

*Figure 7. Data frame training labels*

By taking these thorough steps in dataset organization and labeling, the research ensured a structured and reliable foundation for the subsequent phases of training and evaluating convolutional neural networks for music genre classification based on album artwork.

## 3.3. Limitations and Constraints

The models were trained and tested on album covers from the genres of Electronic, Rock, and Hip-Hop, and the dataset specifically consists of LP album covers from the United States, spanning the years 1970 to 2012. This focus could limit the application of the models to other musical genres, contemporary music, or album covers from other countries and cultures. Therefore, the research findings may have limited applicability to different contexts, genres, or geographical locations.

For future research, one direction could be to diversify the dataset by including album covers from a wider range of genres and from different time periods and locations. This could enhance the ability of the models to generalize about new, unseen data. Another avenue for future work could be to explore the incorporation of new diffusion models or to experiment with other advanced machine learning techniques. Such enhancements could improve the models' scalability, allowing them to handle larger or more complex datasets. These initiatives could widen the scope of applications for these models, making them more versatile tools for image-based classification tasks.

# 4. Analyses and Results

This chapter provides an in-depth discussion of the analyses performed and the results obtained. The code snippets included serve to illustrate key computational steps integral to the research. These snippets can be rerun locally for a deeper understanding or replication of the findings.

## 4.1. Install Required Packages

The first step in the computational analysis involved installing the necessary libraries. The required dependencies are listed in a requirements.txt file, available in the appendix for complete reference [28].

For the current investigation, TensorFlow was employed for building the machine learning models, while NumPy and Matplotlib were used for numerical computations and data visualization, respectively. Additionally, Pandas was integrated for its capabilities in data manipulation and analysis.

```
# Import TensorFlow and other foundational libraries
import os
import random
import numpy as np
import tensorflow as tf
import math
import matplotlib.pyplot as plt
import shutil
import pandas as pd
```

After importing these essential libraries, a seed was set for reproducibility in the computational processes. This ensures consistent results across different runs, aiding in the verification of the study's outcomes.

The groundwork was laid for the subsequent stages of data preprocessing, analysis, and model training.

## 4.2. Dataset Organization and Data Generator

Dataset organization is a crucial aspect of any machine learning project, as it directly impacts the model's training efficiency. First, the dataset_preparation.py script performed three main tasks to organize the data.

### 4.2.1. Directory Structure

Established a directory hierarchy by defining and creating folders for training, testing, and validation sets, as well as subfolders for each music genre—Electronic, Rock, and Hip-hop.

### 4.2.2. Data Partitioning

Splitting the images in the dataset into training, testing, and validation sets, based on their genre labels.

```
Training data set size : 1968
Test data set size : 246
Validation data set size : 246
```

*Figure 8. Data sets sizes*

### 4.2.3. Data Visualization

A subset of training images from each of the genres—Electronic, Rock, and Hip-hop—is visualized to provide an initial understanding of the data.

- **Electronic:**

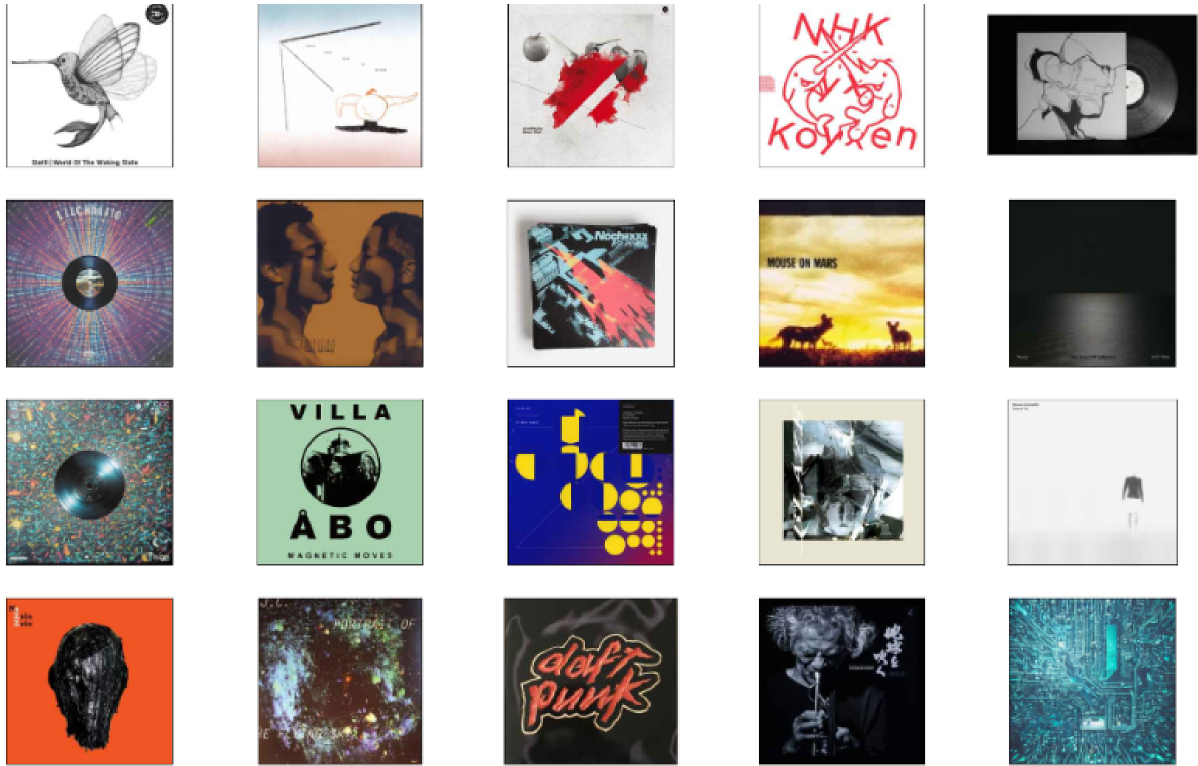


*Figure 9. Electronic training images*

- **Rock:**

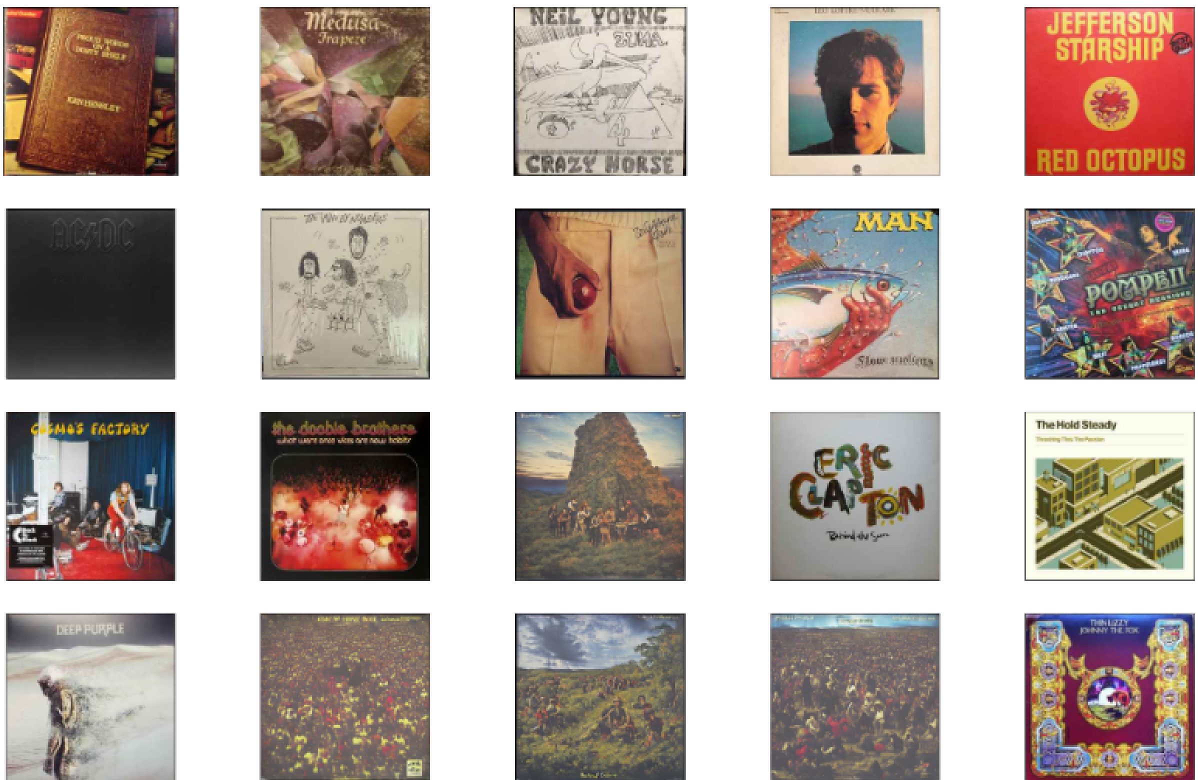

*Figure 10. Rock training images*

- **Hip-hop:**

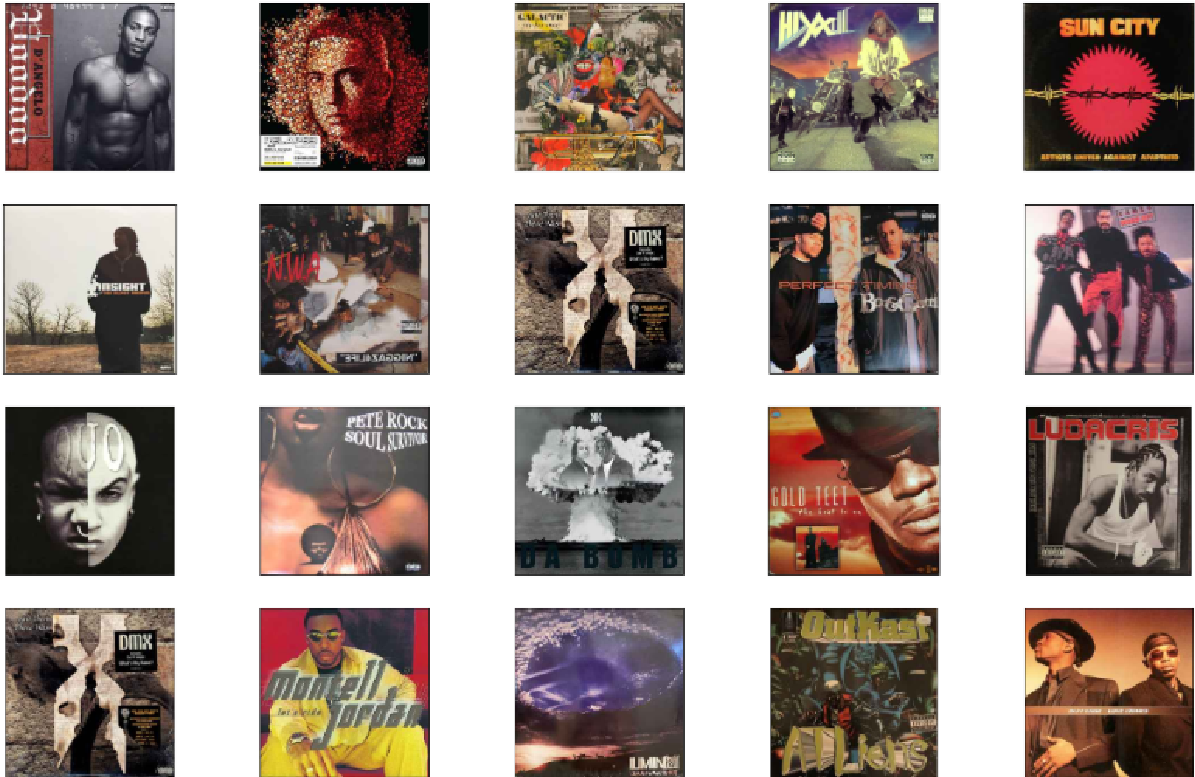


*Figure 11. Hip-hop training images*

After visualizing training images for each genre, the study proceeded to data generation. ImageDataGenerators were used to prepare the data, setting the stage for the upcoming steps in model training and analysis.

ImageDataGenerators serve as efficient tools for data augmentation and preprocessing, crucial for feeding the data into neural networks. The ImageDataGenerator class within TensorFlow is particularly useful for these tasks. Here, the rescale parameter normalizes pixel values to the range [0,1], which is a widespread practice to facilitate model training.

```
Found 2223 images belonging to 3 classes.
Found 422 images belonging to 3 classes.
data batch shape: (20, 300, 300, 3)
labels batch shape: (20, 3)
```

*Figure 12. Data/label batches shapes*

## 4.3. Constructing the Model Without Concept Whitening (bn_model)

The foundational architecture of this study was based on a prior project from the "Advanced Topics in Predictive Analytics" course [29]. In that project, we used VGG19, a model pre-trained on the extensive ImageNet dataset. Using pre-trained models like VGG19 and ResNet152V2, which was also used in this study, provided a head start, as they are trained on large, diverse datasets. Utilizing such a model as a starting point for a new task accelerates the training process and often results in improved performance. By employing these pre-trained models, the study gained a head start in feature identification, significantly contributing to the model's efficiency and efficacy. The study was further refined through a sequence of experiments, which included testing various optimizers and incorporating additional pre-trained models.

Following the series of experiments to fine-tune the model's architecture, we will now investigate the specific code snippets and various layers that were selected for the final model.

**Explanation of Key Components:**

1. **ResNet152V2:** Functions as the underlying architecture of the model. Pretrained on a large dataset, it serves to extract initial features and patterns, reducing the training time and potentially improving the model's performance on the given task.
2. **GlobalMaxPooling2D:** Simplifies the output from the ResNet layer by reducing its dimensions, focusing on the most significant data to optimize computational efficiency.

3. **Dense Layer with 512 Units and ReLU Activation:** Introduces a fully connected layer composed of 512 neurons. The ReLU (Rectified Linear Unit) activation function is applied to introduce non-linearity, allowing for more complex decision boundaries.
4. **Dropout 0.3:** Applies a dropout layer to randomly nullify approximately 30% of the layer's input units during each training iteration. This action serves as a regularization technique to prevent overfitting.
5. **Dense Layer with 256 Units and ReLU Activation:** Adds another fully connected layer with 256 neurons, with ReLU activation to introduce additional non-linearity.
6. **Dropout 0.3:** Incorporates a second dropout layer, also set to a rate of 30%, to further minimize the risk of the model overfitting to the training data.
7. **Dense Layer with 128 Units and ReLU Activation:** Inserts a third fully connected layer with 128 neurons and applies the ReLU activation function, further enriching the model's capability to learn complex patterns.
8. **Dropout 0.3:** Includes a final dropout layer, maintaining a 30% dropout rate to aid in the model's ability to generalize well to new, unseen data.
9. **Dense Layer with 3 Units and SoftMax Activation:** Culminates in an output layer featuring three neurons, each corresponding to a music genre. SoftMax activation is applied to transform the model's output into a probability distribution across the classes, aiding in the final classification task.

Having established the foundational architecture, the next step was to configure various settings to fine-tune the model's training process. This involved defining a learning rate scheduler function to adjust the learning rate as training progresses. Callback functions were also set up to monitor training metrics and adjust accordingly.

Here is a look at the specific configurations:

- **Learning Rate Scheduler Function:** This function dynamically adjusts the learning rate as the model progresses through epochs. Meaning the learning rate reduces by a factor of 0.1 after the 10th and 20th epochs. This strategy assists in fine-tuning the model.

- **Learning Rate Scheduler Callback:** This setting applies the Learning Rate Scheduler Function at the end of each epoch, ensuring that the learning rate is appropriately adjusted.
- **Reduce Learning Rate on Plateau:** This strategy reduces the learning rate if the validation accuracy does not improve for 8 consecutive epochs. The learning rate is reduced by multiplying it with a factor of 0.6. This aids the model in overcoming plateaus in the learning curve.
- **Model Checkpoint:** This setting saves the best-performing version of the model, as judged by its validation accuracy. This is crucial for both avoiding overfitting and ensuring that the best version of the model is retained.
- **Early Stopping:** This mechanism halts the training process if there is no improvement in the model's validation accuracy for 5 consecutive epochs. It serves to prevent overfitting and to minimize computational cost.

These settings ensure that the model can adapt throughout training, preserving the best versions and stopping if gains plateau.

After these settings were applied, the model was prepared for training.

Initially, computational constraints prompted the use of Google Collab, which offers enhanced computational resources. However, the final architecture of the model, combined with the manageable size of the dataset, allowed for effective local training, even on a MacBook Pro.

- **Optimizer:** The model used RMSprop with a learning rate of 0.00001. RMSprop is suitable for scenarios where the loss landscape changes dynamically, known as non-stationary objectives.
- **Loss Function:** The model employed 'categorical_crossentropy' as its loss function, a common choice for multi-class classification tasks. This function quantifies the accuracy of the model in classifying album covers into their respective genres.
- **Metrics:** The metric chosen for model evaluation was accuracy, which aligns with the study's goal to correctly classify album covers.

- **Epochs:** The model was set to train for 10 epochs. Early stopping data indicated that performance gains plateaued around the 7th or 8th epoch.
- **Validation Data:** The validation generator was used to provide data for model evaluation. The number of steps for validation was equal to the length of the validation generator, ensuring each sample was evaluated once.

Following these configurations, the model was compiled and trained, marking the final preparatory steps before the evaluation phase.

## 4.4. Constructing the Model with Concept Whitening (cw_model)

Building upon the foundational architecture and configurations used in bn_model, this study also explored the utilization of Concept Whitening for enhanced model interpretability. The technique aims to provide clearer insight into the model's decision-making process by disentangling the learned representations in the neural network layers. The method is inspired by the paper "Concept Whitening for Interpretable Image Recognition" by Zhi Chen, Yijie Bei, and Cynthia Rudin [13].

The code for Concept Whitening was adapted from a GitHub repository that includes the implementation of the IterNormRotation class.
The dataset structure followed the recommended hierarchy; for the auxiliary concept dataset (MS COCO objects), the data was organized in different splits /concept_train and /concept_test.

**Custom Keras Layer**

**Explanation of Key Components**

- **Epsilon:** A small constant added to the eigenvalues for numerical stability.

- **Covariance Matrix:** Captures the correlations between unique features in the activations.
- **Whitening Matrix:** A matrix that transforms the correlated features into a set of uncorrelated features.
- **Whitened Inputs:** The transformed activations that are now more interpretable.

After successfully implementing the Concept Whitening layer, we incorporated it into our existing deep learning architecture, initially constructed as the Model Without Concept Whitening (bn_model).

The original bn_model utilized a pre-trained ResNet152V2 architecture, as described in earlier steps. In this stage, we added the Concept Whitening layer right after the last layer of the ResNet152V2 architecture. This strategic placement allows us to leverage the feature representations learned by the pre-trained model while also providing a mechanism to understand the latent features more transparently.

This integration prepares the model for the subsequent phases of training and evaluation, setting the stage for a more comprehensive understanding of the model's decision-making process.

**Training with Concept Whitening**

For training with Concept Whitening, a similar configuration to the one used in the bn_model can be employed. However, the architecture is specified as 'resnet_cw' to include the Concept Whitening layer. The layer is specifically activated after 'conv5_block3_preact_cw'.

The same settings for learning rate adjustment, callbacks, and other configurations are maintained as in the bn_model.
This configuration ensures that the model with Concept Whitening is trained effectively, while also being prepared for detailed evaluation.

For model testing and visualization, the top-activated images along each concept axis are presented to display the model's focus on specific features.

## 4.5. Comparison Analysis

In this section, we transition into the final phase of our research: the comparison analysis of the two models. Following the structured methodology previously outlined, this part focuses on evaluating and contrasting the performance and interpretability of both convolutional neural networks.

### 4.5.1. Model Without Concept Whitening (bn_model)

#### Metrics and Performance

To evaluate the bn_model, we retrieved its accuracy and loss metrics over epochs from the model's history.

These metrics were then plotted to visualize the model's training and validation accuracy and loss:

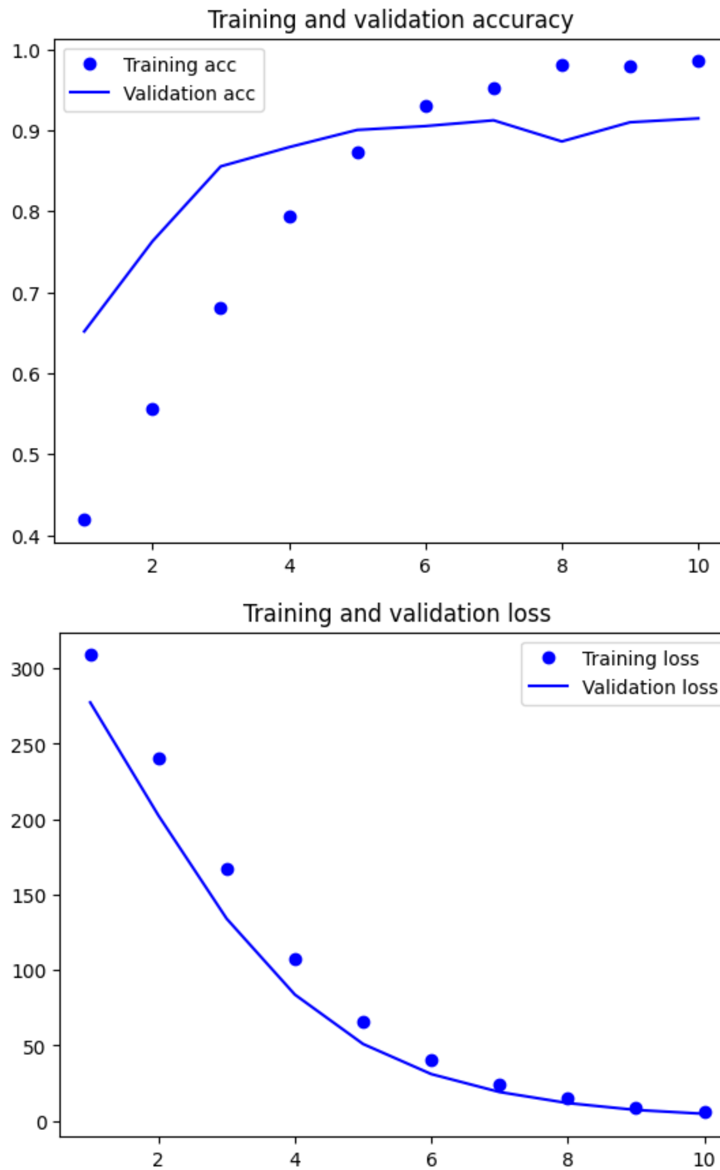


*Figure 13. bn_model performance results*

## 4.5.2. Model with Concept Whitening (cw_model)

**Metrics and Performance**

Similarly, the cw_model's performance was assessed using its accuracy and loss metrics. These metrics were plotted in the same manner as the bn_model:

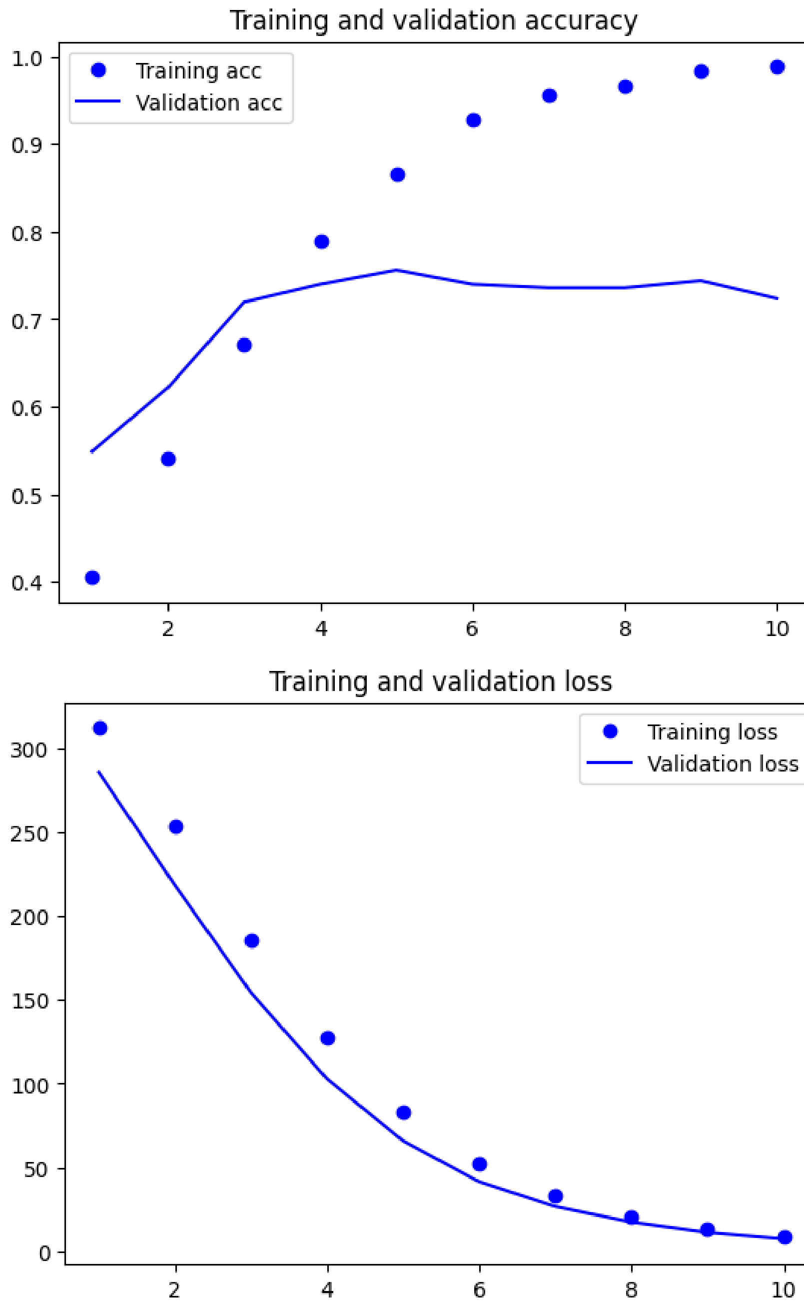


*Figure 14. cw_model performance results*

**Interpretability Assessment**

After training the model with Concept Whitening (cw_model), the next crucial step was to evaluate its interpretability performance and understand how the Concept Whitening layer influenced the model's decision-making process. Specifically, we are interested in visualizing

the top activated images along the concept axes. For this purpose, the top 5 images for each axis are selected and visualized.

- **Electronic:**

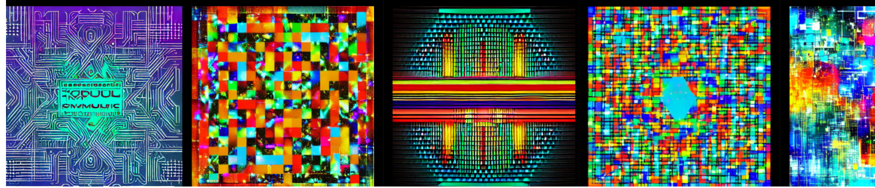

*Figure 15. Electronic top activations*

- **Rock:**

*Figure 16. Rock top activations*

- **Hip-hop Genre:**

*Figure 17. Hip-hop top activations*

By visualizing the top activated images, we can gain insights into how the model perceives and aligns with specific concepts, thereby improving our understanding of its internal workings. This forms an integral part of our study, bridging the gap between high performance and model interpretability.

### 4.5.3. Comparative Visualization

To make a side-by-side comparison of both models, their training and validation accuracy and loss metrics were plotted using Matplotlib.

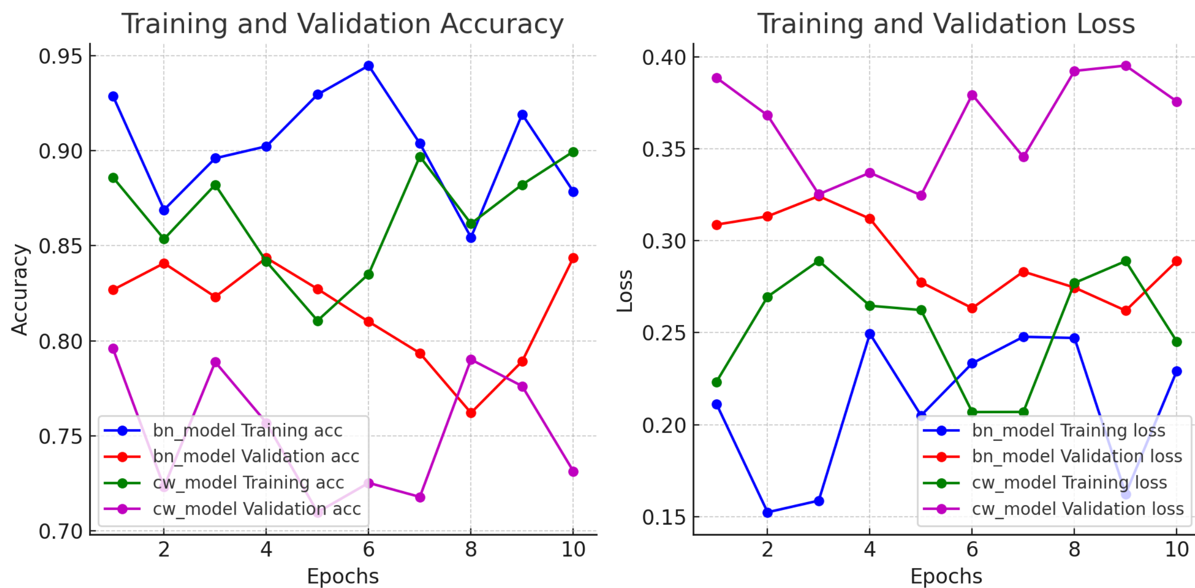


*Figure 18. Crossed performance results*

Through this analysis we gained insights into how the batch normalization and concept whitening models performed in the task of music genre classification based on album artwork. After the visual comparison of key metrics, both models clearly show commendable performance. The batch normalization model shows a slight edge in this scenario, but the concept whitening model also holds its ground, making it a viable option for those interested in model interpretability.

All scripts, code, and Jupyter notebooks supporting this research are available in the GitHub repository titled "Image-Based-Music-Genre-Classification-Using-Convolutional-Neural-Networks".

# 4.6. Deployment

In this final phase, the focus is on the deployment of both models, bn_model and cw_model, to evaluate their generalizability on unseen data. The deployment stage involved saving the models and testing them on new, unseen data to ensure their robustness and reliability.

## 4.6.1. Deployment of Model Without Concept Whitening (bn_model)

The trained bn_model is saved to disk for future use, using the filename 'bn_model.h6'.

The test images are loaded into a list from a specified directory.

A Pandas DataFrame is initialized to store the prediction results. Three categories—'electronic,' 'rock,' and 'hiphop'—are defined for classification.

A loop iterates through each test image, processing it and using bn_model for prediction. The predicted category is then printed and stored in the DataFrame.

```
INFO:tensorflow:Assets written to: bn_model.h6/assets
1/1 [==============================] - 2s 2s/step
image_2305.jpeg is hiphop
1/1 [==============================] - 0s 100ms/step
image_1755.jpeg is electronic
1/1 [==============================] - 0s 100ms/step
image_2368.jpeg is hiphop
1/1 [==============================] - 0s 100ms/step
image_1803.jpeg is electronic
1/1 [==============================] - 0s 102ms/step
image_2410.jpeg is hiphop
1/1 [==============================] - 0s 100ms/step
image_986.jpg is electronic
1/1 [==============================] - 0s 100ms/step
image_2387.jpeg is rock
1/1 [==============================] - 0s 99ms/step
```

*Figure 19. bn_model deployment*

The steps for deploying the cw_model are analogous to those for bn_model, with the model being saved as 'cw_model.h6'.

Both the bn_model and the cw_model were deployed using the same procedures and saved under unique filenames for future inference. Prediction results for each model were logged in individual CSV files. This deployment method ensures the models' real-world applicability and reliability. It also lays the groundwork for future research into balancing model performance and interpretability.

# 5. Main Conclusion and Limitations

This study lays the foundation for future research into balancing model effectiveness with transparency. The project demonstrates the feasibility of using concept whitening for

interpretable image-based music genre classification. Both the standard CNN model and its interpretable counterpart were trained on a dataset of 3,000 album covers in electronic, rock, and hip-hop genres.
The standard CNN model achieved an accuracy of ~87% on the test set. However, while performance was strong, the model lacked interpretability into how genre classifications were made. In contrast, the interpretable CNN model with concept whitening achieved a slightly lower accuracy of ~74% but provided enhanced transparency into the latent features used for prediction. The concept whitening technique decorrelated activations in the final convolutional layer, making the associations between visual concepts and predicted genres more explicit. For example, electronic covers were strongly associated with abstract geometric shapes, while rock covers had higher activations for images of musical instruments. This allows better understanding of the model's reasoning.

A key limitation is the small dataset size and simplicity of the CNN architectures. In future work, more complex state-of-the-art networks could be explored, trained on larger datasets, to improve accuracy. Additionally, more rigorous quantitative evaluation of interpretability could be undertaken. As the field progresses, the question shifts from merely evaluating performance to understanding the balance between interpretability and effectiveness. This study lays the groundwork for such inquiries, suggesting that future research could beneficially focus on improving interpretability metrics to more comprehensively assess the trade-offs involved.

In conclusion, while both models performed well, both models have limitations due to the dataset size and their simple architectures. The interpretable CNN provided unique insights into the model's predictions by disentangling the learned representations, and the standard model achieved particularly good accuracy but lacked transparency in its decision-making process. This transparency may be crucial for trustworthy and explainable AI systems. Therefore, Future research should consider using more complex models and larger datasets while focusing on improving interpretability metrics.

# 6. Appendices

## 6.1. Glossary

- **Dimensionality:** dimensionality refers to the number of dimensions or independent variables in a system or dataset. It can be used in various fields such as mathematics, physics, and computer science to describe the number of parameters needed to fully describe or represent a system or dataset. For example, a two-dimensional dataset would require two variables (such as x and y coordinates) to plot or visualize the data, while a three-dimensional dataset would require three variables (such as x, y, and z coordinates) to plot or visualize the data in 3D space.
- **Latent Space:** a high-dimensional space that encodes a meaningful internal representation of observed events, such as images. This space is constructed by latent variables that capture essential features of the data, allowing for a more efficient and interpretable representation. Samples that are similar in the external world are positioned close to each other in this space, enabling efficient generalization and reducing noise and redundancy.
- **Sparsity:** sparsity refers to the property of having a small number of active (non-zero) elements in each set or representation. In ML sparsity can refer to the number of activated filters in the network in response to input data.
- **Image Feature Space:** a latent space that captures high-level features of images, such as edges, shapes, and colors. It is often used in convolutional neural networks (CNNs) to classify images, where similar images are positioned closer to each other in space based on their shared features.
- **Disentanglement:** refers to the ability of a machine learning model to identify and separate underlying factors of variation in the data, such as distinctive features or causes, in an unsupervised way.

- **Whitening:** a linear transformation that transforms the covariance matrix of random input vectors to be the identity matrix.
- **Normalization:** a technique used in deep learning to standardize input data to improve model training and performance. The process involves rescaling the features to a similar scale, often to have a mean of zero and standard deviation of one.
- **Standarization:** is the process of transforming data so that it has a mean of 0 and a standard deviation of 1, allowing for fair comparison between variables with different units of measurement.
- **Regularization:** is a technique used in machine learning to prevent overfitting by adding a penalty term to the loss function.
- **Batch Normalization layer:** is a module in a neural network that helps to normalize the inputs by adjusting and scaling the values before they are passed to the activation function. It is a technique in deep learning that normalizes the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation.
- **Deep Residual Learning:** residual learning is a technique used to improve the training of deep neural networks by adding shortcut connections that allow information to be passed through the network without going through every layer.
- **PCA (Principal Component Analysis):** it is a dimensionality reduction technique used to identify patterns in data and to represent it in a more concise form. The basic idea is to reduce the number of variables (features) in the dataset while retaining as much information as possible.
- **Features (variables):** a feature is an input variable used in a machine learning model to make a prediction. A feature can be any measurable data property relevant to the task at hand.
- **Feature Engineering:** is the process of selecting, transforming, and scaling the input data to create features that are optimal for the task at hand.
- **Encoding:** encoding refers to the process of converting data from one format or representation to another. This is often necessary when working with data that needs to be stored or transmitted in a way that can be interpreted by different systems or applications.
- **Embeddings:** embeddings are a way of representing words, phrases, or other types of data as vectors of numerical values.

- **Loss Function:** A loss function in machine learning is a measure of how well the model's predictions match the actual data. During training, the aim is to minimize this function, which would imply better model performance.
- **Epoch:** In the context of training a machine learning model, an epoch is one complete forward and backward pass of all the training examples. The model's parameters are updated after each epoch. The number of epochs is a hyperparameter that defines the number of times the learning algorithm will work through the entire training dataset.
- **Categorical Cross-Entropy:** It is a type of loss function commonly used for classification problems where the task is to categorize inputs into two or more classes. In the context of neural networks, it quantifies the difference between the predicted probabilities and the actual class labels, providing a measure of the model's performance.

## 6.2. Bibliography


**[1]** Chen, Z., Bei, Y., & Rudin, C. (2020). Concept whitening for interpretable image recognition. Nature Machine Intelligence, 2(12), 772–782. https://arxiv.org/abs/2002.01650

**[2]** Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence, 1(5), 206–215. https://arxiv.org/abs/1811.10154

**[3]** He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778). https://arxiv.org/abs/1512.03385

**[4]** Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., & Zhong, C. (2021, July 10). Interpretable machine learning: Fundamental principles and 10 grand challenges. arXiv.org. https://arxiv.org/abs/2103.11251

**[5]** Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167

**[6]** Zheng Z. (2022). The Classification of Music and Art Genres under the Visual Threshold of Deep Learning. Computational intelligence and neuroscience, 2022, 4439738. https://doi.org/10.1155/2022/4439738

**[7]** Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105). Curran Associates, Inc.

**[8]** Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2019, April 15). How does batch normalization help optimization? arXiv.org. https://arxiv.org/abs/1805.11604

**[9]** LeCun, Y., Bengio, Y., & Hinton, G. (2015, May 27). Deep learning. Nature News. https://www.nature.com/articles/nature14539

**[10]** Mohammed, E. U. R., Reddy, S. N., & Waseem, M. S. (2022, December 20). A comprehensive literature review on Convolutional Neural Networks. figshare. https://www.techrxiv.org/articles/preprint/A_Comprehensive_Literature_Review_on_Convolutional_Neural_Networks/21746237

**[11]** Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press. http://www.deeplearningbook.org

**[12]** Web scraping in python: Avoid detection like a ninja - zenrows. (2020, October 12). https://www.zenrows.com/blog/stealth-web-scraping-in-python-avoid-blocking-like-a-ninja
**[13]** zhiCHEN96. (2022, March 15) Zhichen96/conceptwhitening. GitHub. https://github.com/zhiCHEN96/ConceptWhitening
**[14]** Gretel.ai platform API.
https://image-synthetics-preview.gretel.cloud/
**[15]** ChatGPT chat API.
https://chat.openai.com/c/d6f51336-8248-49d4-8fdd-56db700bbc53
**[27]** Beautifulsoup4. PyPI. (2023, April 7). https://pypi.org/project/beautifulsoup4/
**[30]** Gijsbrechts, J. (2022). Advanced Topics in Predictive Analytics. Lisboa; Universidade Católica Portuguesa.

## 6.3. Codebase

Interpretable Music Genre Classification Using Whitened Artwork Concepts

requirements.txt

In [ ]:

```
# Importing libraries
import os
import random
import numpy as np
import tensorflow as tf
import math
import matplotlib.pyplot as plt
import shutil
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers
from tensorflow.keras import models
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import LearningRateScheduler
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
import keras
```

In [ ]:

```
# Set matplotlib to inline mode
%matplotlib inline

# Seed value definition
def set_seed(seed=42):
    """Set seed for reproducibility."""
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

# Call the function with the desired seed
set_seed(42)
```

dataset_preparation.py

In [ ]:

```
# Specify paths to the datasets
original_dataset_directory = "/content/drive/MyDrive/documentos/vinyl genre class/data/For thesis"
original_train_directory = os.path.join(original_dataset_directory, 'Training')
```

In [ ]:

```
# 0 for electronic, 1 for rock, 2 for hiphop
# Initialize lists to store genres and image files
genres = []
img_files= []

# List all files in the training directory
imgs = os.listdir(original_train_directory)

# Load training labels from CSV
training_lbls = pd.read_csv('/content/drive/MyDrive/documentos/vinyl genre class/data/For
thesis/training_labels.csv')

# Loop through all labels and append corresponding genres and image files to the lists
for k in range(len(training_lbls)):
    for file in imgs:
        if file == training_lbls['name'][k]:
            genres.append(training_lbls['category'][k])
            img_files.append(file)
```

```
# Create a dataframe to map each image with its label
df = pd.DataFrame({'image': img_files,'category':genres})

# Display the first few rows of the dataframe
df.head(min(3000, len(df)))
```

In [ ]:

```
# Define directories for train, test and validation datasets
base_dir = "./data_split"
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')
validate_dir = os.path.join(base_dir, 'validate')

# Create directories for train, test and validation datasets if they don't exist
for directory in [base_dir, train_dir, test_dir, validate_dir]:
    if not os.path.exists(directory):
        os.mkdir(directory)
        print(directory, "created")

# Define directories for each genre in train, test and validation datasets
subdirectories_electronic = []
subdirectories_rock = []
subdirectories_hiphop = []

# Create directories for each genre in train, test and validation datasets if they don't exist
for subdirectory in [train_dir, test_dir, validate_dir]:
    subdirectories_electronic.append(os.path.join(subdirectory,'electronic'))
    if not os.path.exists(os.path.join(subdirectory,'electronic')):
        os.mkdir(os.path.join(subdirectory,'electronic'))
        print(os.path.join(subdirectory,'electronic'),"created")

    subdirectories_rock.append(os.path.join(subdirectory, 'rock'))
    if not os.path.exists(os.path.join(subdirectory, 'rock')):
        os.mkdir(os.path.join(subdirectory, 'rock'))
        print(os.path.join(subdirectory, 'rock'), "created")

    subdirectories_hiphop.append(os.path.join(subdirectory,'hiphop'))
    if not os.path.exists(os.path.join(subdirectory,'hiphop')):
        os.mkdir(os.path.join(subdirectory,'hiphop'))
        print(os.path.join(subdirectory,'hiphop'),"created")
```

In [ ]:

```
# Split the data into train, test and validation sets
x_train, x_test1, y_train, y_test1 = train_test_split(df["image"], df["category"] ,test_size=0.2,
random_state=42)
x_test, x_val, y_test, y_val = train_test_split(x_test1, y_test1 ,test_size=0.5, random_state=42)

# Print the size of each set
print('Training data set size :', y_train.shape[0])
print('Test data set size :', y_test.shape[0])
print('Validation data set size :', y_val.shape[0])
```

In [ ]:

```
# Copy the files into the corresponding genre directories in train, test and validation datasets
x = [x_train,x_test,x_val]
y = [y_train,y_test,y_val]
k = 0

for images,genres in zip(x,y) :
    for (file,category) in zip(images,genres):
        if category == 'electronic':
            src = os.path.join(original_train_directory, file)
            dst = os.path.join(subdirectories_electronic[k], file)
            shutil.copyfile(src, dst)
        elif category == 'rock':
            src = os.path.join(original_train_directory, file)
            dst = os.path.join(subdirectories_rock[k], file)
            shutil.copyfile(src, dst)
        else:
            src = os.path.join(original_train_directory, file)
            dst = os.path.join(subdirectories_hiphop[k], file)
            shutil.copyfile(src, dst)

    print(len(os.listdir(subdirectories_electronic[k])), " electronic covers copied to:",
subdirectories_electronic[k])
    print(len(os.listdir(subdirectories_rock[k])), " rock covers copies to:", subdirectories_rock[k])
    print(len(os.listdir(subdirectories_hiphop[k])), " hiphop covers copied to:",
subdirectories_hiphop[k])

    k = k + 1
```

In [ ]:

```
# Visualize training electronic images examples
plt.figure(figsize=(25,20))
plt.suptitle("Train Electronic Images", fontsize=20)
```

```
images = os.listdir(subdirectories_electronic[0])
for i in range(len(images)-20, len(images)):
   plt.subplot(5,5,i-len(images)+20+1)

   full_image = plt.imread(os.path.join(original_train_directory, images[i]))
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(full_image, cmap=plt.cm.binary)
```

In [ ]:

```
# Visualize training rock images examples
plt.figure(figsize=(25,20))
plt.suptitle(" Train Rock Images", fontsize=20)

images = os.listdir(subdirectories_rock[0])
for i in range(len(images)-20, len(images)):
   plt.subplot(5,5,i-len(images)+20+1)

   full_image = plt.imread(os.path.join(original_train_directory, images[i]))
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(full_image, cmap=plt.cm.binary)
```

In [ ]:

```
# Visualize training hiphop images examples
plt.figure(figsize=(25,20))
plt.suptitle("Train HipHop images", fontsize=20)

images = os.listdir(subdirectories_hiphop[0])
for i in range(len(images)-20, len(images)):
   plt.subplot(5,5,i-len(images)+20+1)

   full_image = plt.imread(os.path.join(original_train_directory, images[i]))
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(full_image, cmap=plt.cm.binary)
```

In [ ]:

```
# Create ImageDataGenerators for training and validation datasets
# ImageDataGenerators are used to generate batches of tensor image data with real-time data
augmentation
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Define target size for resizing images and batch size
# Class mode is set to 'categorical' for multi-class classification
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(300,300),
        batch_size=20,
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validate_dir,
        target_size=(300,300),
        batch_size=20,
        class_mode='categorical')

# Print the shapes of data and labels batches to confirm they are as expected
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

model_without_whitening.py

In [ ]:

```
# Import ResNet152v2 pre-trained model with ImageNet weights
from tensorflow.keras.applications import ResNet152V2

# 1. ResNet152V2: Utilizes a pre-trained architecture to initialize weights, aiding in feature extraction.
pre_trained_model = ResNet152V2(weights='imagenet', include_top=False, input_shape=(300, 300,
3))
pre_trained_model.summary()
```

In [ ]:

```
# Set all pre-trained model layers to non-trainable
for layer in pre_trained_model.layers:
    layer.trainable = False

# Define the custom neural network architecture
```

```
last_layer = pre_trained_model.get_layer('post_relu')
last_output = last_layer.output

# 2. GlobalMaxPooling2D: Reduces the spatial dimensions of the output volume.
x = tf.keras.layers.GlobalMaxPooling2D()(last_output)

# 3. Dense 512 and ReLU Activation: Fully connected layer with 512 neurons and ReLU activation for non-linearity.
x = tf.keras.layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l1(0.01))(x)

# 4. Dropout 0.3: Prevents overfitting by randomly setting a fraction of input units to 0 during training.
x = tf.keras.layers.Dropout(0.3)(x)

# 5. Dense 256 and ReLU Activation: Another fully connected layer with 256 neurons and ReLU activation.
x = tf.keras.layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l1(0.01))(x)

# 6. Dropout 0.3: Another dropout layer for regularization.
x = tf.keras.layers.Dropout(0.3)(x)

# 7. Dense 128 and ReLU Activation: Further fully connected layer with 128 neurons for feature transformation.
x = tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l1(0.01))(x)

# 8. Dropout 0.3: Final dropout layer to counteract overfitting.
x = tf.keras.layers.Dropout(0.3)(x)

# 9. Dense 3 and Softmax Activation: Output layer with 3 neurons corresponding to the classes, with softmax activation for probability distribution.
x = tf.keras.layers.Dense(3, activation='softmax')(x)

# Combine the pre-trained model with the additional layers to complete the architecture
bn_model = tf.keras.Model(pre_trained_model.input, x)
bn_model.summary()
```

In [ ]:

```
# Open trainable layers
pre_trained_model.trainable = True
set_trainable = False
for layer in pre_trained_model.layers:
    if layer.name == 'conv5_block3_preact_bn':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
```

```
    else:
        layer.trainable = False

# Define the learning rate scheduler function
def lr_scheduler(epoch):
    lr = 1e-4
    if epoch > 10:
        lr *= 0.1
    if epoch > 20:
        lr *= 0.1
    return lr

# Set up callback functions
lr_scheduler_callback = LearningRateScheduler(lr_scheduler)
lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.6, patience=8, verbose=1,
mode='max', min_lr=1e-4)
checkpoint = ModelCheckpoint('bn_finetune.h16', monitor='val_accuracy', mode='max',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=5, verbose=1,
mode='max')

# Compile the model
# This step prepares the model for training.
# It specifies the optimizer to update model weights, the loss function to evaluate performance,
# and the metric to monitor during training.
bn_model.compile(
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),  # RMSprop optimizer with a learning
rate of 1e-4
    loss='categorical_crossentropy',  # categorical cross-entropy for multi-class classification
    metrics=['accuracy']  # Monitoring accuracy during training
)

# Train the model
# This step starts the training process by feeding the data into the model.
bn_history = bn_model.fit(
    train_generator,  # Using the training data generator
    epochs=10,  # Training for 10 complete passes through the training dataset
    validation_data=validation_generator,  # Using the validation data generator
    validation_steps=len(validation_generator),  # Number of batches to draw from the validation
generator for evaluation
    callbacks=[lr_reduce, checkpoint, early_stopping, lr_scheduler_callback]  # Applying various
callback functions
)
```

model_with_whitening.py

In [ ]:

```
#  Create custome Keras layer for concept whitening
class ConceptWhitening(tf.keras.layers.Layer):
    def __init__(self, epsilon=1e-5, **kwargs):
        super(ConceptWhitening, self).__init__(**kwargs)
        self.epsilon = epsilon

    def build(self, input_shape):
        super(ConceptWhitening, self).build(input_shape)

    def call(self, inputs):
        # Compute the covariance matrix of the input activations
        mean = tf.reduce_mean(inputs, axis=[0, 1, 2], keepdims=True)
        centered_inputs = inputs - mean
        cov_matrix = tf.reduce_mean(tf.einsum('bijc,bijd->bcd', centered_inputs, centered_inputs),
axis=0)

        # Compute the whitening matrix
        s, u, v = tf.linalg.svd(cov_matrix)
        whitening_matrix = tf.einsum('bi,bj->bij', tf.math.rsqrt(tf.reshape(s, [-1, 1]) + self.epsilon), u)

        # Squeeze out the singleton dimension
        whitening_matrix = tf.squeeze(whitening_matrix, axis=1)

        # Apply the whitening transformation
        whitened_inputs = tf.einsum('bijc,bd,de->bije', centered_inputs, u, whitening_matrix)

        # Debugging: Print the shape of the centered inputs for verification
        print("Shape of centered_inputs:", tf.shape(centered_inputs))

        # Debugging: Print the shape of 'u' to ensure it's as expected
        print("Shape of u:", tf.shape(u))

        # Debugging: Print the shape of the whitening matrix for validation
        print("Shape of whitening_matrix:", tf.shape(whitening_matrix))

        return whitened_inputs

    def get_config(self):
        config = super(ConceptWhitening, self).get_config()
        config.update({"epsilon": self.epsilon})
        return config
```

In [ ]:

```
# Import ResNet152v2 pre trained model with imagenet weights
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras import regularizers

# Import your pre-trained model
pre_trained_model = ResNet152V2(weights='imagenet', include_top=False, input_shape=(300, 300,
3))

# Make all layers non-trainable
for layer in pre_trained_model.layers:
    layer.trainable = False

# Get the last layer's output
last_layer = pre_trained_model.get_layer('post_relu')
last_output = last_layer.output

# Insert the Concept Whitening layer
x = ConceptWhitening()(last_output)

# Add your additional layers
x = tf.keras.layers.GlobalMaxPooling2D()(x)
x = tf.keras.layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l1(0.01))(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = tf.keras.layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l1(0.01))(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l1(0.01))(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = tf.keras.layers.Dense(3, activation='softmax')(x)

# Create the final model
cw_model = tf.keras.Model(pre_trained_model.input, x)
cw_model.summary()
```

In [ ]:

```
# Open trainable layers
pre_trained_model.trainable = True
set_trainable = False
for layer in pre_trained_model.layers:
    if layer.name == 'conv5_block3_preact_cw':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

```
# Define the learning rate scheduler function
def lr_scheduler(epoch):
    lr = 1e-4
    if epoch > 10:
        lr *= 0.1
    if epoch > 20:
        lr *= 0.1
    return lr

# Set up callback functions
lr_scheduler_callback = LearningRateScheduler(lr_scheduler)
lr_reduce = ReduceLROnPlateau(monitor='val_accuracy', factor=0.6, patience=8, verbose=1,
mode='max', min_lr=1e-4)
checkpoint = ModelCheckpoint('cw_finetune.h16', monitor='val_accuracy', mode='max',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=5, verbose=1,
mode='max')

# Compile the model
cw_model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
            loss='categorical_crossentropy',
            metrics=['accuracy'])

# Train the model
cw_history = cw_model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
    callbacks=[lr_reduce, checkpoint, early_stopping, lr_scheduler_callback])
```

comparison_analysis.py

In [ ]:

```
# Retrieve accuracy and loss from the bn_model's history
acc = bn_history.history['accuracy']
val_acc = bn_history.history['val_accuracy']
loss = bn_history.history['loss']
val_loss = bn_history.history['val_loss']

# Create a range of epochs to use in the plot
epochs = range(1, len(acc) + 1)
```

```
# Plot training and validation accuracy
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

# Plot training and validation loss
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

In [ ]:

```
# Retrieve accuracy and loss from the cw_model's history
acc = cw_history.history['accuracy']
val_acc = cw_history.history['val_accuracy']
loss = cw_history.history['loss']
val_loss = cw_history.history['val_loss']

# Create a range of epochs to use in the plot
epochs = range(1, len(acc) + 1)

# Plot training and validation accuracy
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

# Plot training and validation loss
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

In [ ]:

```
# Extract the metrics from the history dictionaries
bn_acc = bn_history['accuracy']
bn_val_acc = bn_history['val_accuracy']
bn_loss = bn_history['loss']
bn_val_loss = bn_history['val_loss']

cw_acc = cw_history['accuracy']
cw_val_acc = cw_history['val_accuracy']
cw_loss = cw_history['loss']
cw_val_loss = cw_history['val_loss']

epochs = range(1, len(bn_acc) + 1)

# Create the plots
plt.figure(figsize=(10, 5))

# Plot for accuracy
plt.subplot(1, 2, 1)
plt.plot(epochs, bn_acc, 'bo-', label='bn_model Training acc')
plt.plot(epochs, bn_val_acc, 'ro-', label='bn_model Validation acc')
plt.plot(epochs, cw_acc, 'go-', label='cw_model Training acc')
plt.plot(epochs, cw_val_acc, 'mo-', label='cw_model Validation acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot for loss
plt.subplot(1, 2, 2)
plt.plot(epochs, bn_loss, 'bo-', label='bn_model Training loss')
plt.plot(epochs, bn_val_loss, 'ro-', label='bn_model Validation loss')
plt.plot(epochs, cw_loss, 'go-', label='cw_model Training loss')
plt.plot(epochs, cw_val_loss, 'mo-', label='cw_model Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

deploy

In [ ]:

```
# Save the model (bn_model) without concept whitening
bn_model.save('bn_model.h6')

# Load test images
images_test = os.listdir("/content/drive/MyDrive/documentos/vinyl genre class/data/For
thesis/data/For thesis/Testing")

# Initialize a dataframe to store test results
df_test = pd.DataFrame(columns=['category', 'name'])
categories = ['electronic','rock','hiphop']

# Loop through all test images, predict their category and append results to the dataframe
for image in images_test:
    if image.endswith(".jpg") is True or image.endswith(".jpeg") is True:
        img = tf.keras.utils.load_img (
            f"/content/drive/MyDrive/documentos/vinyl genre class/data/For thesis/data/For
thesis/Testing/{image}", target_size=(300, 300)
        )
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0)
        img_array /= 255.

        preds = bn_model.predict(img_array)
        score = preds[0]
        max_indexes = np.argmax(score)

        df_test = pd.concat([df_test, pd.DataFrame({'category': [categories[max_indexes]], 'name':
[image]})], ignore_index=True)
        print(f"{image} is {categories[max_indexes]}")

# Save test results to a CSV file
df_test.to_csv('results_bn_new.csv', header=True)
```

In [ ]:

```
# Save the model (cw_model) with concept whitening
cw_model.save('cw_model.h6')

# Load test images
images_test = os.listdir("/content/drive/MyDrive/documentos/vinyl genre class/data/For
thesis/Testing")

# Initialize a dataframe to store test results
df_test = pd.DataFrame(columns=['category', 'name'])
categories = ['electronic','rock','hiphop']
```

```
# Loop through all test images, predict their category and append results to the dataframe
for image in images_test:
    if image.endswith(".jpg") is True or image.endswith(".jpeg") is True:
        img = tf.keras.utils.load_img (
            f"/content/drive/MyDrive/documentos/vinyl genre class/data/For thesis/data/For thesis/Testing/{image}", target_size=(300, 300)
        )
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0)
        img_array /= 255.

        preds = cw_model.predict(img_array)
        score = preds[0]
        max_indexes = np.argmax(score)

        df_test = pd.concat([df_test, pd.DataFrame({'category': [categories[max_indexes]], 'name': [image]})], ignore_index=True)
        print(f"{image} is {categories[max_indexes]}")

# Save test results to a CSV file
df_test.to_csv('results_cw_new.csv', header=True)
```