POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Machine Learning Cinema: Color grading style classification through Vectorscopes and CNNs

Supervisors

Prof. Tania CERQUITELLI

Prof. Bartolomeo VACCHETTI

Candidate

Krzysztof KLEIST

November 2023

Summary

Identifying a movie's genre from a single frame may pose challenges but is entirely achievable. Various movie genres exhibit unique visual elements and stylistic cues that hint at their identity. For instance, action films often feature fast-paced scenes, intense lighting, and dynamic camera angles. In contrast, horror movies tend to create dark, mysterious settings with unsettling audio cues. Comedies typically present bright, colorful frames with exaggerated expressions and physical humor. Additionally, clothing, set design, and the overall atmosphere of the frame contribute valuable clues to the movie's genre. By examining these distinct components, viewers can make informed guesses about the movie's genre.

Convolutional Neural Networks (CNNs) have found diverse applications in movie classification, including genre prediction. Early research focused on audio-visual cues within movie previews, such as camera movements and sound effects, for mood-based genre analysis. Later, CNNs were applied to movie trailers, achieving higher accuracy. Movie posters have also been explored for genre prediction, with varying degrees of success. Some studies utilized deep learning models, significantly improving accuracy. In addition, text-based approaches, such as predicting movie titles from plot summaries, have been employed, reaching substantial hit rates across genres. These works collectively demonstrate the evolving landscape of using neural networks and machine learning for movie genre classification and attribute prediction.

My research aligns with prior studies that leverage deep learning to classify movie shots or frames based on visual attributes, rather than relying solely on movie posters. These studies demonstrate the effectiveness of deep learning in accurately categorizing movie shots, with some achieving high accuracy rates. Additionally, they explore innovative techniques such as combining multiple models and utilizing data-driven approaches to discover editing patterns within movies. In my research, I aim to extend this line of inquiry by investigating the relationship between genre identification and movie frames while excluding movie posters. Furthermore, I plan to explore the potential advantages of incorporating vectorscope images into existing models to potentially enhance their performance. Through these investigations, I

hope to contribute to the ongoing advancements in genre classification techniques and provide insights into the utility of using movie frames and their vectorscope representations for this purpose.

In my research, acquiring a suitable dataset was crucial. Although online services provide movie data with genre labels, copyright restrictions prevented their use. As a result, I collected and constructed my dataset using the repository, which includes over 3,000 movies with more than 30,000 captioned clips and 1,000 hours of video. From this extensive dataset, I extracted relevant information from tables. I performed a various operations to create the table, which included 27,077 rows with videoid and genre information. To refine the dataset further, I used Python libraries like pafy and OpenCV. I created a function to filter videos and obtained the table, reducing the rows to 17,724. Next, I addressed grayscale videos. It assessed pixel variations within frames and identified grayscale videos based on color content. The resulting dataset, contained 16,263 rows, excluding grayscale frames and ensuring meaningful color information for analysis and processing.

After filtering and refining the dataset, I obtained 16,263 rows across 23 different genres. However, the dataset exhibited a skewed distribution, with some genres significantly underrepresented. To address this issue, I limited the dataset to the eight most frequently occurring genres and assigned movies to a maximum of two genres. To collect the data, I utilized the `youtube_dl` library. This library helped me extract frames from movie clips based on parameters such as the start frame, interval, and genre. To optimize efficiency, I parallelized the frame extraction process using the `multiprocessing` library, significantly reducing the extraction time. I also applied grayscale checks to avoid extracting frames from grayscale videos. The resulting dataset used binary encoding to represent genres for efficient data management and enabled multi-label classification. For the train-test split, I divided the dataset into training and testing sets, ensuring a balanced genre distribution in both subsets. I also created visualizations to analyze and compare genre distributions within different datasets. Additionally, I implemented two splitting approaches: random split and separate movies split, to evaluate the model's generalization abilities.

To integrate vectorscope analysis into the research, I've created a dedicated script that systematically processes images. This script involves resizing the images, converting them into arrays, and extracting color channel data for further analysis. These preparatory steps are crucial for generating vectorscope representations, which help in understanding color characteristics within the frames. Additionally, I've applied data augmentation techniques to enrich the dataset and enhance the model's robustness and generalization. This includes operations like rotations, flips, color adjustments, and more. Data augmentation is crucial for training machine learning models effectively, especially for tasks like genre classification. I've implemented

two data augmentation strategies: one that applies transformations uniformly to all frames and another that focuses on augmenting underrepresented genres more aggressively. These strategies aim to address imbalanced class distributions in the dataset. To facilitate data loading for training and testing, I've created a custom class that takes care of loading and preprocessing frames, including resizing and normalizing. It ensures that the data is ready to be fed into Convolutional Neural Networks (CNNs) for model training.

The initial phase of experimentation primarily revolved around selecting the most suitable dataset configuration, considering factors like augmentation. It also involved fine-tuning critical hyperparameters, particularly the learning rate and the identification of the point where overfitting becomes apparent. Furthermore, these early experiments provided insights into the computational time required for calculations. These initial trials were conducted using the AlexNet model, serving as an initial benchmark for performance evaluation. They aimed to establish preliminary results and make comparisons between the raw image dataset and the vectorscope dataset. The outcome of these initial experiments prompted the formulation of two distinct approaches for the final tests and some slight modifications and simplifications to the problem. Firstly, due to the underperformance of the model with 8 classes, I decided to merge some of the classes, resulting in having only 4 classes. I also decided to abandon the idea of multi-label classification, which led to beneficial results. Lastly, both datasets - the raw dataset and vectorscope representations dataset - went through data augmentation that focused on underrepresented classes. This decision stemmed from the performance findings and the computational efficiency of vectorscope calculations, even with augmented data.

The concluding experiments utilized the two previously mentioned datasets and four distinct models: AlexNet, VGG-16, ResNet-50, and Vision Transformer. The optimal results were achieved using ResNet-50 for raw frames, obtaining an accuracy of 60.54%, and VGG-16 for vectorscope representations, with an accuracy of 45.73%.

In the final phase, the experiments were replicated for the dataset where frames from a single clip did not appear in both the training and testing datasets. As anticipated, these yielded inferior results due to greater dissimilarity between the training and testing sets. Once more, ResNet-50 produced the highest accuracy for raw frames at 51.55%, while VGG-16 performed the best for vectorscope representations with an accuracy of 43.81%.

In order to thoroughly assess the potential for improvement, I opted to undertake a binary classification experiment focusing on the two largest classes. The outcome of these experiments was promising, with the best-performing experiment achieving an accuracy of 76.87%. This positive result indicates a notable success in the binary classification task.

In general, the overall results are suboptimal. However, a noteworthy aspect is the accuracy of experiments conducted on vectorscope representations. Not only were they nearly twice as fast, but their accuracy remained consistent across different experiment conditions—performing well on both simpler and more challenging datasets (random distribution of data versus movies separated between training and testing datasets). Another area for potential improvement is the dataset itself. The dataset used was automatically collected with basic filtering and limited to a small number of clips. Despite the current limitations in accuracy measures, the results show promise for future research, especially with the prospect of a larger and more comprehensive dataset.

Acknowledgements

I wish to convey my sincere gratitude to Professor Tania Cerquitelli and Professor Bartolomeo Vacchetti, whose inspiration served as the catalyst for my thesis. Their introduction to their research and unwavering support guided me through every step of my research journey.

This academic endeavor would have been impossible without the support of my mother, Teresa, my sister, Kasia, and the rest of my family. Their financial assistance during my studies was indispensable, and their constant presence provided not only emotional support but also invaluable advice whenever needed.

A special acknowledgment is reserved for my incredible girlfriend, Ania, who not only navigated the challenges of a long-distance relationship but also stood by me during moments when I contemplated giving up. She understood the time and dedication required for my studies, and her steadfast support has been priceless.

I express my heartfelt thanks to the friends I made during this journey - individuals I know will remain an integral part of my life forever: Ege, Amine, Enrico, Luigi, Diego, and Eren. A special mention goes to my flatmate, Gianluca, who welcomed me at the beginning of my Italian adventure, provided guidance through every settling-up challenge, and whose parents, Roberta and Maurizio, made me feel at home despite being over a thousand kilometers away from my own.

Lastly, I want to extend my gratitude to my Polish friends - Magda, Łukasz, Marta and Damian - who assisted me during my travels between Poland and Italy, always ready to lend a helping hand when needed.

I would also like to dedicate this thesis to my father, who, unfortunately, is no longer with us. He raised me to be the hardworking person I am today, and without him, this achievement would certainly not have been possible.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
Artificial Intelligence

CNN
Convolutional Neural Network

MLP
Multilayer Perceptron

csv
comma-separated values

tsv
tab-separated values

MSE
Mean Squared Error

SGD
Stochastic Gradient Descent

GPU
Graphics Processing Unit

ViT
Vision Transformer

1. Introduction

Identifying the genre of a movie from a single frame can be challenging, but not impossible. Different movie genres have distinct visual elements and stylistic techniques that can give away their identity. For example, action movies often have fast-paced scenes with intense lighting and dynamic camera angles, while horror movies may use dark and shadowy settings with eerie sound effects. Comedies often feature bright and colorful frames with exaggerated facial expressions and physical comedy. Additionally, the costumes, set design, and overall tone of the frame can also provide clues to the movie's genre. By examining these various elements, viewers can make an educated guess about the genre of the movie they are watching.

While a movie frame can provide helpful clues to identify the genre of a movie, it can also be misleading. Sometimes a single frame may not capture the essence of the movie's overall genre or storyline. For example, a horror movie may have a few scenes that take place during the day with bright lighting, which can make it appear to be a different genre, such as a drama or romance. Additionally, some movies may intentionally use a mix of genres or subvert genre expectations, making it difficult to categorize them based on a single frame. In these cases, it is necessary to watch the entire movie and take into account its overall tone and themes to accurately identify its genre.

One way to manipulate the perception of a movie's genre is through the use of color. Color is a powerful tool in filmmaking and can significantly impact how a film is perceived. Different colors can evoke various emotions and set the tone for the entire movie. For example, warm colors like red, orange, and yellow are often associated with action, adventure, and excitement. In contrast, cool colors like blue, green, and purple can create a more calming and relaxing mood and are often used in dramas or romantic movies. Horror movies often employ dark and muted colors like black, gray, and brown to create a foreboding and ominous atmosphere. Comedies frequently utilize bright and vibrant colors to establish a lighthearted and cheerful tone. Overall, color is an essential aspect of filmmaking, and filmmakers often carefully select and use colors to help convey the genre and

mood of their films.

The topic of movies and Convolutional Neural Networks might seem distant at first, but in the era of Artificial Intelligence gradually permeating various aspects of life, culture, and new industries, it's actually easy to find ways to incorporate Machine Learning into the movie industry. One of the first things that may come to mind is the implementation of recommendation systems in streaming and broadcasting services. Additionally, CNNs can be used in movie-making to identify similarities and patterns among movies belonging to specific categories. Neural networks have revolutionized the movie industry, enabling filmmakers to create incredible visual effects and improve various aspects of the filmmaking process. For example, machine learning algorithms can aid in everything from pre-production planning to post-production editing. During pre-production, neural networks can analyze scripts and generate ideas for storylines or even generate entirely new content. During filming, machine learning can assist with scene and camera setup, allowing for more precise and efficient filming. After filming, neural networks can be employed for special effects and color grading to enhance the visual aspects of the movie. Additionally, machine learning can assist in audience analysis, predicting which movies will be popular and determining the most effective marketing strategies. Overall, the utilization of neural networks in the movie industry has greatly improved the efficiency and quality of movie-making.

The goal of the project is to explore the possibility of implementing CNNs for both, classifying movie frames by genre, and utilizing vectorscopes to connect the color of a shot with a particular genre. Vectorscopes are valuable tools in video production that allow professionals to measure and analyze the color accuracy of their footage. They display chrominance information as a graph, with the horizontal axis representing the red-green color balance and the vertical axis representing the blue-yellow balance. Filmmakers can use vectorscopes to assess if the colors in their footage are properly balanced and make adjustments accordingly. Vectorscopes are particularly useful during post-production for color grading, ensuring that the final product is visually appealing and accurate. With the rise of digital video, vectorscopes have become essential for maintaining the quality of video work. Building upon this knowledge, there is a possibility of automating the coloring process of movies in post-production. The initial step involves finding the connection between raw movie frames and the genre they represent. Subsequently, the images will be processed through vectorscopes to extract color-related data, which will serve as a basis for further experiments. In the classification stage, various CNN models will be tested to determine the most suitable one.

The first step in the project is to collect the movie frame data. Free-to-use datasets typically do not provide movie frames associated with genres, but rather offer movie descriptions and genres, or short YouTube video clips. However,

YouTube provides a vast library of movie clips, trailers, and other videos that can be utilized to train machine learning models for recognizing and classifying movie frames based on their genre, color, or other visual features. With this knowledge, it is possible to retrieve movie frames using YouTube libraries in Python and create a custom dataset for the project.

After collecting and preprocessing the data, the next step is to conduct the tests. The initial tests will focus on classifying movie genres using a subset of the dataset. Utilizing the movie frames database and their corresponding descriptions, which primarily include the genres they represent, I will train the network by considering only the movie frames and the genres they represent. To create a more realistic scenario where a movie can belong to multiple genres, I will explore the topic of multi-label classification. This approach, widely used in image recognition and computer vision research, involves training a CNN to classify images into multiple categories simultaneously, rather than assigning each image to a single label. Multi-label classification using CNNs allows for a more comprehensive representation of the diverse genres that a movie can encompass.

In the initial phase of testing, a subset of the dataset will be utilized, starting with raw movie frames and later incorporating vectorscope images. The primary objective of these tests is to identify the optimal model for the given tasks, considering both accuracy measures and computation time. By experimenting with different models, architectures, and parameters, I aim to find the most effective approach for classifying movie frames based on genre using both raw frames and vectorscope representations. Once the initial testing phase is complete, the entire dataset will be employed to conduct final tests. This expanded dataset will provide a more comprehensive and representative sample, allowing for further fine-tuning of the models and parameters. To ensure an unbiased evaluation, a train-to-test split of 85% for training and 15% for testing will be employed, ensuring that the models are assessed on previously unseen data. During the testing phase, various performance metrics will be evaluated, including accuracy, precision, recall, and F1 score, to gauge the effectiveness of the models in classifying movie frames by genre. Additionally, computation time will be measured to assess the efficiency of each model in processing and classifying a large volume of movie frames.

To create a more realistic scenario, the dataset will be split in a manner that ensures frames from the same movie are not shared between the train and test datasets. This approach helps to simulate a scenario where the model is required to generalize well to unseen movies, as it cannot rely on prior knowledge of frames from the same movie during training. The testing phase will be conducted for both raw movie frames and vectorscope images using the same CNN model that was selected earlier. This allows for a direct comparison of the model's performance on both types of representations.

The experiments will be conducted using a variety of approaches, including the utilization of pretrained models such as AlexNet, as well as the exploration of custom architectures. By employing pretrained models like AlexNet, which have been trained on large-scale datasets such as ImageNet, we can leverage their learned features and transfer their knowledge to our movie frame classification task.

2. Related Work

2.1 Movie Genre Classification

The usage of Convolutional Neural Networks (CNNs) has been studied for various applications, including the classification of movies based on different features such as genre, sentiment, and content.

In 2002, Zeeshan Rasheed and Mubarak Shah [1] undertook research on genre classification using movie previews. Their objective was to analyze audio-visual cues within the previews, such as camera movements, sound effects, and lighting, which contribute to the creation of mood and atmosphere. The authors' methodology involved several steps. Firstly, they determined shot length by detecting changes in shots and extracting a key frame positioned at the midpoint of each shot. They then examined visual disturbances within the scenes by analyzing scene motion and assigning gray levels to moving pixels. This allowed them to differentiate between static and moving scenes. The visual disturbance was measured as the ratio of moving pixels to the total number of pixels in a slice, with action movies typically exhibiting more local motion. Furthermore, the researchers focused on non-action movies and analyzed them based on lighting characteristics, classifying them into one of three genres: comedy, horror, or drama (other). As for action movies, their analysis concentrated on identifying audio peaks that indicated the presence of fire or explosions within a shot.

In 2016, Gabriel S. Simões et al. [2] proposed using CNNs for pattern recognition and classification of movie genres. This approach aimed to leverage the fact that videos are composed of images and focused on movie trailers, which are short clips that showcase the highlights of the movie and are often used to promote it. The researchers considered movie trailers simply as a sequence of movie frames and used CNNs to analyze these frames and classify the movie genre based on the visual features present in the frames. To achieve this, the authors created a new classification method called CNN-MoTion (Convolutional Neural Networks for Movie Trailer Classification). They used a large dataset of movie trailers and preprocessed them to extract the frames and convert them into images.

They then trained a CNN model on these images to learn the visual features that are indicative of different movie genres. The CNN-MoTion model was able to achieve an accuracy of 65.31% in classifying the movie genre, which is a significant improvement over the previous state-of-the-art methods.

Movie trailers and posters have been explored extensively for predicting the genre and other attributes of movies. While movie trailers have been a popular choice due to their ability to provide a glimpse of the movie's content, movie posters have also been used to predict the genre with a high degree of accuracy. Even in the midst of visual complexity and a multitude of details, a movie poster can swiftly convey the genre (such as drama, comedy, horror, etc.) to viewers.

One of the first ones to utilize movie posters in genre prediction were Marina Ivasic-Kos et al. [3]. They used 1500 movie posters evenly balanced across 6 different genres. They considered multi-label classification task, that was transformed to single-label task, using distance ranking, Naïve Bayes and RAKEL. The authors decided to focus on low-level features of movie posters such as dominant color of the whole poster, texture or color histogram, similarly to my approach. The highest accuracy they reached was 67% for at least one of two correctly detected labels but only 14% for two out of two correctly detected labels.

In the past also, Nirman Dave [4] used a dataset of 40,000 movie posters across 28 different genres to predict the genre using machine learning techniques. Dave's approach involved using a simple 7-layer neural network similar to the VGG model for multi-label classification. The neural network was trained on the dataset of movie posters, and the learned features were used to predict the genre of unseen posters. Dave's approach achieved an accuracy of 50.50%.

In a similar vein to Nirman Dave's approach, Gabriel Barney and Kris Kaya [5] also conducted research in the field of genre classification. However, they utilized different architectures, such as ResNet34 or custom architectures, depending on the specific experiment. Their work made use of the MovieLens Dataset from Kaggle [6], which encompasses entries for 45,466 movies spanning across 21 genres. After filtering, they extracted a dataset comprising approximately 35,000 movie posters. Similar to the methodology adopted in my research, the authors standardized each image in their dataset to a resolution of 224x224 pixels. However, their results demonstrated slightly lower accuracy measures compared to the findings of Nirman Dave. The "all match" accuracy ranged between 6% and 13%, indicating that the models exhibited limited success in correctly predicting all genres associated with a given movie poster. On the other hand, the "at least one match" accuracy ranged between 19% and 45%, suggesting that the models were more consistent in identifying at least one correct genre for a movie poster.

In line with the research conducted by Marina Ivasic-Kos et al. [3], Nayeem Hossain et al. [7] also delved into the realm of multi-label classification for movie

genres. They employed distance ranking, Naive Bayes, and RAKEL algorithms as part of their approach. However, in contrast to previous studies, they introduced a comprehensive ensemble of six distinct deep learning models, including Lenet, Alexnet, VGG-16, VGG-19, Resnet-50, and a custom model. These models were utilized to extract relevant features from the movie poster images and subsequently classify the genres. Remarkably, the custom model proposed by the authors achieved an impressive accuracy rate of 91.15%. This result represents a significant improvement compared to the aforementioned research. The utilization of deep learning models and the creation of a customized architecture enabled Hossain et al. to enhance the accuracy and effectiveness of movie genre classification.

In 2018, Quan Hoang [8] adopted a distinct methodology by employing Machine Learning techniques to predict movie titles based on plot summaries. They utilized various approaches, including Naive Bayes, Word2Vec+XGBoost, and Recurrent Neural Networks, for text classification. To address the multi-label problem inherent in genre tagging, they employed methods such as K-binary transformation, rank method, and probabilistic classification with a learned probability threshold. For their experiments, the authors utilized a dataset comprising 250,000 movies along with their corresponding plot summaries. By leveraging their chosen Machine Learning techniques, they achieved a hit rate of 80.5% across 10 genres in their genre tagging task.

2.2 Deep Learning-Based Classification of Movie Shots

The goal of my research focuses on movie frames alone without considering movie posters. Similar approach was taken by Bartolomeo Vacchetti, Tania Cerquitelli, and Riccardo Antonio [9] who explored the application of deep learning techniques in classifying movie shots into four categories: full figure, half figure, half torso, and close-up. The aim of their research was to assist professionals in creative fields who frequently encounter unorganized data. To tackle this task, they utilized the VGG-16 architecture, which had been pretrained on the ImageNet dataset [10]. To further optimize the model's performance, they employed two datasets: one comprising RGB images and another containing monochrome images. With a dataset of 3000 images, the authors achieved an average accuracy of 81.30% using the VGG-16 architecture. Their research demonstrated the effectiveness of deep learning in accurately classifying movie shots according to specific visual attributes.

Bartolomeo Vacchetti and Tania Cerquitelli expanded their research in the field of film shot classification by considering a broader range of shot categories. In their study [11], they examined eight distinct classes of film shots: long shot, medium shot, full figure, American shot, half figure, half torso, close-up, and

extreme close-up. To accomplish this, they employed the VGG-16 architecture, but with a novel approach involving three separate VGG-16 models. These models were combined using the stacking learning technique to enhance performance. Each of the three VGG-16 models was trained on a different version of the dataset. The first model utilized the original dataset, while the second model used a dataset with the most relevant patterns highlighted through hypercolumns extraction. The third model leveraged a dataset that had undergone semantic segmentation. The predictions from the three models were then combined and used as input for a MLP model. Unlike their previous study, the authors worked with a significantly larger dataset consisting of 10,545 images across the eight aforementioned shot classes. They achieved impressive results, with a training accuracy of 95.66% and a validation accuracy of 77.02% for the final classifier. The individual VGG-16 classifiers exhibited training accuracy values around 95% and testing accuracy values around 70%.

The research conducted by the same group of researchers delved deeper into the intersection of machine learning and the movie industry [12]. They introduced Movie Lens, a data-driven approach aimed at discovering and characterizing editing patterns through the analysis of short movie sequences. The methodology employed in this research involved leveraging the Levenshtein distance, the K-Means algorithm, and a Multilayer Perceptron (MLP). To gather the necessary data, the authors utilized the Cinescale dataset, which provided labels for each movie, and extracted frames from the available movies. They categorized the frames into eight different types of shots: extreme long shot, long shot, medium long shot, medium shot, medium close-up, close-up, extreme close-up, and foreground shot. To further increase the number of classes, they applied K-Means clustering and calculated the Levenshtein distance. This resulted in a total of 4, 8, 16, or 32 classes. The accuracy of the classification models was evaluated based on the number of classes utilized. The results showed accuracies of 93%, 88%, 81%, and 77% for 4, 8, 16, or 32 classes, respectively. These findings highlight the effectiveness of the approach in accurately identifying and characterizing editing patterns within movie sequences.

In line with previous studies, my research aims to establish a connection between genre identification and movie frames, rather than relying solely on movie posters. Furthermore, I plan to investigate the potential benefits of incorporating vectorscope images into the existing models to enhance their performance and potentially improve accuracy. By exploring these avenues, I aim to contribute to the ongoing advancements in genre classification techniques and provide insights into the effectiveness of utilizing movie frames and their vectorscope representations for this purpose.

3. Methodology

3.1 Dataset

For the research I'm conducting, my first task was to find a suitable dataset. This dataset needed to include movie frames with assigned labels indicating their genres. Although there are several online services like SHOTDECK [13], Flim [14], and SHOT.CAFE [15] that provide data collected in this manner, I couldn't utilize them due to copyright restrictions.

Therefore, I had to devise a method to gather the data and construct the dataset myself. Fortunately, I stumbled upon the CondensedMovies repository [16], which proved to be valuable. This dataset encompasses over 3,000 movies, featuring more than 30,000 professionally captioned clips, totaling over 1,000 hours of video. Additionally, it includes over 400,000 facetracks and precomputed features from 6 different modalities. The dataset also provides links to movie clips from the MOVIECLIPS YouTube channel [17] and a brief description for each clip. Since my research solely required movie frames and their corresponding genres, the most relevant files within the dataset were `clips.csv` [Table 3.1], that contained 34,185 rows of data, and `movie_info.csv` [Table 3.2], that contained 2,894 rows of data.

videoid	...	title	clip_name	imdbid
Sv-BxH3SVS8	...	Les Misérables	One Day More Scene	tt1707386
g8pt9OoaPlY	...	Shazam!	Dr. Sivana Attacks Scene	tt0448115
4-BWFsE_TQE	...	Good Boys	Frat House Fight Scene	tt7343762
iPcAns5pKVw	...	mother!	The Agony of Birth Scene	tt5109784
YHvTfLaOREg	...	Charlotte's Web	I Can Talk! Scene	tt0070016

Table 3.1: `clips.csv` with five exemplary rows

imdbid	genre	country
tt0020629	"['Drama', 'War']"	['USA']
tt0068767	"['Action', 'Drama', 'Romance']"	['Hong Kong']
tt0021814	"['Fantasy', 'Horror']"	['USA']
tt0021884	"['Drama', 'Horror', 'Sci-Fi']"	['USA']
tt0068909	"['Drama', 'Fantasy', 'Musical']"	"['Italy', 'USA']"

Table 3.2: `movie_info.csv` with five exemplary rows

The original tables I worked with were presented above [Table 3.1, Table 3.2]. In the `clips.csv` table, along with the `videoid`, `title`, `clip_name`, and `imdbid` columns, there were additional columns like `year`, `clip_idx`, `clip_tot`, and `upload_year` that were not relevant to my research. The `imdbid` column acted as a foreign key, while the `videoid` column served as the primary key and contained an extended YouTube link directing to the corresponding movie clip. The second table, `movie_info.csv`, contained columns for `imdbid`, `genre`, and `country`. The `imdbid` column acted as the primary key, and it was used later to perform a join operation with the `clips.csv` table. This join operation resulted in the creation of the `movieclips_all.tsv` table [Table 3.3]. Moving forward, I decided to work with `.tsv` files as they were easier to read and manipulate. All the table operations were performed using Python's `pandas` library, utilizing dataframes.

imdbid	...	videoid	genre
tt0448115	...	Ok63vpXNhNc	['Action', 'Adventure', 'Comedy']
tt0069995	...	r0iSxOsPGl8	['Drama', 'Horror', 'Thriller']
tt0073440	...	dbX-ekoWGWE	['Comedy', 'Drama', 'Music']
tt1386932	...	9aR0JkmZLk0	['Action', 'Biography', 'Drama']
tt0332375	...	VlMy5-BAjzo	['Comedy', 'Drama']

Table 3.3: `movieclips_all.tsv` with five exemplary rows

The table displayed above [Table 3.3] shows the outcome of the join operation between the two previously mentioned tables. It comprises 27,077 rows. The original table includes six columns: `imdbid`, `title`, `clip_name`, `year`, `videoid`, and `genre`. However, only the `videoid` and `genre` columns are relevant for further analysis and developments.

In the next step of filtering the dataset, I applied various techniques using the `pafy` and `OpenCV` libraries. I created a function called `video_available(link)`

[Listing A.1] that took a YouTube video link from the table as input and returned either True or False, indicating if the video was available or not. Based on this output, the corresponding table row was either discarded or appended to a new dataframe. This process resulted in a new table called `movieclips_available_only.tsv`, which had the same structure as `movieclips_all.tsv` 3.3], but with a reduced number of rows, specifically 17,724 rows.

Subsequently, I encountered the need to eliminate links to YouTube videos that were in grayscale, as they proved to be irrelevant for the vectorscope analysis. To accomplish this, I implemented the `color_check(link)` [Listing A.2] function, which examined the pixel variations within a previously extracted frame. The function calculated the sum of these variations, determined the ratio between the differences and the image size, and compared it to a predefined threshold. Based on this comparison, the videos were either discarded or retained.

For the task, I utilized the `cv2.split(frame)` function from the `OpenCV` library. This function is specifically designed to split a multi-channel image, such as a frame from a video, into its individual color channels. By applying this function, the input frame is transformed into a list of separate image channels, with each channel representing a specific color. When working with RGB images, the `cv2.split()` function returns a list consisting of three channels: one for the red channel, one for the green channel, and one for the blue channel. These channels are represented as matrices, where each matrix element corresponds to the intensity of the respective color at a particular pixel. To determine whether the frame represents a grayscale movie, I utilized the `NumPy` library. Specifically, I employed the `np.count_nonzero(abs(r-g))` function, where `r` and `g` are the matrices representing the separate red and green color channels, respectively. By calculating the absolute difference between the two color channels and counting the number of non-zero elements, we obtain an indication of the amount of color variation in the frame. The next step involves summing up the outputs of `np.count_nonzero()` for each color channel and dividing the result by the total size of the frame. This calculation yields a ratio that represents the proportion of non-zero color differences in the frame relative to its overall size. Finally, if the obtained ratio is lower than 0.005, the frame is discarded as it likely represents a grayscale movie. This threshold value serves as a criterion to distinguish between predominantly grayscale frames and frames with significant color variation. By applying this process, we can effectively identify and exclude grayscale frames from further analysis or processing, focusing only on frames that contain meaningful color information.

Consequently, I obtained the final dataset called `movieclips_final.tsv`, which contained 16,263 rows after this filtering process.

3.1.1 Reducing the Dataset

After filtering the dataset, I obtained a substantial amount of data comprising 16,263 rows across a diverse range of 23 different genres. However, I encountered a challenge as some genres were significantly underrepresented, with only 6 links associated with the least represented genres, while the most represented genre had a staggering 6,945 links. This skewed distribution posed difficulties for conducting accurate experiments, as certain genres lacked sufficient representation. To address this issue and ensure a balanced distribution, I made the decision to limit the number of genres to the eight most frequently occurring ones. This approach would allow for a more equitable distribution of representatives across genres, facilitating more reliable and comprehensive experiments. Additionally, in order to streamline the multi-label classification process, I opted to assign a movie to a maximum of two genres. The final selection of eight genres, along with their respective representation in the dataset, are as follows:

- Comedy: 7512,
- Drama: 6945,
- Action: 4340,
- Adventure: 3305,
- Romance: 3083,
- Crime: 2976,
- Thriller: 2214,
- Horror: 1718.

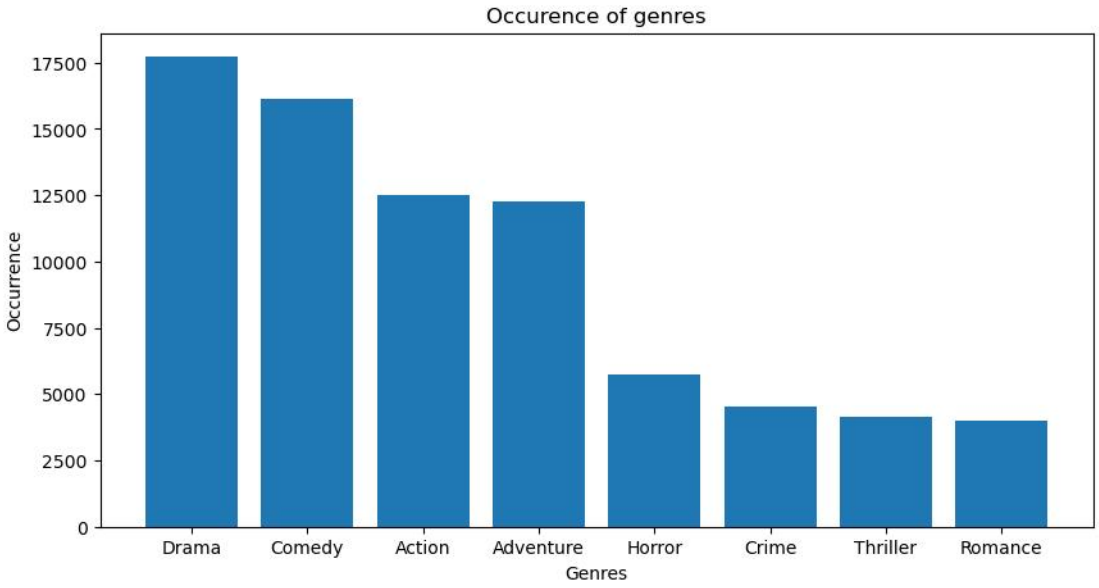

Figure 3.1: Number of occurrences of each genre

3.1.2 Collecting the Data

Once I had finalized the dataset, consisting of 16,106 rows with genres and corresponding links to movie clips, I proceeded with the data collection process. To extract the frames from the movie clips, I utilized the `youtube_dl` Python library, leveraging the functionality provided by the `extract_frame(link, genre, start_frame, interval, folder_path)` function [Listing A.3]. This function played a crucial role in the data collection pipeline, as it required several input parameters. Firstly, the function took the `link` to the movie clip as an argument, which served as the source for frame extraction. Additionally, the `genre` associated with the movie was also provided as input, allowing for proper categorization and organization of the extracted frames. Moreover, the function required the specification of the `start_frame` and `interval` parameters, both measured in seconds. The `start_frame` parameter was particularly useful, as it enabled the exclusion of initial frames that typically consisted of advertisements or opening credits. By specifying a suitable value for `start_frame`, we ensured that the extraction process commenced at an appropriate point within the video. The `interval` parameter played a significant role in determining the frequency at which frames were extracted. It indicated the time duration, in seconds, between each extracted frame. By adjusting this parameter, we could control the granularity of the extracted frames, striking a balance between capturing sufficient information and avoiding redundancy. To prevent the extraction of end credits, which often

appear towards the conclusion of a video, I chose to extract frames up until approximately three-fourths of the video's duration. This approach ensured that the last extracted frame would capture a representative moment from the movie, while excluding the end credits. Lastly, the function required the specification of the `folder_path`, which determined the destination folder where the extracted frames were saved. This parameter facilitated the organization and storage of the collected data, ensuring easy access and retrieval during subsequent stages of the project. By employing the `extract_frame()` function in conjunction with the `youtube_dl` library and appropriately configuring its parameters, I was able to efficiently collect and extract frames from the movie clips, contributing to the development of a robust and comprehensive dataset.

Upon executing the frame extraction process, I soon became aware of the considerable computational time required to complete the task. Realizing the need for optimization, I sought to implement the powerful `Pool()` function from the `multiprocessing` Python library. This approach proved instrumental in improving the efficiency of the solution by leveraging all available CPU cores. By utilizing the `Pool()` function, I was able to parallelize the frame extraction code, distributing the workload across multiple cores simultaneously. This enabled me to collect the entire dataset in a significantly reduced timeframe, achieving speeds up to ten times faster compared to the initial implementation. To further enhance the extraction process, I made adjustments to the parameters of the `extract_frame()` function. I set the `start_frame` parameter to 2 seconds, ensuring that the extraction process commenced slightly beyond the initial moments of the video. Additionally, I set the `interval` parameter to 10 seconds, allowing for a more spaced-out extraction of frames. This adjustment struck a balance between capturing an adequate number of frames and avoiding excessive redundancy, resulting in a more representative sample of frames from each video. Before extracting frames from a particular video link, I implemented a check to ensure its availability. This preliminary verification helped to avoid extracting frames from broken or inaccessible links, ensuring the integrity and reliability of the collected data. Furthermore, I performed a grayscale check on the video to determine if it was already in grayscale format. After completing the frame extraction process using the optimized approach, I successfully obtained a total of 45,845 frames from the dataset. This achievement was made possible through the implementation of multiprocessing techniques, parameter adjustments, and thorough verification checks. These optimizations collectively contributed to a significant improvement in the efficiency and effectiveness of the frame extraction process. The resulting dataset of extracted frames now provides a comprehensive foundation for conducting further analysis and experimentation.

The extracted data was saved in a structured format within the `data.tsv` file [Table 3.4]. To ensure efficient data management and facilitate future analysis,

I employed a specific formatting strategy. Instead of storing the genre information as a string, I opted for a more streamlined approach by mapping each genre to a binary value. In this format, I designated the `path` to the file as the primary key, ensuring a unique identifier for each frame. This key served as a reference point for easy retrieval and linking of relevant information during subsequent data processing tasks. For each frame, the genre information was represented using a binary encoding system. If a frame belonged to a particular genre, it was assigned a binary value of 1. Conversely, if a frame did not fall within that genre, it was assigned a binary value of 0. This mapping approach allowed for efficient and compact representation of the genre information, reducing storage requirements and facilitating streamlined data manipulation. Additionally allowed to implement the multi-label classification. The process was executed in a loop that ran through folders and saved the data inside a dataframe, which was later saved as a `data.tsv` file [Listing A.4].

path	genre	Action	Adventure	...	Thriller
frames/ Action/ 06Its9LhIHQ/ 1196.jpg	'Action'	1	0	...	0
.../1196.jpg	['Action', 'Adventure']	1	1	...	0
.../1196.jpg	['Thriller']	0	0	...	1
.../46.jpg	['Action', 'Thriller']	1	0	...	1
.../736.jpg	['Action', 'Thriller']	1	0	...	1

Table 3.4: `data.tsv` with five exemplary rows

In addition to the genres listed in the provided table [Table 3.4], the absent genres include Comedy, Crime, Drama, Horror, and Romance.

3.1.3 Train Test Split

To effectively train Convolutional Neural Networks (CNNs), it is crucial to provide separate datasets for both training and testing (validation) purposes. The objective is to feed the network with the training set during a series of epochs to optimize its performance, and subsequently evaluate the network's ability to accurately classify and predict labels by running the test set through the trained network. To ensure the fairness and reliability of the evaluation process, it is essential that both the training and test sets exhibit a similar distribution of data.

This means that the genres represented within each set should be representative of the overall dataset. To achieve this, I made the decision to split the full dataset into distinct subsets, allocating approximately 85% of the data for training purposes and reserving the remaining 15% for testing. This partitioning strategy ensures that the CNNs receive an ample amount of data for training while also enabling a comprehensive evaluation of their performance on unseen samples during the testing phase. By maintaining a balanced distribution of genres within both the training and test sets, we can effectively assess the network's ability to generalize and make accurate predictions on new, unseen data. This partitioning approach provides a robust foundation for training, evaluating, and fine-tuning the CNN models, ultimately enhancing their overall performance and predictive capabilities.

The distribution of data is a crucial aspect to consider when analyzing datasets. In Figure 3.2, we can observe the representation of data and gain valuable insights. To ensure accurate comparisons, the values have been normalized [Listing A.6], allowing us to focus on the ratios of distributions among different datasets. While visually examining the figure, it may appear that there are slight variations between the distributions. However, it is important to note that the primary focus lies in the order of class sizes in the datasets rather than the minor differences in their shapes. This means that regardless of the dataset size, the relative order of magnitude is preserved.

To facilitate the proper analysis and visualization of the data, I developed a custom function called `get_genre_distribution(df)` [Listing A.5]. This function was designed to streamline the process by taking a dataframe as input, generating a figure displaying the distribution of genres, and returning a dictionary with the assigned genres and their corresponding values. To begin, within the function, I implemented a series of iterative steps to accurately extract the genre information from the dataframe. Initially, I iterated over the dataframe to create a list of unique genre combinations. This step ensured that each genre combination was accounted for and provided a comprehensive overview of the different genres present in the dataset. Subsequently, I performed another iteration over the list of genre combinations. This time, the objective was to isolate individual genres from the combinations and create a new list containing only single genres. This separation allowed for a more granular analysis and a better understanding of the distribution of each genre in the dataset. Finally, leveraging the obtained list of single genres, I constructed a dictionary where each genre was assigned a value based on its occurrence in the dataset. This dictionary, serving as the output of the function, provided a comprehensive overview of the genre distribution within the dataset, facilitating further analysis and interpretation.

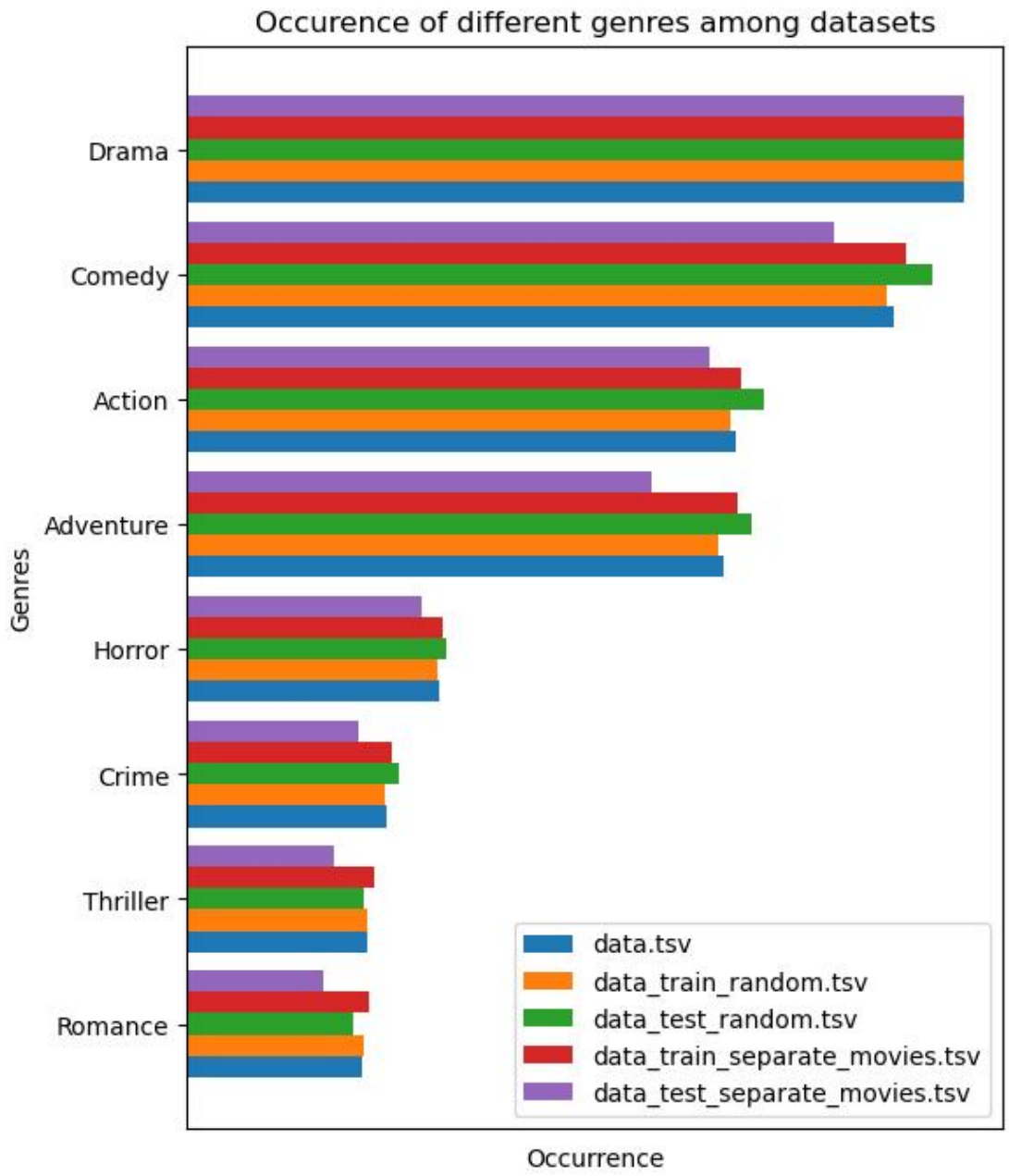

Figure 3.2: Distribution of data among test and training datasets

Figure 3.1 displays the distribution of genres for 5 different `.tsv` files. The file containing all the data, `data.tsv`, was divided into training and testing sets using two different approaches.

In the first approach, the data was split randomly, and the resulting frames were assigned to two separate files: `data_train_random.tsv` and `data_test_random.tsv` [Listing A.7]. To achieve this, the code utilizes the `train_test_split()` function from the `sklearn` Python library. This function takes a dataframe and the desired split ratio as inputs and returns two subsets. In this case, I opted to allocate 85% of the dataset for training and 15% for testing purposes.

To further explore the data, I employed a second approach for dividing the dataset, resulting in the creation of two distinct files: `data_train_separate_movies.tsv` and `data_test_separate_movies.tsv` [Listing A.8]. Unlike the random split approach, this method ensures that frames from the same movie are never present in both the training and testing sets. This separation mimics a more realistic scenario, where the model is trained on one set of movies and tested on another set that it has never encountered during training. By implementing this approach, we can evaluate the model's ability to generalize and make accurate predictions on unseen movies, providing valuable insights into its performance in real-world scenarios.

3.2 Data Transformations

3.2.1 Vectorscope

My research primarily aims to establish a correlation between the color palettes employed in movie frames and the corresponding cinematic genres they represent. To accomplish this objective, I am employing vectorscopes as the instrumental tools for analysis.

Vectorscopes are specialized tools used in visual analysis, particularly in video and film production. They help to graphically display and interpret the distribution of colors within an image or video frame. Vectorscopes provide insights into how different colors are being used, allowing for the assessment of color balance, saturation, and other chromatic characteristics. In essence, vectorscopes help professionals in the visual arts industry to better understand and manipulate color elements in their work.

To integrate the vectorscope tool into my research methodology, I developed a dedicated script titled `vectorscope.py`, as detailed in Listing A.9. This script plays a crucial role in the process of applying vectorscope analysis. The code, presented within Listing A.9, follows a systematic sequence of operations.

The initial steps encompass the opening of the target image, followed by a resizing operation that ensures compatibility with subsequent computations. The resized image is then translated into a `NumPy` array, thereby facilitating efficient

data manipulation and analysis. Within this array, pertinent data pertaining to individual color channels is meticulously extracted and isolated into distinct variables. This preparatory phase is an essential prerequisite for subsequent transformations that ultimately culminate in the creation of the final vectorscope representation, as illustrated in Figure 3.4.

Figure 3.3: Example of the raw image

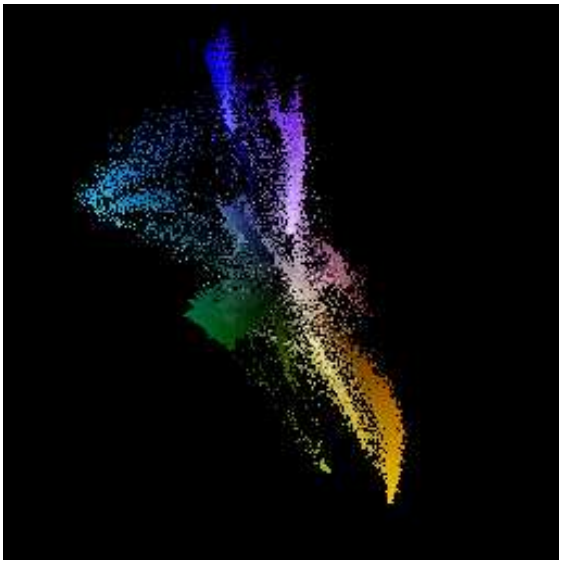

Figure 3.4: Example of the vectorscope image

3.2.2 Data Augmentation

To broaden the dataset synthetically, I opted to implement data augmentation techniques strategically. Data augmentation, widely used in machine learning and computer vision, involves purposefully applying various modifications to existing data to achieve two goals: enhancing dataset diversity by creating plausible variations of original data points and reinforcing models' robustness and generalization.

By seamlessly integrating data augmentation into the training process, I enriched the dataset's content, potentially boosting learning algorithm performance. This involves operations like geometric transformations (rotations, translations, flips), color adjustments, noise addition, and occlusion simulations. These transformations introduce new instances that retain core features while incorporating realistic deviations.

This expansion curbs overfitting risks, enabling the model to grasp a wider array of patterns. Models trained on this augmented dataset are expected to excel in accuracy and adaptability to real-world data. This deliberate augmentation effort aligns with the goal of crafting a more effective machine learning framework.

The effectiveness of these augmentation techniques will be rigorously tested in later project stages. Through thorough evaluation and validation of the augmented dataset, the impact on model performance will be accurately gauged. This entails benchmark tests and comparisons to understand how augmentation enhances the model's adaptability to real scenarios. This iterative process aims to validate the hypothesis that data augmentation significantly improves the model's learning capacity and prediction accuracy.

The set of transformations I chose to implement uses several operations, such as random cropping, horizontal and vertical flipping, color manipulation, and blurring. Each operation plays a crucial role in shaping a comprehensive augmentation strategy aimed at diversifying the dataset. The parameters governing the degree of these transformations were intentionally selected through a randomized process, as shown in Listing A.10. These transformative procedures were executed within the `generate_modified_image(image_path)` function, as outlined in Listing A.11.

In the subsequent workflow, images were loaded within a loop and subsequently subjected to the transformation routines. While the original images were preserved in their unadulterated states, a new images were introduced by incorporating the outcomes of these transformations. This augmentation approach was strategically employed on 70% of the image collection, chosen at random.

The results of the earlier mentioned transformations were shown in Figures 3.5, 3.6, 3.7 and 3.8.

Figure 3.5: Raw image before the transformation

Figure 3.6: Raw image after the transformation

Figure 3.7: Raw image before the transformation

Figure 3.8: Raw image after the transformation

The transformations led to the acquisition of a training set comprising 66,205 images, as opposed to the original 38,968 samples prior to augmentation. Simultaneously, the initial test set of 6,877 images remained unaltered. The distribution of genres within the augmented dataset is visually represented in Figure 3.9, mirroring the patterns observed in the original distribution, as depicted in Figure 3.1.

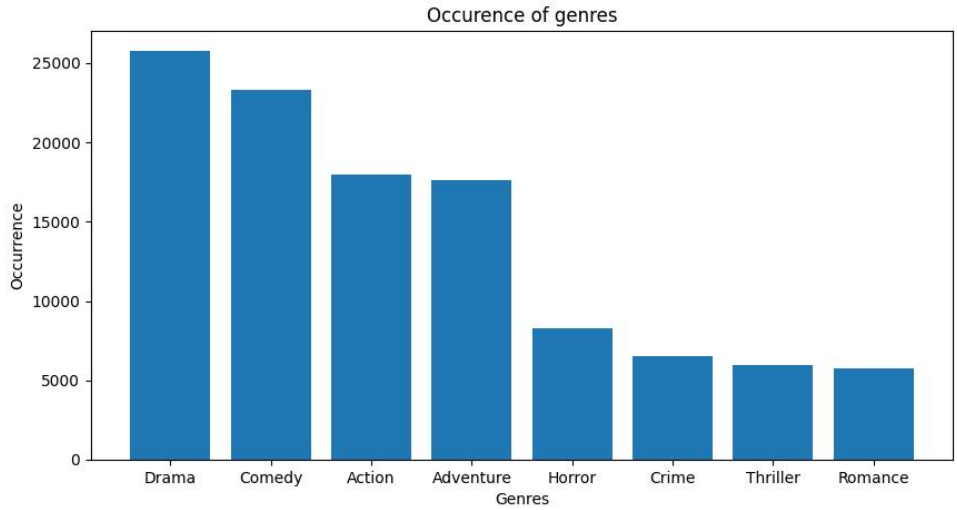

Figure 3.9: Number of occurrences of each genre in augmented dataset

The issue that augmentation failed to address is the imbalanced distribution of data. This discrepancy is evident in Figures 3.1 and 3.9, where genres such as horror, crime, and thriller are significantly underrepresented. To mitigate this, I developed an alternative approach to data augmentation. This involved a strategic focus on augmenting the number of samples within the earlier mentioned genres. Although the image transformations remained unchanged, I modified the execution of the loop. In the previous version, the script randomly transformed 70% of the data. However, in this iteration, I took a different route. I initially extracted the combination of genres attributed to each movie. If the genres included horror, crime, thriller, or romance, there was a 90% likelihood of the frame undergoing transformation. For the remaining samples, the likelihood of transformation was reduced to 10%.

As a result of this adaptation, the issue of imbalanced class distribution was somewhat alleviated, as depicted in Figure 3.10. However, it's important to note that this approach was executed in the context of a multilabel classification problem. The resulting training dataset comprised 54,123 samples, reflecting the

outcomes of this adjustment.

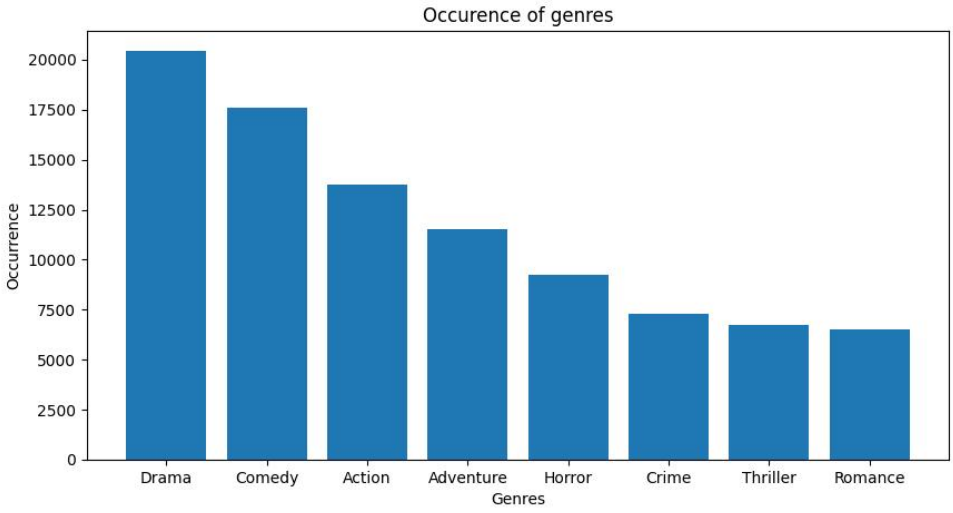

Figure 3.10: Number of occurrences of each genre in augmented dataset with focus on rare genres

The same procedure was applied to a dataset in which frames from the same movie were not shared between the training and testing datasets. This resulted in the acquisition of 66,417 frames [Figure 3.11] when applying uniform transformations to all frames, and 54,179 frames [Figure 3.12] when specifically targeting underrepresented genres.

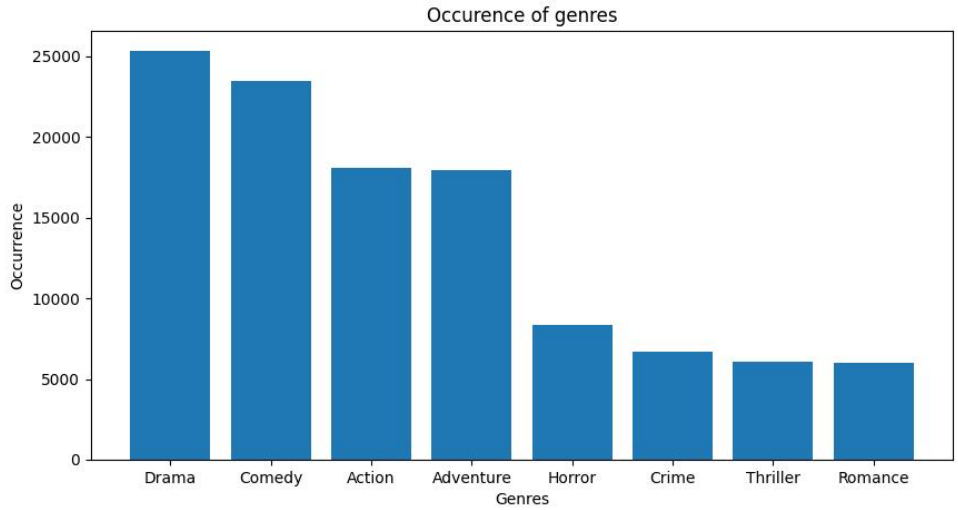

Figure 3.11: Number of occurrences of each genre in augmented dataset for frames separated among datasets

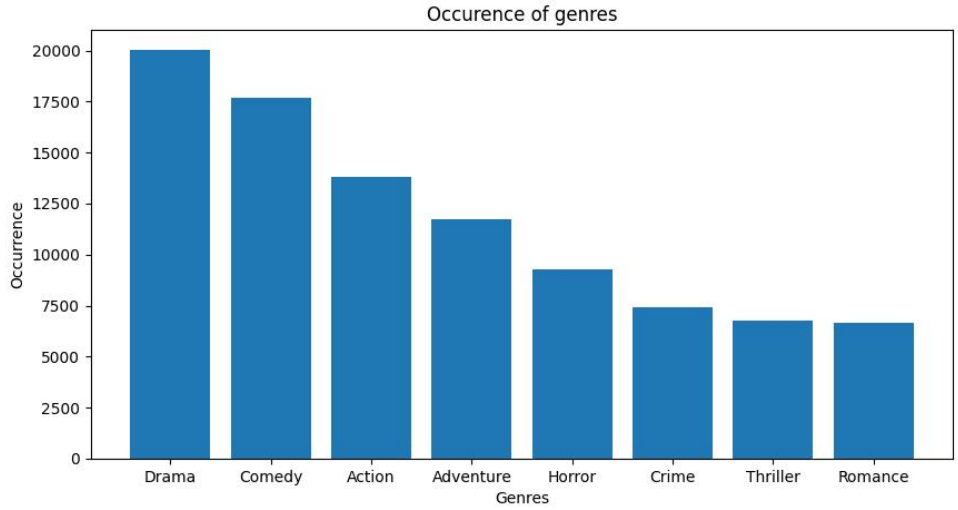

Figure 3.12: Number of occurrences of each genre in augmented dataset for frames separated among datasets with focus on rare genres

In the context of the vectorscope analysis, I contemplated two distinct approaches:

1. Utilizing both the original and transformed raw images and subjecting them

to vectorscope analysis.

2. Directly applying various transformations to the vectorscope images themselves.

After careful consideration, I opted for the first approach. This decision was influenced by the recognition that the vectorscope extracts very specific information from images. Implementing transformations directly on the vectorscope images could potentially introduce alterations that might negatively impact classification results. For instance, common transformations like image cropping could significantly disrupt the vectorscope's distribution, while altering colors might manipulate the key parameter derived from the vectorscope analysis. Ultimately, the choice to analyze the original and transformed raw images using the vectorscope was deemed more conducive to preserving the inherent insights generated by this analytical tool, thus ensuring a more stable basis for subsequent classification processes. The differences between ram images and vectorscope images are shown on Figures 3.13, 3.14, 3.15 and 3.16.

Figure 3.13: Raw image before the transformation

Figure 3.14: Vectorscope of the raw image

Figure 3.15: Raw image after the transformation

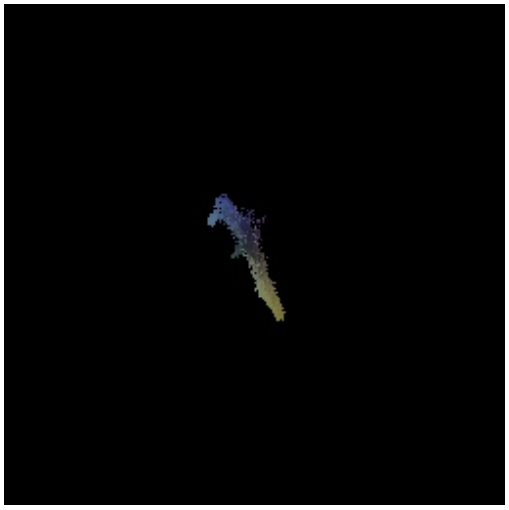

Figure 3.16: Vectorscope of the image after the transformation

3.2.3 Loading the Data

To load the dataset inside of the CNN I needed to create a custom `MovieFrameDataset(Dataset)` [Listing A.12] class that inherited from the `Dataset` class from Python `torch` library.

The `Dataset` class is a fundamental component of the `torch.utils.data` library in Python's `PyTorch` framework. It provides an interface to load and process data for machine learning tasks, such as training and evaluation of deep learning

models. The `Dataset` class serves as an abstraction for a collection of data samples and their corresponding labels (if applicable). It is designed to be flexible and can handle various types of data, including images, text, audio, and more. By encapsulating the data in a `Dataset` object, it becomes easier to manipulate and transform the data in a consistent manner. To use the `Dataset` class, you typically create a custom subclass that inherits from it and implement two key methods:

- `__len__()`: this method returns the total number of samples in the dataset,
- `__getitem__(index)`: this method allows you to retrieve a specific sample and its associated label (if available) given its index. It should return the sample and label in a format that can be processed by your model.

By implementing these methods, I could access individual samples from the dataset using indexing, iterate over the dataset, and leverage `PyTorch` data loading utilities. Once I have created my `Dataset` object, I could use it in conjunction with other components of the `torch.utils.data` library, such as `DataLoader`, to efficiently load and preprocess the data in batches, shuffle the data, and parallelize data loading across multiple threads.

My custom `MovieFrameDataset(Dataset)` subclass takes three inputs: `root_dir`, `csv`, and `transform`. The `root_dir` parameter represents the path to the parent folder where the frames are stored. It serves as the base directory for accessing the frames. The `csv` parameter contains the exact location of each frame within the dataset, along with the corresponding genres assigned to each frame. This information is essential for accessing and associating the frames with their respective genres during the dataset loading process. Lastly, the `transform` parameter contains the transformation applied to the input image [Figure 3.17] to ensure consistency with the CNN. In the provided code, the frames are resized to a resolution of 224x224 pixels and centered. Additionally, the images are normalized to facilitate faster calculations, as shown in Figure 3.18. It's important to note that this transformation is only applied to the raw images on both training and testing set, as the vectorscope images do not require any transformation. By incorporating these functionalities within the `MovieFrameDataset(Dataset)` subclass, the dataset can be effectively loaded, accessed, and preprocessed for training or evaluation purposes.

Figure 3.17: Example of a raw frame

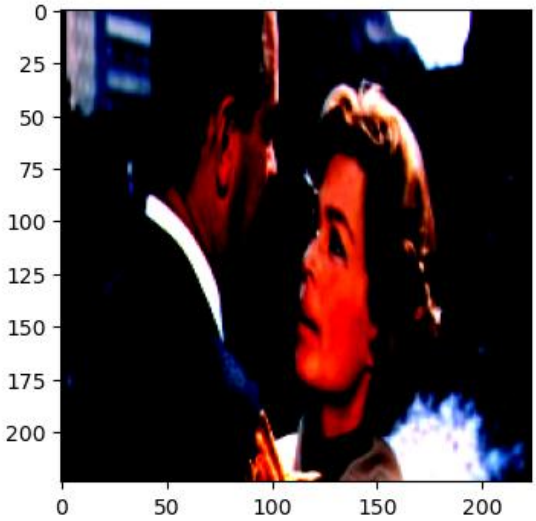

Figure 3.18: Example of a transformed frame

Following these stages, the `train_set` and `test_set` operate as inputs for the `DataLoader` class within the `torch.utils.data` library. The parameters employed in the conducted experiments are supplied to the program via the command line, utilizing the `parse_args` library as demonstrated in Listing A.14. The images are loaded in batches and are either transformed or preserved as vectorscope representations, as depicted in Figures 3.19 and 3.20.

Figure 3.19: Example of loaded raw data

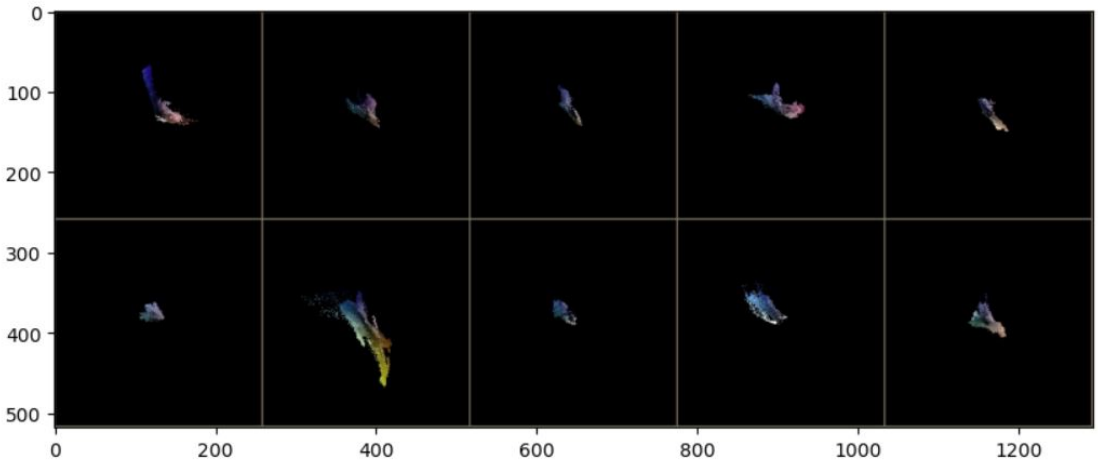

Figure 3.20: Example of loaded vectorscope representations

4. Experiments

In terms of the experimental process, the main focus was to evaluate how a single model performs with different versions of the dataset, as well as to assess various models in different scenarios. Initially, the goal was to check the model's performance using the original dataset and to see if adding augmented data brings any improvements. Additionally, comparisons were made between raw images and vectorscope images, considering both overall performance and calculation time. These initial evaluations were carried out using a slightly modified version of the AlexNet model. Following this, the experimentation broadened to include a range of models, each tested with the different dataset variations (with or without augmentation, for both raw images and vectorscope images). This approach allowed for a comprehensive understanding of how different models perform under varying circumstances. Lastly, the selected models underwent testing in a more realistic setting. This involved scenarios where frames from a single movie could potentially appear in both the training and test datasets. This aspect of the testing provided insights into how well the models could generalize to more practical situations.

4.1 Model Selection

In this section, I will elaborate on the selection of three specific models for experimentation: AlexNet [18], VGG-16 [19], and ResNet50 [20].

4.1.1 AlexNet

AlexNet [Figure 4.1] represents a notable convolutional neural network (CNN) model that left a lasting mark on the field of computer vision and deep learning. Introduced in 2012 by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton [18], it marked a significant shift by substantially improving the performance of image classification tasks in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). One of the distinct characteristics of AlexNet is its depth and capacity to extract intricate hierarchical features from images. Comprising a

total of eight layers of neurons, this architecture is made up of five convolutional layers followed by three fully connected layers. An innovation it introduced was the integration of the ReLU (Rectified Linear Unit) activation function, which not only addressed the vanishing gradient problem but also expedited training convergence. The rationale behind selecting AlexNet for your research is solid due to its transformative impact on the realm of computer vision. It served as a breakthrough, showcasing that deep neural networks could outperform traditional methods in image classification tasks. Its architectural innovations, including the use of multiple layers and ReLU activations, laid a strong foundation for subsequent advancements in the field of deep learning architectures.

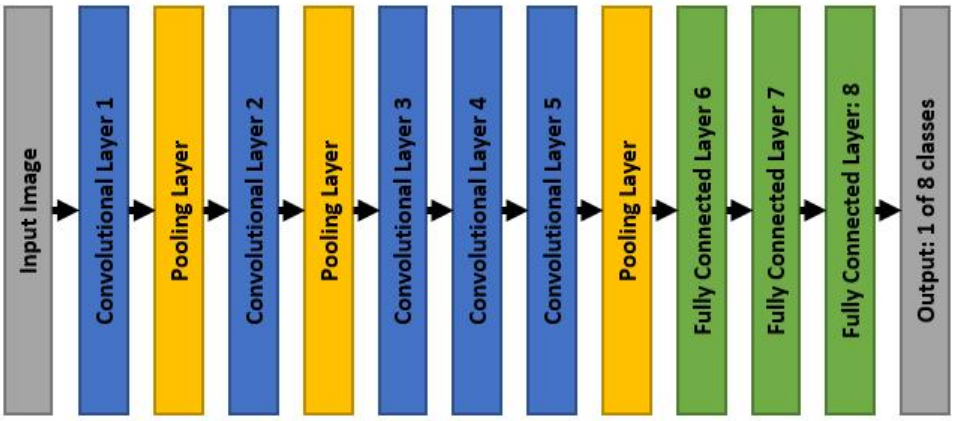

Figure 4.1: AlexNet model diagram

4.1.2 VGG-16

VGG-16 [Figure 4.2] stands as a convolutional neural network (CNN) architecture that occupies a distinct place within the realm of computer vision. Its notable characteristic lies in its deep structure, comprising 16 layers, which contributes to its capability in capturing intricate features from images. VGG-16 gained prominence for its strong performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and its ability to handle complex image classification tasks. Incorporating VGG-16 into my research is a well-justified decision, based on its consistent usage in previous studies exploring both cinematic and CNN-related applications. This architecture has been a recurrent choice in numerous papers investigating visual analysis within the cinematic domain. Its demonstrated success across various visual recognition tasks makes it a fitting choice for my current investigation. By opting for VGG-16, my intention is to leverage a model that has already proven its capacity to effectively handle intricate

features and patterns in cinematic frames.

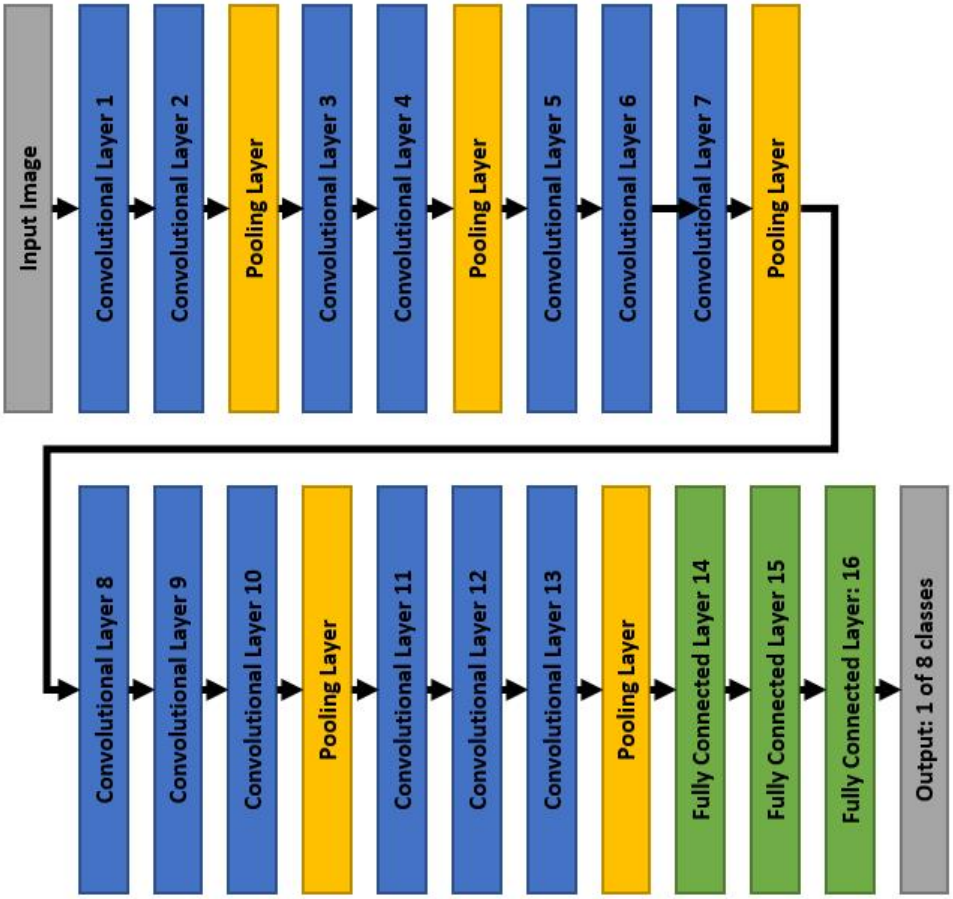

Figure 4.2: VGG-16 model diagram

4.1.3 ResNet-50

ResNet-50 [Figure 4.3] is a notable convolutional neural network (CNN) architecture that holds a distinct significance in the realm of computer vision. Its distinguishing feature lies in its deep structure, comprising 50 layers, which results in a remarkable capability to capture intricate features from images. Introduced in 2015, ResNet-50 gained prominence for its exceptional performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), demonstrating its adeptness in handling complex image classification tasks. Incorporating ResNet-50 into my research stems from its proven track record of outperforming AlexNet, another

model I've utilized. ResNet-50 has exhibited superior performance, especially when dealing with intricate visual recognition tasks. Its demonstrated ability to understand complex features in images aligns with my research's primary objective of enhancing genre classification accuracy in movie frames. By leveraging ResNet-50, I aim to capitalize on its established superiority over AlexNet, thus maximizing the potential for more accurate and effective genre classification in my investigation.

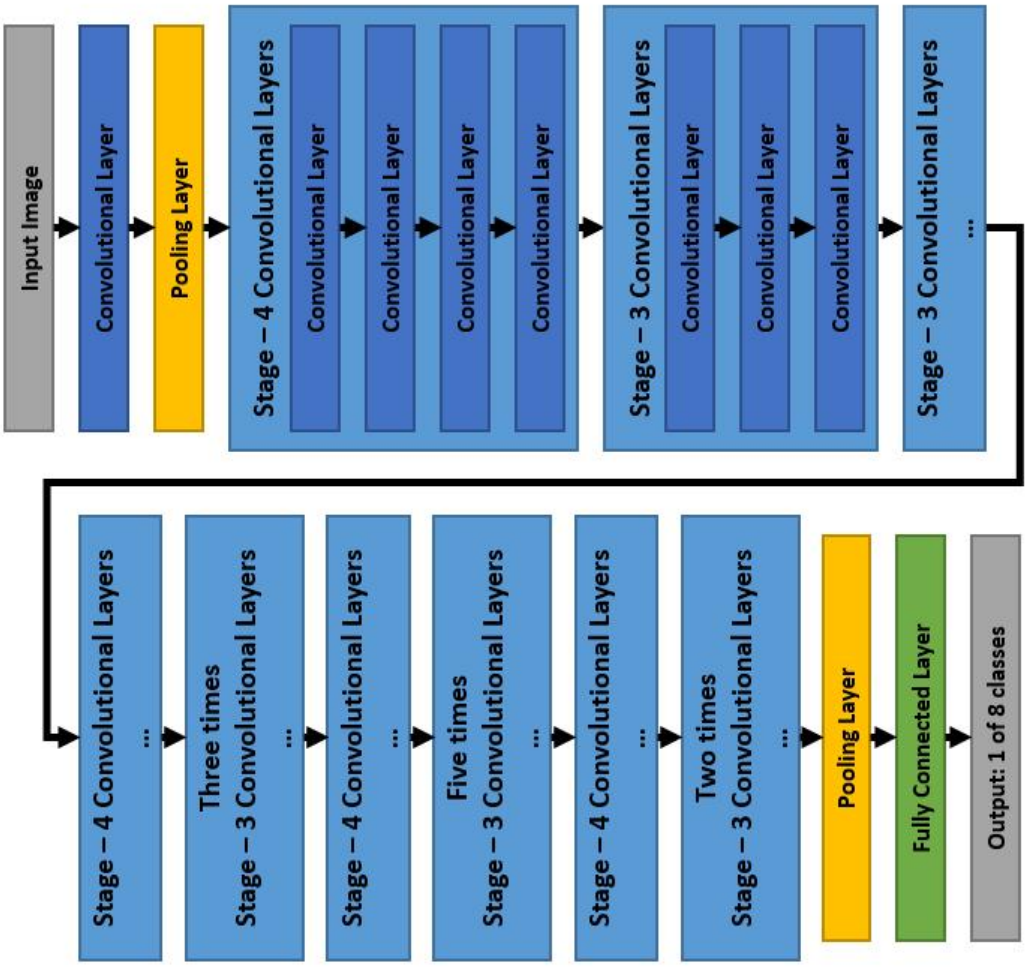

Figure 4.3: ResNet-50 model diagram

The only changes made to the models involve adapting the output of their final layers to align with the total number of classes, which in this instance is 8. Additionally, it's worth noting that all the models presented here offer the option to utilize pretrained versions trained on the ImageNet dataset [10].

4.2 Loss Function

The subsequent step was to choose the objective function, a significant component in the field of machine learning and optimization. This function measures the difference between the predicted values produced by a model and the actual ground truth values present in the training data. The primary aim of a machine learning algorithm is to minimize this difference, often referred to as the loss function. In supervised learning tasks, the loss function calculates the error between the predicted outputs and the real target values. It provides essential guidance for adjusting the model's parameters during the training process, with the goal of decreasing the error and enhancing its predictive abilities. Various types of loss functions are available, each tailored to specific tasks. For instance, in regression tasks, the Mean Squared Error (MSE) is a commonly used metric, while in classification tasks, the Cross-Entropy Loss (also known as Log Loss) is frequently employed. The choice of a suitable loss function depends on the nature of the problem and the intended behavior of the model during the training phase.

I opted for the Cross-Entropy Loss as my choice of loss function due to its suitability for classification tasks like the one I'm addressing. Cross-Entropy Loss is particularly effective when dealing with problems involving multiple classes, as in my genre classification endeavor. It not only captures the disparity between predicted and actual class probabilities but also penalizes larger errors more heavily, leading to more robust and accurate model training. Given that my research involves assigning movie frames to specific genres, the nature of the problem aligns well with the strengths of the Cross-Entropy Loss in handling multi-class classification scenarios.

In addition to employing the Cross-Entropy Loss, I implemented the Stochastic Gradient Descent (SGD) optimization algorithm to fine-tune the parameters of my models during training. SGD is a fundamental optimization technique commonly used in machine learning tasks, including neural network training. Stochastic Gradient Descent operates by iteratively updating the model's parameters based on the gradients of the loss function with respect to these parameters. Unlike traditional gradient descent, which computes gradients over the entire training dataset, SGD performs updates on smaller subsets, or mini-batches, of the data. This approach introduces randomness into the optimization process, which can help escape local minima and accelerate convergence. SGD is known for its efficiency and ability to handle large datasets, making it particularly advantageous when working with extensive image datasets like mine. The inherent stochastic nature of SGD allows the optimization process to navigate through the parameter space in a more dynamic manner, ultimately facilitating the model's convergence to a suitable solution. By combining the Cross-Entropy Loss with Stochastic Gradient

Descent, I aimed to enhance the training process of my models, leading to improved accuracy and efficiency in classifying movie frames into their respective genres.

The final training-related aspect I chose to incorporate was the step-down policy. This strategy involves a dynamic adjustment of the learning rate during the training process, and it is commonly utilized in machine learning models, including neural networks. The primary objective of this approach is to find a balance between swift convergence in the initial training stages and fine-tuning in the later stages. At its core, the step-down policy encompasses the reduction of the learning rate, generally by a predetermined factor, following a specific count of training epochs or iterations. This reduction enables the optimization process to take more substantial strides in the initial phases, thereby facilitating rapid convergence. As training advances and the optimization process approaches a potential minimum, the learning rate is diminished, allowing for more delicate refinements to the model parameters. By integrating the step-down policy, the training process becomes more adaptable and effective, avoiding overshooting and permitting the model to settle into a more optimal local minimum. This technique frequently yields enhanced generalization and overall convergence of the model. Its utility becomes particularly evident when dealing with intricate datasets or deep architectures, where a well-balanced learning rate adjustment strategy is pivotal for successful training. For the implementation of this policy, I employed the `StepLR()` function from the `torch.optim` library [Listing B.1]. This function necessitates three parameters: the optimizer (SGD), the step size (indicating the number of epochs before the learning rate reduction), and the gamma factor (used as a multiplicative coefficient for the learning rate decrease).

4.3 Hyperparameter Tuning

In the pursuit of building robust and accurate models for the genre classification of movie frames, an essential phase involves fine-tuning the hyperparameters of the chosen neural network architectures. Hyperparameters play a critical role in shaping the behavior and performance of the models, influencing aspects such as convergence speed, generalization ability, and overall effectiveness. This section outlines the process of hyperparameter tuning and details the initial trials conducted to identify optimal configurations. Hyperparameter tuning is a crucial step in model development, requiring a systematic exploration of various parameter settings to strike the right balance between underfitting and overfitting. The aim is to discover hyperparameter values that facilitate the model's capacity to generalize well on unseen data while maintaining strong performance on the training data. Additionally, the section delves into the initial trials, which serve as a starting point for gauging the model's response to different hyperparameter setups.

The hyperparameters under my consideration encompass:

1. Batch size dictates the number of training examples utilized in each iteration during gradient descent. A larger batch size can lead to faster convergence but requires more memory.
2. Learning Rate controls the step size taken in the direction of minimizing the loss function during optimization. A higher learning rate may accelerate convergence, but a lower value might lead to more stable and accurate results.
3. Momentum is a factor that introduces inertia into the optimization process, allowing the optimization algorithm to accumulate velocity in directions with consistent gradients. It helps overcome local minima and accelerates convergence.
4. Weight decay is a regularization technique that adds a penalty term to the loss function based on the magnitudes of the model's parameters. It discourages overly large parameter values, reducing the risk of overfitting.
5. Number of epochs denotes how many times the entire training dataset is iterated over during training. It affects how many times the model updates its parameters and can influence both underfitting and overfitting.
6. Step size indicates the frequency at which the learning rate is adjusted during training. It determines how often the learning rate is reduced to facilitate convergence as training progresses.
7. Gamma is a multiplier applied to the learning rate during each step size interval. It governs the extent of the learning rate reduction and impacts the model's adaptability and convergence behavior.

The selection of the batch size was influenced by the available computational memory. In the beginning, I opted for larger batches containing 256 images; however, this proved to be excessive and strained the GPU's capacity. To mitigate this issue, I progressively reduced the batch size until the challenges were resolved, ultimately settling at 64. Momentum and weight decay values were chosen arbitrarily.

Among the parameters requiring significant attention, the adjustment of the learning rate and the determination of the number of epochs assumed central importance in the optimization process. Given the incorporation of the step-down policy in my implementation, a strategic approach was taken for these critical parameters. I started with relatively high learning rates, understanding that they would be reduced as training advanced. It's important to note that different learning rate values were used for the original raw frames and their corresponding vectorscope representations. However, the initial adoption of these higher learning rates posed a challenge with loss values resulting in `NaN` (Not-a-Number) occurrences, impacting the training process. To address this issue, a considered course of action was taken. Subsequent adjustments to the learning rate played a role in addressing this problem,

restoring training stability. Eventually, after iterative refinements, learning rates were set at 0.005 for raw frames and 0.001 for vectorscope representations. This decision aimed to find a balance between fast convergence and stable learning, enabling the model's learning progression while preventing undesirable fluctuations during training.

The appropriate number of epochs was identified by conducting the training and assessing the loss plots, as well as the associated loss values at various epoch checkpoints. This analysis revealed the stage at which the training loss consistently decreased, while the test loss either leveled off or displayed signs of increase. This moment served as a significant indicator, signifying the delicate balance between model convergence and the potential for overfitting. Figures 4.4, 4.5, and 4.6 depict the loss plots corresponding to the preliminary experiments carried out on the raw frames across various data augmentation approaches on AlexNet model.

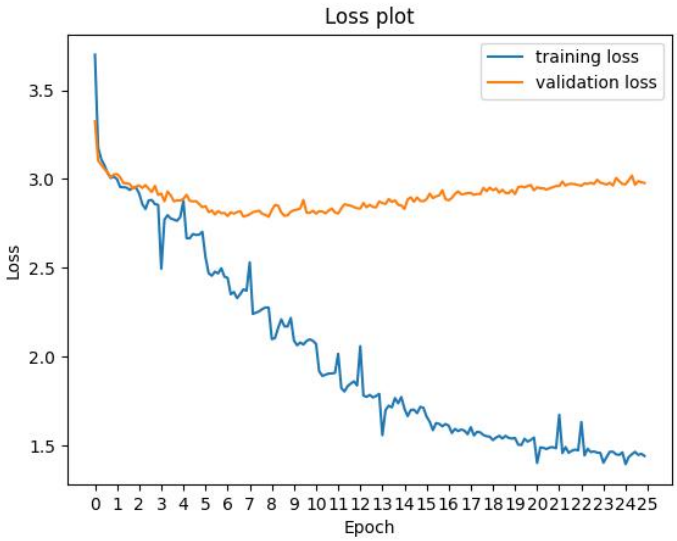

Figure 4.4: Loss plot for raw frames without data augmentation, ran for 25 epochs

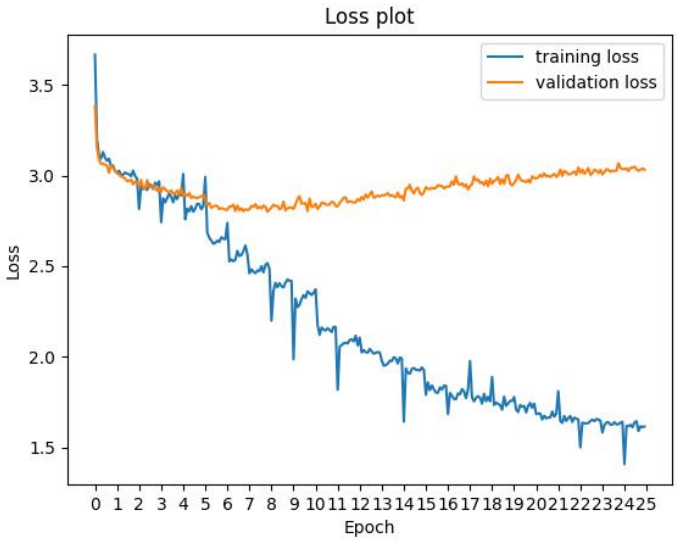

Figure 4.5: Loss plot for raw frames with data augmentation on all samples, ran for 25 epochs

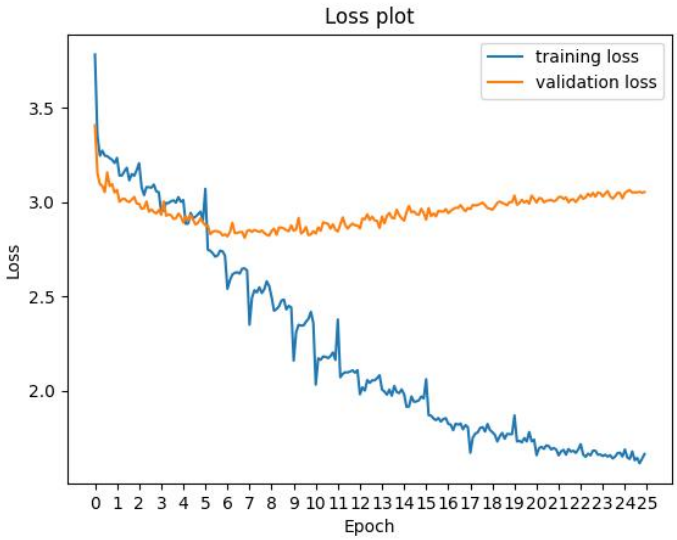

Figure 4.6: Loss plot for raw frames with data augmentation on rare classes, ran for 25 epochs

Upon careful examination of the provided plots, a clear pattern emerges: the onset of overfitting becomes noticeable before the 10th epoch, as evident from

the test loss exhibiting an upward trend. Notably, when considering scenarios involving data augmentation, as depicted in Figures 4.5 and 4.6, a distinct pattern unfolds, with the initial training loss surpassing that of the test loss. This pattern highlights the influence of data augmentation on early model training stages. The inclusion of the augmented dataset appears to impact the training process, resulting in discernible loss dynamics between the training and test datasets. This observation emphasizes the need for a comprehensive exploration of these dynamics to ensure the model's resilience and adaptability. One plausible reason for this phenomenon could be the increased similarity of data in the examples involving data augmentation. This similarity might extend the learning process as the model aims to recognize these analogous patterns, contributing to the observed divergence between training and test loss.

Similar situation appears when training the model on vectorscope representation of frames what can be seen on Figures 4.7, 4.8, and 4.9.

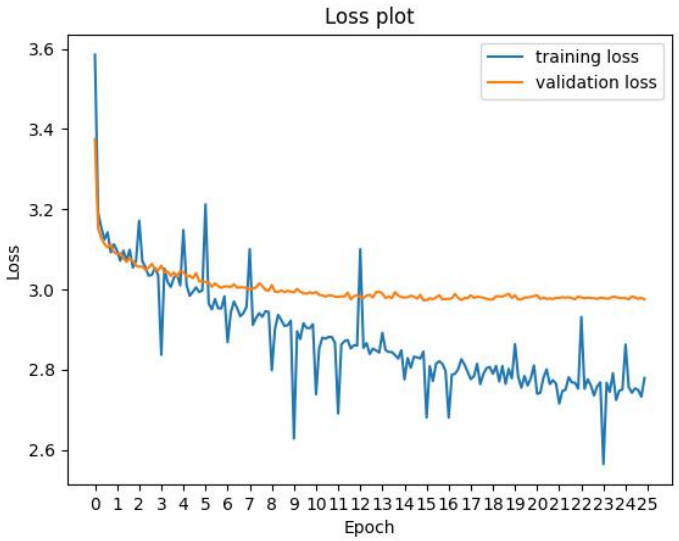

Figure 4.7: Loss plot for vectorscope frames without data augmentation, ran for 25 epochs

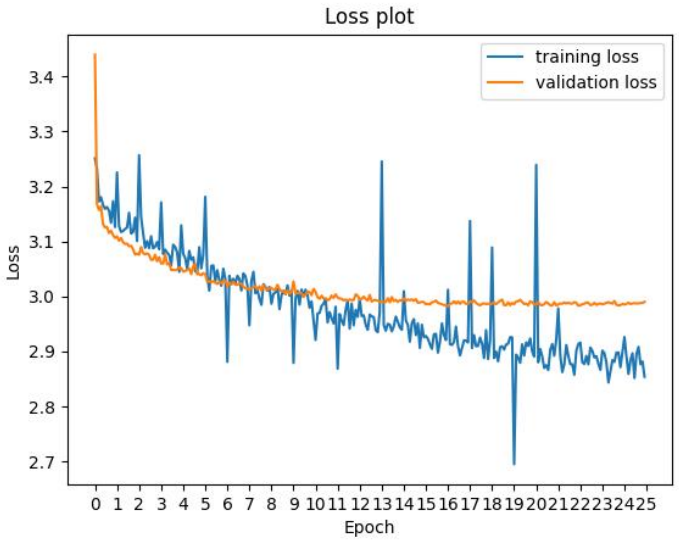

Figure 4.8: Loss plot for vectorscope frames with data augmentation on all samples, ran for 25 epochs

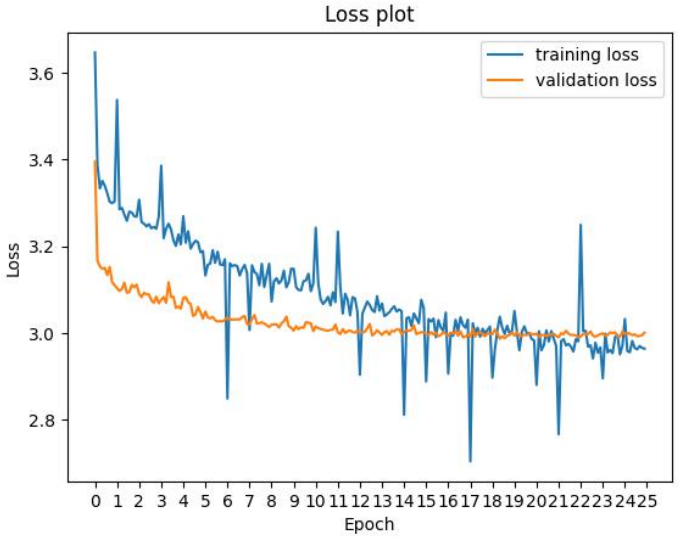

Figure 4.9: Loss plot for vectorscope frames with data augmentation on rare classes, ran for 25 epochs

In the case of vectorscope representations, the occurence of overfitting becomes evident around the 10th epoch. Similarly, in the experiments involving

data-augmented datasets, a parallel issue arises, related to that observed in the raw frames experiments. Notably, training losses surpass test losses, and the rationale behind this phenomenon may align with the explanation for the divergence witnessed in the raw frames scenario. Another distinction worth noting is that, unlike the raw frames scenario, the test set's loss doesn't experience an increase but rather stabilizes on a plateau.

The outcomes of the initial runs are illustrated in Tables 4.1 and 4.2. The metrics employed for assessing the experiments were:

- Precision is the proportion of correctly predicted positive instances out of all instances predicted as positive. It focuses on the accuracy of positive predictions.
- Recall, also known as Sensitivity or True Positive Rate, is the proportion of correctly predicted positive instances out of all actual positive instances. It emphasizes the model's ability to capture all relevant instances.
- F1-score, is the harmonic mean of precision and recall. It provides a balanced measure that considers both false positives and false negatives, making it useful for imbalanced datasets, like the one I'm dealing with.
- Support represents the number of actual occurrences of each class in the dataset. It gives context to precision and recall values by indicating the size of each class.
- Micro average calculates precision, recall, and F1-score across all classes by summing up the individual true positives, false positives, and false negatives and then calculating the metrics. It's suitable for imbalanced datasets.
- Macro average calculates precision, recall, and F1-score for each class separately and then takes their average. It treats all classes equally and doesn't consider class imbalances.
- Weighted average calculates precision, recall, and F1-score for each class separately and then takes their weighted average, where the weight is the support of each class. It addresses class imbalances.
- Samples average calculates precision, recall, and F1-score for each instance separately and then takes their average. It's used for multilabel classification when each instance can belong to multiple classes.

Genre	Precision	Recall	F1-score	Support
action	0.531	0.652	0.585	1921
adventure	0.539	0.647	0.588	1882
comedy	0.603	0.725	0.658	2481
crime	0.431	0.444	0.437	705
drama	0.551	0.721	0.624	2587
horror	0.455	0.492	0.473	865
romance	0.423	0.413	0.418	555
thriller	0.369	0.340	0.354	588
micro avg	0.532	0.630	0.577	11584
macro avg	0.488	0.554	0.517	11584
weighted avg	0.527	0.630	0.573	11584
samples avg	0.532	0.649	0.571	11584

Table 4.1: Results of AlexNet run for raw frames without data augmentation, for 25 epochs

Genre	Precision	Recall	F1-score	Support
action	0.451	0.544	0.493	1921
adventure	0.456	0.473	0.464	1882
comedy	0.516	0.741	0.608	2481
crime	0.293	0.140	0.190	705
drama	0.458	0.768	0.574	2587
horror	0.382	0.320	0.348	865
romance	0.286	0.178	0.220	555
thriller	0.269	0.073	0.115	588
micro avg	0.457	0.542	0.496	11584
macro avg	0.389	0.405	0.377	11584
weighted avg	0.435	0.542	0.470	11584
samples avg	0.457	0.566	0.494	11584

Table 4.2: Results of AlexNet run for vectorscope frames without data augmentation, for 25 epochs

Given the substantial dataset imbalance, my primary focus will be on evaluating the F1-score and the weighted average. Tables 4.1 and 4.2 demonstrate

that raw frames exhibit slightly better results compared to their vectorscope counterparts. It's also important to highlight the significant computational advantage of vectorscope calculations, with a runtime of approximately 10-15 minutes, notably faster than the 30-45 minutes required for raw frames. Another noteworthy observation is the reduced F1-score achieved through vectorscope calculations, especially for classes with limited representation, which I attempted to address through data augmentation techniques.

Genre	F1-score				Support
	raw frames		vec frames		
	aug all	aug rare	aug all	aug rare	
action	0.573	0.564	0.496	0.488	1921
adventure	0.578	0.585	0.454	0.434	1882
comedy	0.663	0.661	0.599	0.611	2481
crime	0.416	0.430	0.153	0.218	705
drama	0.628	0.628	0.567	0.572	2587
horror	0.452	0.455	0.328	0.377	865
romance	0.404	0.411	0.114	0.181	555
thriller	0.341	0.363	0.075	0.155	588
weighted avg	0.567	0.542	0.470	0.468	11584

Table 4.3: Results of AlexNet runs with data augmentation, for 25 epochs

Table 4.3 provides a comprehensive depiction of the F1-score values across a range of conducted experiments. The data presented in the table reveals that the average F1-score results exhibit a slight reduction for the augmentation approach targeting specifically the less represented frames. However, this reduction is counterbalanced by a notable increase in the F1-scores for individual genres classified as rare (e.g., the F1-score for the thriller genre escalates from 0.075 for all augmented frames to 0.155 for the augmented frames focusing on rare instances), particularly apparent in the context of vectorscope representations. Examining the context of raw frames, an interesting trend emerges: there is an improvement in the model's performance with regard to less commonly represented frames, though this enhancement is relatively small. Additionally, it is worth noting that for raw frames, the outcomes without any augmentation demonstrate better results.

4.4 Modifications to the Experiment - Merging Classes

Because the F1-scores were low in prior experiments [Tables 4.1, 4.2 and 4.3], particularly for the vectorscope representations of frames, I chose to simplify the experiment by combining classes based on my own criteria.

Up to this point, I have been addressing a multi-label classification problem in which a movie could be assigned to a maximum of two genres out of the following eight:

- Drama,
- Comedy,
- Action,
- Adventure,
- Horror,
- Crime,
- Thriller,
- Romance.

Analyzing the initial class distribution, which I obtained during the first phase of class reduction, where I assumed that a movie could belong to a maximum of two genres (as shown in Figure A.1, I noticed that the second most prominent category comprises the combination of Action and Adventure classes. Similarly, I found that Comedy and Romance frequently appear together, and thus, I merged them into a single category. Moreover, I identified Thriller, Horror, and Crime as genres that could be grouped due to their related semantic meanings. Consequently, I condensed the genres into four classes, each with the following representations [4.10]:

- Action and Adventure: 16277 (later referred to as Act_Adv),
- Thriller, Horror and Crime: 12869 (Thr_Hor_Cri),
- Comedy and Romance: 10768 (Com_Rom),
- Drama: 5931 (Drama).

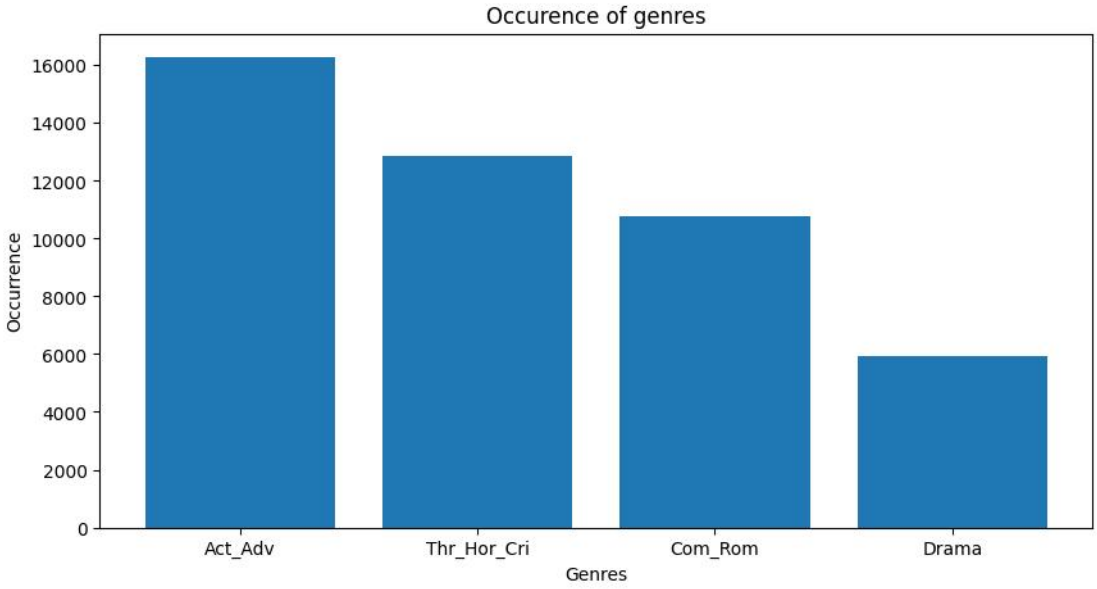

Figure 4.10: Data distribution for the single label classification task

Subsequently, I implemented data augmentation for all frames based on the current distribution, as depicted in Figure 4.10. The characteristics of the image alterations precisely match the transformations applied to frames in the previous experiments. These transformations were applied to 90% of the Drama frames, 40% of the Thr_Hor_Cri frames, 30% of the Com_Rom frames, and 10% of the Act_Adv frames. Consequently, I achieved the following distribution for the training set, as shown in Figure 4.11:

- Thr_Hor_Cri: 15336,
- Act_Adv: 15194,
- Com_Rom: 11865,
- Drama: 9571.

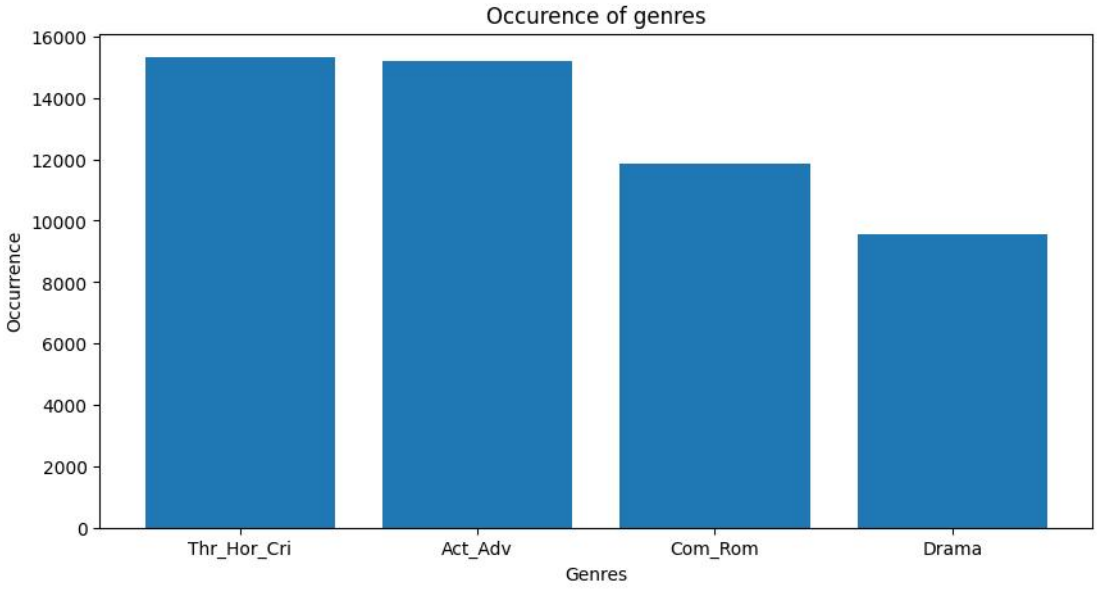

Figure 4.11: Data distribution for the single label classification task after the data augmentation

Regarding the testing set it wasn't altered by the transformations so it's distribution is similar to the original one depicted on Figure 4.10:

- Act_Adv: 2486,
- Thr_Hor_Cri: 1894,
- Com_Rom: 1604,
- Drama: 893.

4.4.1 Experiments on the Modified Dataset

The initial experiments conducted on the new dataset aimed to determine the optimal number of epochs for the final experiments. I maintained the use of the three models previously selected, and I introduced an additional model in subsequent experiments. Furthermore, I retained the same loss function employed in the earlier experiments. Simplifying the problem to single-label classification allowed for the evaluation of the model's performance based on accuracy as well.

The initial experiment utilized an AlexNet model and ran for 10 epochs, as shown in Figure 4.12. In the plot, it is evident that the loss begins to stabilize around the 4th epoch. Regarding the model evaluation, as presented in Table 4.4, the parameter values were improved compared to the multi-label classification experiments. However, the least represented class, which is Drama in this case,

still yielded the worst result. With the implementation of single-label classification, I achieved an accuracy of 52.26%, enabling a direct comparison with the accuracies of other models.

Genre	Precision	Recall	F1-score	Support
drama	0.415	0.217	0.285	893
act_adv	0.608	0.605	0.607	2486
com_rom	0.519	0.496	0.507	1604
thr_hor_cri	0.458	0.581	0.512	1894
micro avg	0.523	0.523	0.523	6877
macro avg	0.500	0.475	0.478	6877
weighted avg	0.521	0.523	0.516	6877
samples avg	0.523	0.523	0.523	6877

Table 4.4: Results of AlexNet run for raw frames, for 10 epochs

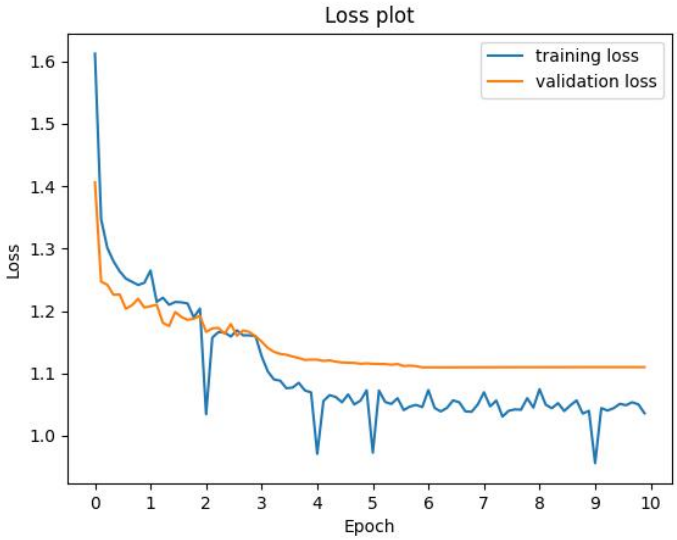

Figure 4.12: Loss plot of AlexNet run for raw frames, for 10 epochs

The second model evaluated on the updated dataset was VGG-16 ran for 8 epochs [Tbale 4.13]. Once more, the model begins to exhibit signs of overfitting around the 4th epoch, with slightly improved results compared to the AlexNet, especially for the Drama class [Figure 4.5]. VGG-16 achieved an accuracy of 54.75%. The sole aspect where VGG-16 lags behind is the computation time, which will be

emphasized in subsequent sections.

Genre	Precision	Recall	F1-score	Support
drama	0.446	0.225	0.299	893
act_adv	0.637	0.631	0.634	2486
com_rom	0.522	0.541	0.531	1604
thr_hor_cri	0.490	0.596	0.538	1894
micro avg	0.547	0.547	0.547	6877
macro avg	0.524	0.498	0.500	6877
weighted avg	0.545	0.547	0.540	6877
samples avg	0.547	0.547	0.547	6877

Table 4.5: Results of VGG-16 run for raw frames, for 8 epochs

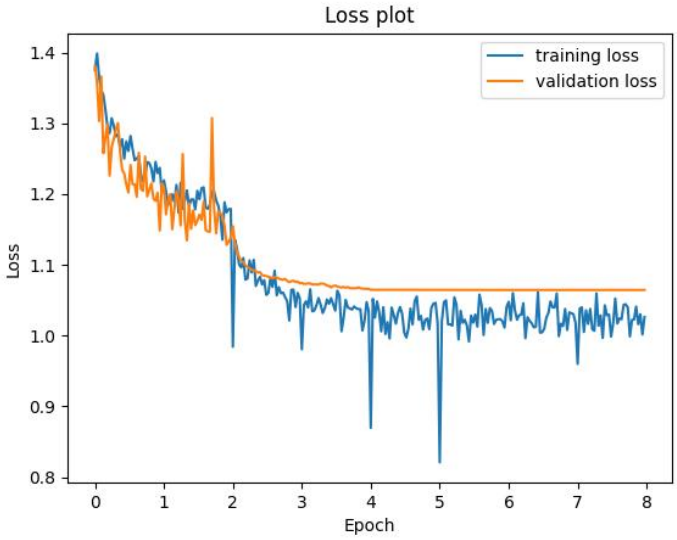

Figure 4.13: Loss plot of VGG-16 run for raw frames, for 8 epochs

The final model tested was ResNet-50, trained for 8 epochs [Figure 4.14]. As expected, overfitting begins even before the 4th epoch. ResNet outperforms the previous models by a significant margin, starting with an accuracy of 60.46%. Regarding the other metrics presented in Table 4.6, the scores are more balanced and notably higher for the less represented Drama class.

Genre	Precision	Recall	F1-score	Support
drama	0.542	0.420	0.473	893
act_adv	0.685	0.698	0.692	2486
com_rom	0.583	0.575	0.579	1604
thr_hor_cri	0.543	0.594	0.568	1894
micro avg	0.605	0.605	0.605	6877
macro avg	0.588	0.572	0.578	6877
weighted avg	0.604	0.605	0.603	6877
samples avg	0.605	0.605	0.605	6877

Table 4.6: Results of ResNet-50 run for raw frames, for 8 epochs

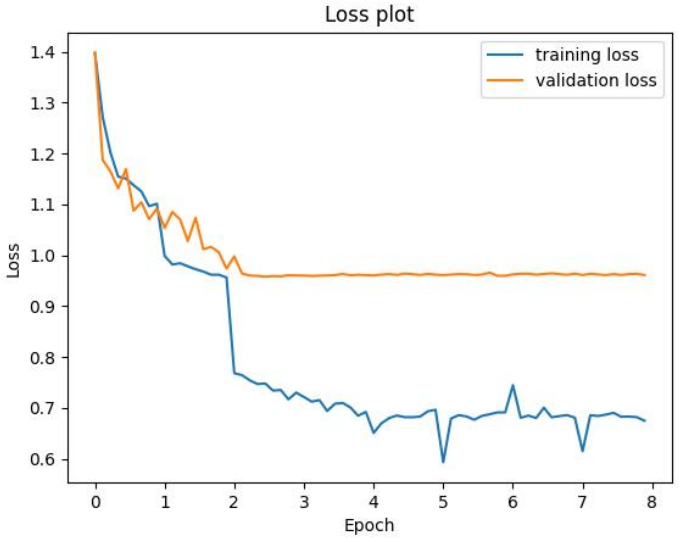

Figure 4.14: Loss plot of ResNet-50 run for raw frames, for 8 epochs

Following the trial runs, I decided to use 4 epochs, a choice that was subsequently validated in experiments involving vectorscope representations.

4.4.2 Model Selection - Vision Transformer

With the experiment's slight modification, I opted to explore another model, specifically the Vision Transformer [21] introduced in 2021. The Vision Transformer (ViT) represents a groundbreaking advancement in deep learning architecture, transforming the field of computer vision. In contrast to traditional

Convolutional Neural Networks (CNNs), ViT leverages self-attention mechanisms to capture global image patterns, replacing localized operations. It dissects images into patches and treats them as sequences, resembling the methodology of Natural Language Processing (NLP) models. This shift has resulted in enhanced image classification capabilities and adaptability to tasks such as object detection and image captioning.

ViT's versatility extends to fine-tuning pre-trained models for specific tasks, making it a popular choice across various domains. Additionally, it has facilitated the development of multimodal models, bridging visual and textual data for applications in natural language understanding and more. Ultimately, ViT redefines computer vision and propels artificial intelligence forward by effectively handling diverse data sources. In my experiment, I'm specifically utilizing the ViT-Base-16 model to assess its performance and suitability in the context of my research on movie genre classification.

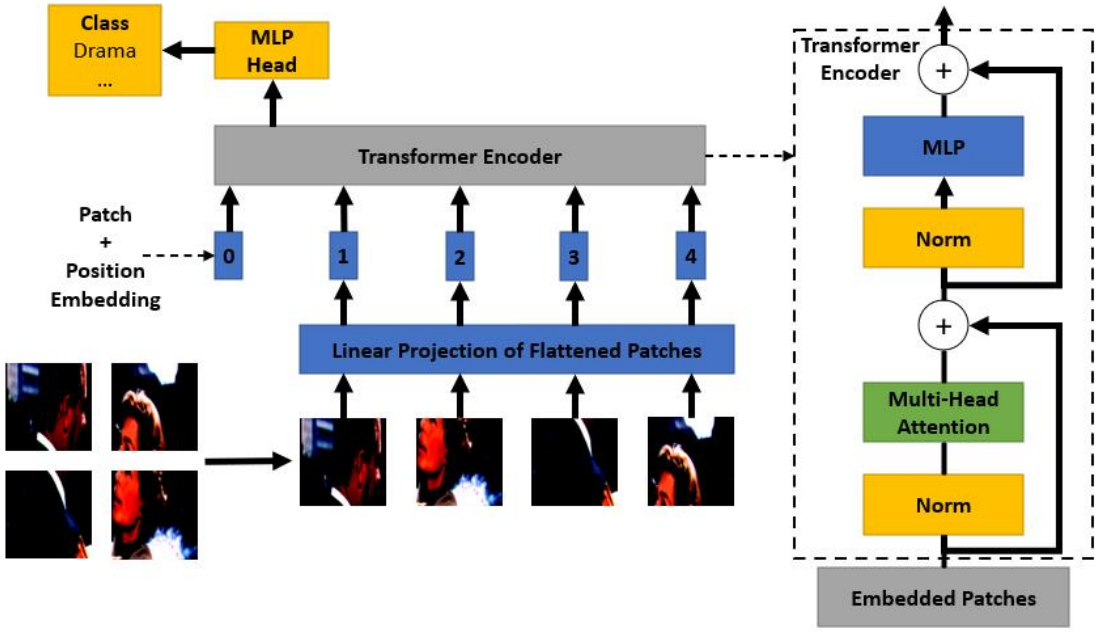

Figure 4.15: ViT model diagram [21]

5. Results

After conducting thorough analyses in the preceding sections, I have chosen to evaluate four distinct models in my research:

1. AlexNet,
2. VGG-16,
3. ResNet-50,
4. Vision Transformer.

Using these models, I will explore four variations of the experiment, each with its own unique dataset distribution:

1. raw frames randomly distributed between the training and testing sets.
2. vectorscope representations of frames randomly allocated between the training and testing sets.
3. raw frames distributed in a manner that ensures frames from the same movie are not present in both the training and testing sets.
4. vectorscope representations of frames distributed to prevent frames from the same movie appearing in both the training and testing sets.

The purpose of these sixteen experiment variations is to provide in-depth insights into how the selected models perform with different dataset distributions, ultimately advancing our understanding of movie genre classification. All experiments were conducted for 4 epochs across the four aforementioned models.

5.1 Raw Frames - Random Distribution

To begin, I conducted the test on raw frames with random distribution, resulting in the following accuracy rates:

1. AlexNet: 50.49%,
2. VGG-16: 54.08%,
3. ResNet-50: 60.54%,

4. Vision Transformer: 56.07%.

Similar to the preliminary experiments, the Resnet-50 model consistently delivers the best results, including computation time per one epoch (all experiments were performed on the same GPU - NVIDIA RTX A6000):

1. AlexNet: 35 minutes,
2. VGG-16: 102 minutes,
3. ResNet-50: 35 minutes,
4. Vision Transformer: 47 minutes.

In these experiments, ResNet has demonstrated both superior effectiveness and efficiency. The loss plot of the best model is shown in Figure 5.1.

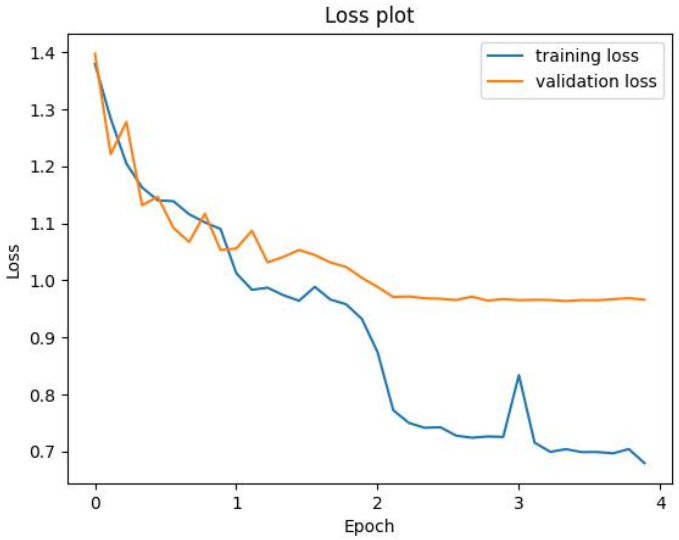

Figure 5.1: Loss plot of ResNet-50 run for raw frames, for 4 epochs

As indicated in Table 5.1, the metric results closely resemble those of the trial runs, with consistent performance across all classes and a minor reduction in performance for the least represented class, Drama.

Genre	Precision	Recall	F1-score	Support
drama	0.534	0.449	0.488	893
act_adv	0.677	0.688	0.682	2486
com_rom	0.592	0.576	0.584	1604
thr_hor_cri	0.554	0.574	0.568	1894
micro avg	0.605	0.605	0.605	6877
macro avg	0.589	0.577	0.582	6877
weighted avg	0.604	0.605	0.604	6877
samples avg	0.605	0.605	0.605	6877

Table 5.1: Results of ResNet-50 run for raw frames, for 4 epochs

As depicted in Table 5.2, the majority of classes were accurately predicted. However, Drama was consistently misclassified with other genres, Com_Rom was frequently misclassified as Thr_Hor_Cri and Act_Adv, and Act_Adv was often misclassified as Thr_Hor_Cri and vice versa.

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	401	164	134	194
Act_Adv	106	1710	228	442
Com_Rom	131	276	924	273
Thr_Hor_Cri	113	377	276	1128

Table 5.2: Confusion matrix of ResNet-50 run for raw frames, for 4 epochs

5.2 Vectorscope Representations - Random Distribution

In the second series of experiments, we conducted tests by running models on vectorscope representations of frames with a random distribution, which led to the following accuracy results:

1. AlexNet: 43.11%,
2. VGG-16: 45.73%,
3. ResNet-50: 42.69%,
4. Vision Transformer: 43.90%.

While the differences in accuracy are not substantial, ResNet-50 performed the least effectively among these models, with the highest accuracy achieved by

VGG-16. However, it's worth noting that this superior performance came at a cost, as VGG-16 required the longest training time per epoch in this set of experiments, as shown below:

1. AlexNet: 17 minutes,
2. VGG-16: 59 minutes,
3. ResNet-50: 19 minutes,
4. Vision Transformer: 26 minutes.

Despite the longer training time, VGG-16 outperformed the other models. You can view the loss plot for the VGG-16 experiment in Figure 5.2.

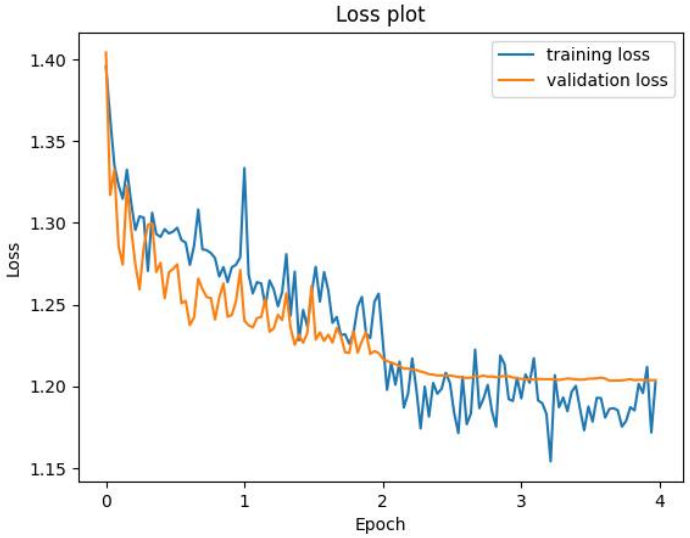

Figure 5.2: Loss plot of VGG-16 run for vectorscope representations, for 4 epochs

The results obtained from the vectorscope experiments cannot be directly compared to the trial runs, as they generally yielded lower accuracy. This is expected since vectorscope representations contain less data than raw frames. The report for the best experiment in this section is detailed in Table 5.3.

Genre	Precision	Recall	F1-score	Support
drama	0.332	0.142	0.199	893
act_adv	0.537	0.508	0.522	2486
com_rom	0.444	0.494	0.468	1604
thr_hor_cri	0.409	0.508	0.453	1894
micro avg	0.457	0.457	0.457	6877
macro avg	0.430	0.413	0.410	6877
weighted avg	0.453	0.457	0.448	6877
samples avg	0.457	0.457	0.457	6877

Table 5.3: Results of VGG-16 run for vectorscope representations, for 4 epochs

Regarding the confusion matrix [Table 5.4], it's evident that for vectorscope representations, correct classifications are not as straightforward. In the case of the least represented class, Drama, the incorrect prediction Thr_Hor_Cri occurred nearly as frequently as the correct prediction. The same pattern is observed for the other classes, with the exception of Act_Adv, which is often correctly classified.

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	127	236	203	327
Act_Adv	90	1263	435	698
Com_Rom	56	388	793	367
Thr_Hor_Cri	465	357	276	962

Table 5.4: Confusion matrix of VGG-16 run for vectorscope representations, for 4 epochs

5.3 Raw Frames - "Separate" Distribution

In the subsequent series of tests, I evaluated the models using raw frames that were further divided in such a way that frames from the same movie could be present in both the training and testing sets. The resulting accuracies were as follows:

1. AlexNet: 46.86%,
2. VGG-16: 48.78%,
3. ResNet-50: 51.55%,
4. Vision Transformer: 50.24%.

As anticipated, in comparison to the previous experiment conducted with raw frames, ResNet-50 emerged as the most suitable solution for the task, with the following training times:

1. AlexNet: 35 minutes,
2. VGG-16: 116 minutes,
3. ResNet-50: 42 minutes,
4. Vision Transformer: 44 minutes.

In this specific instance, ResNet-50 exhibited a less efficient training time compared to AlexNet, with a significant gap. However, when taking its overall performance into consideration, it remains a recommended choice. You can examine the loss plot of the ResNet-50 experiment in Figure 5.3. The primary distinction between the loss plots of randomly distributed frames and those that are separated lies in the distance between the validation and training loss. This difference is a result of the frames in the testing and training sets being more dissimilar from each other than in previous experiments.

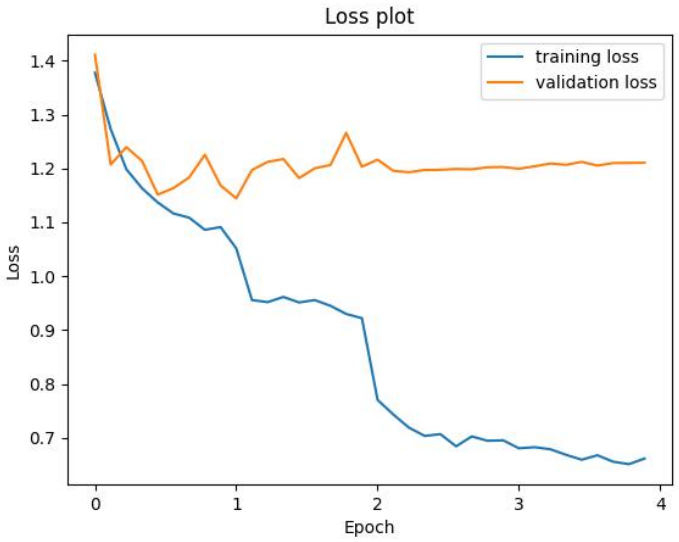

Figure 5.3: Loss plot of ResNet-50 run for raw frames, for 4 epochs

As indicated in Table 5.5, the results are predictably less favorable than those with random frames distribution. Nevertheless, an accuracy exceeding 50% remains valuable for future research, particularly as this experiment simulates a more realistic scenario.

Genre	Precision	Recall	F1-score	Support
drama	0.341	0.325	0.333	750
act_adv	0.621	0.592	0.606	2503
com_rom	0.478	0.487	0.482	1607
thr_hor_cri	0.486	0.513	0.500	2090
micro avg	0.516	0.516	0.516	6950
macro avg	0.481	0.480	0.480	6950
weighted avg	0.517	0.516	0.516	6950
samples avg	0.516	0.516	0.516	6950

Table 5.5: Results of ResNet-50 run for raw frames, for 4 epochs

Similarly to section 5.1 majority of the classes were predicted correctly [Table 5.6] with slightly lesser results, especially regarding the Drama class. When it comes to the Com_Rom it is often misclassified as Thr_Hor_Cri, which is often mistaken for Act_Adv.

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	244	141	160	205
Act_Adv	144	1483	289	587
Com_Rom	163	320	783	341
Thr_Hor_Cri	165	445	407	1073

Table 5.6: Confusion matrix of ResNet-50 run for raw frames, for 4 epochs

5.4 Vectorscope Representations - "Separate" Distribution

In the final series of experiments, I assessed the performance of models using vectorscope representations of frames with a distinct distribution, which resulted in the following accuracy outcomes:

1. AlexNet: 42.58%,
2. VGG-16: 43.81%,
3. ResNet-50: 42.68%,
4. Vision Transformer: 42.17%.

Once again, similar to the results in Section 5.2, the VGG-16 model performed the best. What's intriguing is the close similarity between the results of these

experiments and the ones in Section 5.2 despite the differences in the experimental modalities compared to raw frames. In the case of vectorscope representations, the absolute difference in performance between the separate distribution and random distribution experiments is approximately half the size. However, VGG-16 still lags in terms of computation time, with the following training times:

1. AlexNet: 17 minutes,
2. VGG-16: 47 minutes,
3. ResNet-50: 23 minutes,
4. Vision Transformer: 25 minutes.

Despite the extended training duration, VGG-16 surpassed the performance of the other models. You can examine the loss plot for the VGG-16 experiment in Figure 5.4.

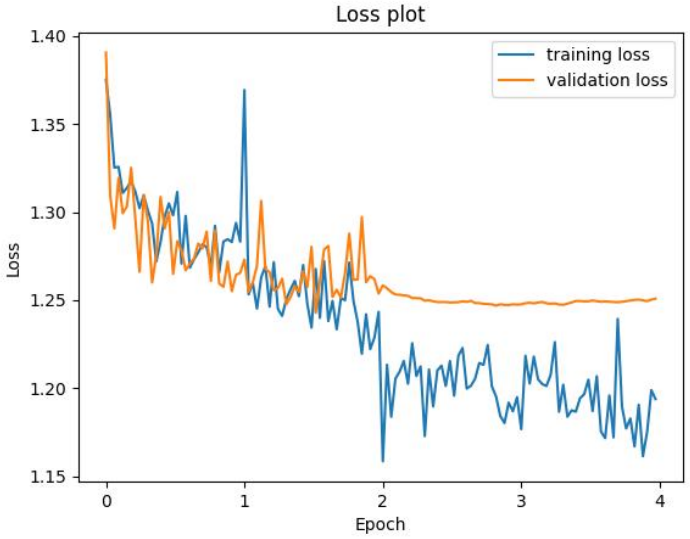

Figure 5.4: Loss plot of VGG-16 run for vectorscope representations, for 4 epochs

In this section, the outcomes closely resemble those in Section 5.2, showing lower accuracy than the experiments conducted on raw frames. The comprehensive report for the best experiment in this section is provided in Table 5.7.

Genre	Precision	Recall	F1-score	Support
drama	0.330	0.187	0.239	750
act_adv	0.476	0.437	0.456	2503
com_rom	0.405	0.428	0.416	1607
thr_hor_cri	0.444	0.537	0.486	2090
micro avg	0.438	0.438	0.438	6950
macro avg	0.414	0.397	0.399	6950
weighted avg	0.434	0.438	0.432	6950
samples avg	0.438	0.438	0.438	6950

Table 5.7: Results of VGG-16 run for vectorscope representations, for 4 epochs

The confusion matrix (refer to Table 5.8) for vectorscope representations in the "separate" distribution experiments reveals the least favorable results compared to the other experiments, as initially anticipated.

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	140	214	167	229
Act_Adv	121	1095	483	804
Com_Rom	77	470	688	372
Thr_Hor_Cri	86	522	360	1122

Table 5.8: Confusion matrix of VGG-16 run for vectorscope representations, for 4 epochs

5.5 Binary Classification

With consistently inadequate results, I opted to explore the last possible simplification to the problem: binary classification. I chose to narrow down the dataset to the two largest classes, Thr_Hor_Cri and Act_Adv [refer to Figures 5.5, 5.6, 5.7 and 5.8].

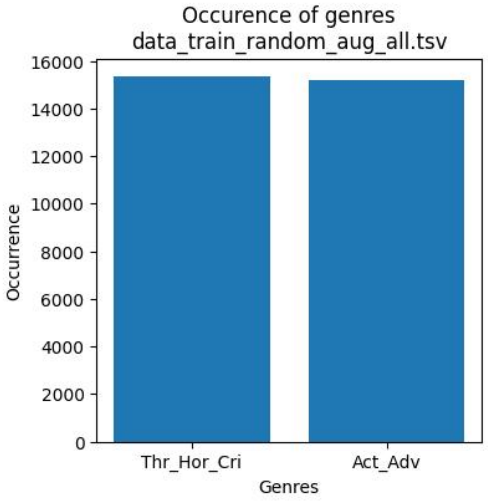

Figure 5.5: Data distribution of the training dataset for random frame distribution

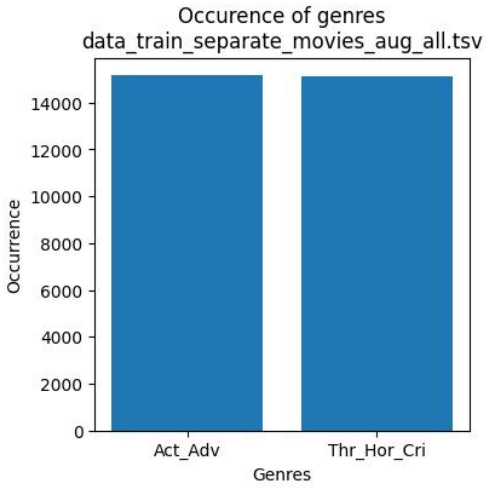

Figure 5.6: Data distribution of the training dataset for "separate" frame distribution

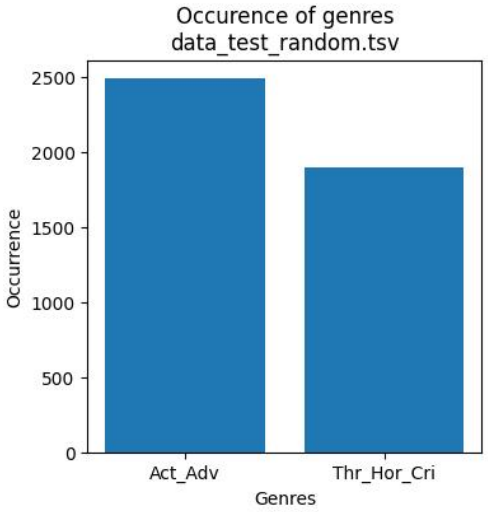

Figure 5.7: Data distribution of the testing dataset for random frame distribution

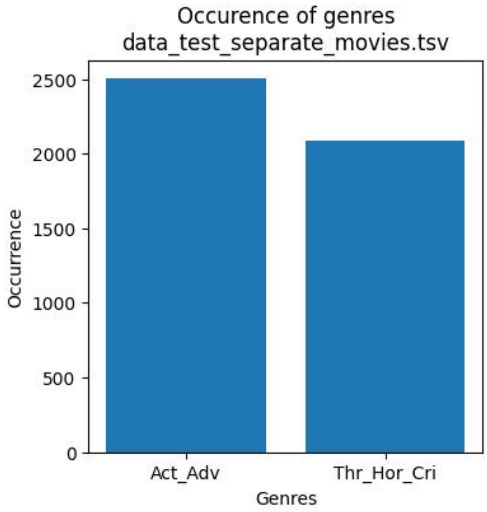

Figure 5.8: Data distribution of the testing dataset for "separate" frame distribution

I decided to follow the path identified earlier - the raw frames were processed through the ResNet-50 model, and vectorscope representations were handled by the VGG-16 model. The results of the initial experiment yielded an accuracy of 76.87%, indicating a promising aspect of the experiments. However, for binary classification, the result remains low. The detailed classification report is presented

in Table 5.9 and the confusion matrix in Table 5.10.

Genre	Precision	Recall	F1-score	Support
act_adv	0.811	0.773	0.791	2486
thr_hor_cri	0.719	0.763	0.741	1894
micro avg	0.769	0.769	0.769	4380
macro avg	0.765	0.768	0.766	4380
weighted avg	0.771	0.769	0.769	4380
samples avg	0.769	0.769	0.769	4380

Table 5.9: Results of ResNet-50 run for raw frames, for 5 epochs

Genre	Act_Adv	Thr_Hor_Cri
Act_Adv	1921	565
Thr_Hor_Cri	448	1446

Table 5.10: Confusion matrix of ResNet-50 run for raw frames, for 5 epochs

With vectorscopes, the situation is more unfavorable than anticipated, resulting in an accuracy of 63.49% on the VGG-16 model. The detailed results are presented in Tables 5.11 and 5.12.

Genre	Precision	Recall	F1-score	Support
act_adv	0.811	0.773	0.791	2486
thr_hor_cri	0.719	0.763	0.741	1894
micro avg	0.769	0.769	0.769	4380
macro avg	0.765	0.768	0.766	4380
weighted avg	0.771	0.769	0.769	4380
samples avg	0.769	0.769	0.769	4380

Table 5.11: Results of VGG-16 run for vec frames, for 5 epochs

Genre	Act_Adv	Thr_Hor_Cri
Act_Adv	1921	565
Thr_Hor_Cri	448	1446

Table 5.12: Confusion matrix of VGG-16 run for vec frames, for 5 epochs

Additional experiments concentrated on a "separate" frames distribution. Similar to prior experiments, the results are diminished. The accuracy for raw frames tested on ResNet-50 is 70.22%, indicating a correlation between raw frames and movie genres. Detailed results are provided in the tables 5.13 and 5.14.

Genre	Precision	Recall	F1-score	Support
act_adv	0.744	0.691	0.717	2503
thr_hor_cri	0.719	0.763	0.741	2090
micro avg	0.702	0.702	0.702	4593
macro avg	0.702	0.703	0.701	4593
weighted avg	0.705	0.702	0.703	4593
samples avg	0.702	0.702	0.702	4593

Table 5.13: Results of ResNet-50 run for raw frames, for 5 epochs

Genre	Act_Adv	Thr_Hor_Cri
Act_Adv	1730	773
Thr_Hor_Cri	595	1495

Table 5.14: Confusion matrix of ResNet-50 run for raw frames, for 5 epochs

Just as in the case of experiments conducted on four classes, the decline in dataset performance is less pronounced when using vectorscope representations compared to raw frames. For vectorscope representations processed with VGG-16 in the 'separate' distribution, the accuracy was 60.09% [refer to Tables 5.15 and 5.16].

Genre	Precision	Recall	F1-score	Support
act_adv	0.649	0.583	0.614	2503
thr_hor_cri	0.555	0.622	0.587	2090
micro avg	0.601	0.601	0.601	4593
macro avg	0.602	0.603	0.600	4593
weighted avg	0.606	0.601	0.602	4593
samples avg	0.601	0.601	0.601	4593

Table 5.15: Results of VGG-16 run for vec frames, for 5 epochs

Genre	Act_Adv	Thr_Hor_Cri
Act_Adv	1460	1043
Thr_Hor_Cri	790	1300

Table 5.16: Confusion matrix of VGG-16 run for vec frames, for 5 epochs

6. Conclusions

In general, the overall outcomes fall short of expectations. However, a noteworthy observation directs to the accuracy exhibited in experiments utilizing vectorscope representations. Not only did these experiments demonstrate a nearly double the increase in speed, but their accuracy also displayed consistency across diverse experimental conditions. This included successful performance on both simpler and more challenging datasets, such as those featuring a random distribution of data versus datasets where movies were separated between training and testing sets.

One notable observation lies in the distinct performance of each model, with ResNet-50 emerging as the most proficient choice for raw frames, achieving an accuracy of 60.54%. Conversely, VGG-16 demonstrated its superiority in handling vectorscope representations, attaining an accuracy of 45.73%. It's worth highlighting that although VGG-16 excelled in accuracy, it concurrently exhibited an extended training duration compared to alternative models, rendering it the slower option. For a comprehensive overview of the average performance of all models and their corresponding epoch times, refer to Figure 6.1. Notably, ResNet-50 surpasses all models in terms of accuracy while maintaining the second-best computational efficiency, whereas VGG-16 stands out as the slowest option.

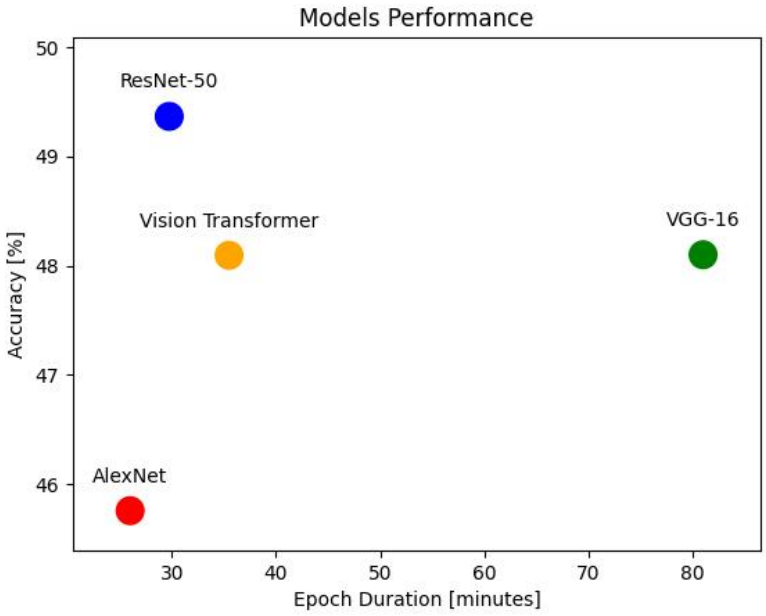

Figure 6.1: Model accuracies in relation to the duration of each epoch

Another crucial aspect to consider in interpreting results is the inherent difficulty of the task. It's not always feasible to accurately classify the genre of a movie based solely on a movie frame, let alone the extracted color data. Examples of misclassification are illustrated below.

The initial example [Figure 6.2] depicts an image from a movie belonging to the class Act_Adv that was inaccurately classified as Com_Rom. The misclassification in this instance can be rationalized. The scene contains numerous lively and vibrant colors, implying a sense of happiness. Additionally, there is a person wearing a costume and a smiling girl in the background.

Figure 6.2: Example of Act_Adv frame classified as Com_Rom

The second example [Figure 6.3] illustrates a typical misclassification where Act_Adv is wrongly classified as Thr_Hor_Cri. This misclassification is justifiable when considering the predominantly dark colors in the frame. Additionally, the person in the frame appears nervous and has a bloody scar on their forehead.

Figure 6.3: Example of Act_Adv frame classified as Thr_Hor_Cri

The next example [Figure 6.4] depicts Com_Rom misclassified as Drama, representing another instance of a common mistake, as many movies can be broadly categorized as drama. In this example, there are no close-ups in the frame, people are not displaying emotions, and the colors are dull with nothing particularly standing out.

Figure 6.4: Example of Act_Adv frame classified as Thr_Hor_Cri

The final example [Figure 6.5] is noteworthy as it represents an uncommon situation where Thr_Hor_Cri is misclassified as Com_Rom. In this case, the frame depicts an elegant woman with a slight smile, red lipstick, and jewelry, suggesting the possibility of the movie being a romance.

Figure 6.5: Example of Act_Adv frame classified as Thr_Hor_Cri

Another facet open for potential enhancement revolves around the dataset itself. The dataset employed was generated automatically, incorporating basic filtering and limited to a small collection of clips. Despite the current constraints in accuracy measures, the results exhibit promise for future research, particularly when considering the prospect of employing a larger and more comprehensive dataset.

A. Data Preprocessing

A.1 Filtering the Dataset

The provided Python code snippet serves the purpose of verifying the availability of a video.

```
    def video_available(link):
        # checking if video is available
        url = 'https://www.youtube.com/watch?v=' + link

        try:
            pafy.new(url)
            return True
        except OSError:
            return False
        except:
            return False
```

Listing A.1: Python `video_available()` function

The provided Python code snippet is used to determine whether a YouTube video is in grayscale or not.

```
def color_check(link):
    url = 'https://www.youtube.com/watch?v=' + link

    try:
        vPafy = pafy.new(url)
        play = vPafy.getbestvideo(preftype='webm')

        video = cv2.VideoCapture(play.url)

        fps = int(video.get(cv2.CAP_PROP_FPS))

        # vid_len = vPafy.length

        # we want to extract one frame from the 2 second of the video
```

```
        frame_num = int(fps * 2)

        # print('frame to extract:', frame_num)

        current_frame = 0

        while (True):
            # read frame
            ret, frame = video.read()

            if ret:
                if current_frame == frame_num:
                    # cv2.imshow('frame', frame)

                    # splitting b, g, r channels
                    b, g, r = cv2.split(frame)

                    # getting differences between (b,g), (r,g), (b,r)
                    # channel pixels
                    r_g = np.count_nonzero(abs(r-g))
                    r_b = np.count_nonzero(abs(r-b))
                    g_b = np.count_nonzero(abs(g-b))

                    # sum of differences
                    diff_sum = float(r_g+r_b+g_b)

                    # finding ratio of diff_sum with respect to size
                    # of image
                    ratio = diff_sum/frame.size

                    if ratio > 0.005:
                        # print("image is color")
                        ret_val = True
                    else:
                        # print("image is greyscale")
                        ret_val = False

                    break
                    # current_frame += 1
                else:
                    current_frame += 1
            else:
                ret_val = False
                break


        # release VideoCapture
        video.release()

```

```
64          cv2.destroyAllWindows()
65
66          return ret_val
67
68      except:
69          return False
```

Listing A.2: Python `color_check()` function

A.2 Limiting the Dataset

Distribution of all available movie genre combinations:

- Drama: 5931,
- Action, Adventure: 5319,
- Comedy: 4052,
- Adventure, Comedy: 3979,
- Comedy, Drama: 3090,
- Crime, Drama: 2329,
- Action, Drama: 1746,
- Comedy, Romance: 1740,
- Drama, Romance: 1683,
- Adventure, Drama: 1478,
- Horror, Thriller: 1361,
- Horror: 1327,
- Adventure: 1276,
- Action: 1248,
- Action, Thriller: 1204,
- Action, Comedy: 1160,
- Comedy, Horror: 1072,
- Action, Crime: 919,
- Comedy, Crime: 902,
- Action, Horror: 859,
- Drama, Horror: 793,
- Drama, Thriller: 679,
- Thriller: 455,
- Romance: 203,
- Romance, Thriller: 168,
- Crime: 156,
- Adventure, Horror: 154,
- Comedy, Thriller: 131,
- Crime, Horror: 109,
- Crime, Thriller: 80,
- Horror, Romance: 72,
- Action, Romance: 71,
- Crime, Romance: 56,
- Adventure, Thriller: 43.

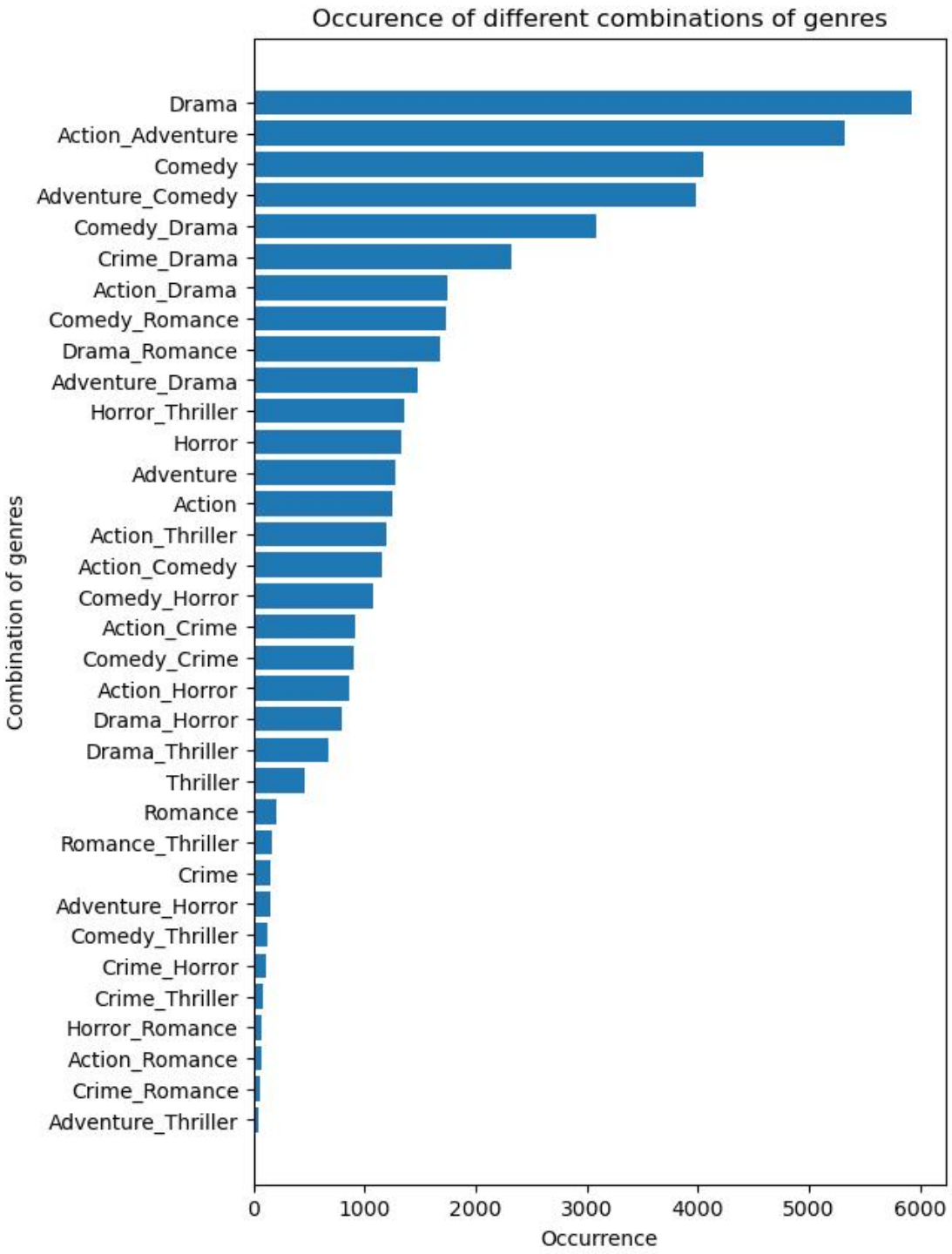

Figure A.1: Number of occurrences of combinations of genres

A.3 Collecting the Data

The provided Python code snippet serves the purpose of extracting frames from a youtube video.

```
def extract_frame(link, genre, start_frame, interval, folder_path):

    genre_str = genre.replace("'", "").replace(
        "[", "").replace("]", "").replace(" ", "").replace(",", "_")

    if os.path.exists(folder_path + "/" + genre_str + "/" + link):
        print(folder_path + "/" + genre_str + "/" + link + " exists")
        return
    else:
        create_dir(folder_path + "/" + genre_str + "/" + link)
        print("Creating folder..." + folder_path + "/" + genre_str)

        url = 'https://www.youtube.com/watch?v=' + link

        vPafy = pafy.new(url)
        play = vPafy.getbestvideo(preftype='webm')

        video = cv2.VideoCapture(play.url)

        fps = int(video.get(cv2.CAP_PROP_FPS))

        vid_len = vPafy.length

        # we want to get the first frame at start_frame seconds
        # and then each frame after the next interval seconds
        start_value = start_frame * fps
        interval_fps = interval * fps
        # we stop at the 3/4 of the video because of the fact
        # that videos usually contain some ending credits
        stop_value = int(3/4 * fps * vid_len)

        frame_numbers = [start_value]

        while frame_numbers[-1] + interval_fps < stop_value:
            frame_numbers.append(frame_numbers[-1] + interval_fps)

        current_frame = 0

        while (True):
            # read frame
            ret, frame = video.read()

            if ret:
                if current_frame in frame_numbers:
```

```
                    name = folder_path + "/" + genre_str +\
                        "/" + link + "/" + str(current_frame) + '.jpg
    '

                    if os.path.isfile(name):
                        print(name + "exists")
                    else:
                        print("Creating..." + name)
                        cv2.imwrite(name, frame)

                    # shutil.move(name, dest)
                    current_frame += 1
                else:
                    current_frame += 1
            else:
                break

            if current_frame > frame_numbers[-1]:
                break

        # release VideoCapture
        video.release()

        cv2.destroyAllWindows()
```

Listing A.3: Python `extract_frame()` function

The provided Python code snippet serves the purpose of creating the data.tsv file.

```
# Loop over subfolders in the parent folder
for subfolder in os.listdir(parent_folder):

    subfolder_path = os.path.join(parent_folder, subfolder)

    if os.path.isdir(subfolder_path):

        # Loop over sub-subfolders in the subfolder
        for subsubfolder in os.listdir(subfolder_path):
            subsubfolder_path = os.path.join(subfolder_path,
    subsubfolder)

            for frame in os.listdir(subsubfolder_path):

                frame_path = os.path.join(subsubfolder_path, frame)

                if os.path.exists(frame_path):
                    current_genres = subfolder.split('_')
                    binary_genre_list = []

```

```
                    for genre in genre_list:
                        if genre in current_genres:
                            binary_genre_list.append(1)
                        else:
                            binary_genre_list.append(0)

                    relative_frame_path = frame_path[3:]

                    row_list = [relative_frame_path, current_genres]
    + binary_genre_list

                    row = pd.DataFrame([row_list], columns=header)

                    data_df = data_df.append(row.iloc[0],
    ignore_index=True)

data_df.to_csv(r'..\files\data.tsv', sep="\t")
```

Listing A.4: Python code for creating the data.tsv file

The provided Python code is used to show the distribution of training and testing subsets of the data fed into the CNN.

```
def get_genre_distribution(df) -> dict:

    existing_genres = {}

    # Loop over subfolders in the parent folder
    for index, row in df.iterrows():

        genre = '_'.join(ast.literal_eval(row['genre']))

        if genre not in existing_genres:
            existing_genres[genre] = 1
        else:
            existing_genres[genre] += 1


    existing_genres = dict(sorted(existing_genres.items(), key=lambda
    x:x[1], reverse=True))


    single_genre_list = []

    for gen in existing_genres.keys():

        gen_list = gen.split('_')

        for g in gen_list:

```

```
27              if g not in single_genre_list:
28                  single_genre_list.append(g)
29
30
31      existing_single_genres = {}
32
33      for gen in single_genre_list:
34
35          if gen not in existing_single_genres:
36              existing_single_genres[gen] = 0
37
38          for g in existing_genres:
39
40              if gen in g:
41                  existing_single_genres[gen] += existing_genres[g]
42
43      existing_single_genres = dict(sorted(existing_single_genres.items
        (), key=lambda x:x[1], reverse=True))
44
45      plt.figure(figsize = (10, 5))
46
47      plt.bar(existing_single_genres.keys(), existing_single_genres.
        values())
48
49      plt.xlabel("Genres")
50      plt.ylabel("Occurrence")
51      plt.title("Occurence of genres")
52
53      return existing_single_genres
```

Listing A.5: Python `distribution_of_data()` function

The provided Python code is used to normalize the values inside of the dictionary.

```
def normalize_dict(data_dict):
    norm_dict = {}

    for key in data_dict:
        norm_value = float(data_dict[key])/max(data_dict.values())
        norm_dict[key] = norm_value

    return norm_dict
```

Listing A.6: Python `normalize_dict()` function

The provided Python code snippet serves the purpose of dividing the dataset on training and testing split in a random manner.

```
df = df.sample(frac=1).reset_index(drop=True)
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

df_train_random, df_test_random = train_test_split(df, test_size
    =0.15)

df_train_random = df_train_random.reset_index(drop=True)
df_test_random = df_test_random.reset_index(drop=True)
```

Listing A.7: Python code for splitting the dataset randomly

The provided Python code is used to divide the dataset in a manner which doesn't allow frames of one movie to be in both training and testing dataset.

```
# Add a column with the path of the movie folder but without the .jpg
    files

df['path_no_file'] = df.apply(lambda row: row['path'][:row['path'].
    rfind("\\")], axis=1)

df = df[[df.columns[-1]] + df.columns[:-1].tolist()]

# Get the pandas series with separate movie paths to later merge them
    with the previous table

df_path_no_file = df['path_no_file'].drop_duplicates()

# Split the pandas series

df_path_no_file = df_path_no_file.sample(frac=1).reset_index(drop=
    True)

```

```
df_path_no_file_train_separate_movies,
    df_path_no_file_test_separate_movies = train_test_split(
    df_path_no_file, test_size=0.15)

df_path_no_file_train_separate_movies =
    df_path_no_file_train_separate_movies.reset_index(drop=True)
df_path_no_file_test_separate_movies =
    df_path_no_file_test_separate_movies.reset_index(drop=True)

# Merge tables and check the distribution of a training set

df_train_separate_movies = pd.merge(df,
    df_path_no_file_train_separate_movies, on='path_no_file', how='
    inner')

df_train_separate_movies = df_train_separate_movies[
    df_train_separate_movies.columns.tolist()[1:]]

get_genre_distribution(df_train_separate_movies)

# Merge tables and check the distribution of a test set

df_test_separate_movies = pd.merge(df,
    df_path_no_file_test_separate_movies, on='path_no_file', how='
    inner')

df_test_separate_movies = df_test_separate_movies[
    df_test_separate_movies.columns.tolist()[1:]]

get_genre_distribution(df_test_separate_movies)
```

Listing A.8: Python code for splitting the dataset for the second experiment

The provided Python code snippet presents a part of `vectorscope.py`.

```
src = Image.open(raw_image)
src = src.resize((320, 180))
src = asarray(src)

if src.dtype == np.uint16:
    src = (src / 2**8).astype(np.uint8)

R, G, B = src[:, :, 0], src[:, :, 1], src[:, :, 2]

Y = (0.299 * R) + (0.587 * G) + (0.114 * B)
Cb = (-0.169 * R) - (0.331 * G) + (0.499 * B) + 128
Cr = (0.499 * R) - (0.418 * G) - (0.0813 * B) + 128

# traditional vectorscope orientation
Cr = 256 - Cr
```

```
16
17 dst = np.zeros((256, 256, 3), dtype=src.dtype)
18
19 for x in range(src.shape[0]):
20     for y in range(src.shape[1]):
21         dst[int(Cr[x, y]), int(Cb[x, y])] = np.array([R[x, y], G[x, y
   ], B[x, y]])
22         # print(len(dst))
23
24 cv2.imwrite(vec_image, dst)
```

Listing A.9: Python `vectorscope.py` code snippet

The provided Python code shows how the data augmentation parameters for the transformations were chosen.

```
1  def generate_random_parameters(im_width, im_height):
2      width = random.randint(im_width / 2, im_width)
3      height = random.randint(im_height / 2, im_height)
4
5      brightness = random.uniform(0.2, 0.8)
6      contrast = random.uniform(0.2, 0.8)
7      saturation = random.uniform(0.2, 0.8)
8      hue = random.uniform(0.1, 0.2)
9
10     blur_kernel_size = random.choice(range(3, 100, 2))
11
12     return (height, width, brightness, contrast, saturation, hue,
       blur_kernel_size)
```

Listing A.10: Python function returning random parameters for image dransformation used in data augmentation

The provided Python function generates the modified image in data augmentation.

```
1  def generate_modified_image(image_path):
2      # Load the image
3      image_path = image_path.replace("\\", "/")
4      image = Image.open(image_path)
5      width, height = image.size
6
7      (
8          crop_height,
9          crop_width,
10         brightness,
11         contrast,
12         saturation,
13         hue,
14         blur_kernel_size,
```

```
    ) = generate_random_parameters(width, height)

    # Define transformations to apply to the image
    transform = transforms.Compose(
        [
            transforms.RandomCrop((crop_height, crop_width)),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.ColorJitter(
                brightness=brightness, contrast=contrast, saturation=
saturation, hue=hue
            ),
            transforms.GaussianBlur(kernel_size=blur_kernel_size),
        ]
    )

    # Transform the image
    transformed_image = transform(image)

    # # Display the transformed image
    # plt.imshow(transformed_image)
    # plt.axis("off")
    # plt.show()

    # Extract directory path
    directory_path = os.path.dirname(image_path)

    # Create augmented directory path
    augmented_directory_path = directory_path.replace("_frames", "
_frames_aug")
    create_dir(augmented_directory_path.replace("\\", "/"))

    # Create augmented image file path
    augmented_image_path = image_path.replace("_frames", "_frames_aug
").replace(
        ".jpg", "_aug.jpg"
    )

    transformed_image.save(augmented_image_path.replace("\\", "/"))

    return augmented_image_path.replace("raw_frames_aug/", "").
replace("\\", "/")
```

Listing A.11: Python `generate_modified_image()` function

A.4 Loading the Data

The provided Python code snippet presents the `MovieFramesDataset` class.

```
class MovieFrameDataset(Dataset):

    def __init__(self, root_dir, csv, transform=None, raw=False):

        super().__init__()
        self.root_dir = root_dir
        self.csv = csv
        self.transform = transform
        self.raw = raw

        df = pd.read_csv(csv, sep="\t")

        self.elements = [root_dir + p for p in df['path'].to_list()]
        self.labels = [row.to_list() for _, row in df.iloc[:, -8:].
    iterrows()]
        self.length = len(df)

        self.labels_order = df.iloc[:, -8:].columns.tolist()

    def __len__(self):

        return self.length

    def __getitem__(self, index):

        img = Image.open(self.elements[index].rstrip())

        print(self.elements[index])

        target = self.labels[index]

        if self.raw:
            # two following steps of transform are not enclosed in
    transform
            # parameter as cropping in that way is not available in
    transforms
            # library
            # they are only used for raw images
            img = transforms.Resize((700, 1000))(img)
            img = TF.crop(img, 90, 0, 520, 1000)

        if self.transform is not None:
            img = self.transform(img)


```

```
43          image, label = img, target
44
45          return torch.tensor(image, dtype=torch.float32), torch.tensor
    (label, dtype=torch.float32)
```

Listing A.12: Python `MovieFramesDataset` class

The provided Python code shows how the frames were transformed inside of the `MovieFramesDataset` class.

```
# Transforming the data
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    # transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])
```

Listing A.13: Python code for transforming the raw frames

The provided Python code snippet illustrates the utilization of the `Dataset` and `DataLoader` classes.

```
# Creating the dataset and data loaders
train_frames_type, test_frames_type, train_list, test_list =
    choose_data_params(
    opt["frames"], opt["augmentation"], opt["data_distribution"]
)

train_set_all = MovieFrameDataset(
    "{}/".format(train_frames_type),
    "files/{}.tsv".format(train_list),
    transform=transform,
    raw=True,
)
test_set_all = MovieFrameDataset(
    "{}/".format(test_frames_type),
    "files/{}.tsv".format(test_list),
    transform=transform,
    raw=True,
)

train_set = torch.utils.data.Subset(
    train_set_all, list(range(0, int(0.4 * len(train_set_all))))
)
test_set = torch.utils.data.Subset(
    test_set_all, list(range(0, int(0.4 * len(test_set_all))))
)

```

```
26 if not opt["trial"]:
27     train_set = train_set_all
28     test_set = test_set_all
29
30 print("train_set:", len(train_set))
31 print("test_set:", len(test_set))
32
33 train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=
       True)
34 test_loader = DataLoader(test_set, batch_size=BATCH_SIZE, shuffle=
       False)
```

Listing A.14: The utilization of the `Dataset` and `DataLoader` classes

B. Experiments and Results

B.1 Loss and Optimization

The provided Python code snippet illustrates the implementation of loss function, optimizer and the step-down policy.

```
# Define loss function
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(
    model.parameters(), lr=LR, momentum=MOMENTUM, weight_decay=
    WEIGHT_DECAY
)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=STEP_SIZE,
    gamma=GAMMA)
```

Listing B.1: The implementation of loss, optimizer and step-down functions

B.2 Results

B.2.1 Raw Frames - Random Distribution

AlexNet

Additional figures and tables for AlexNet experiments in 5.1 section.

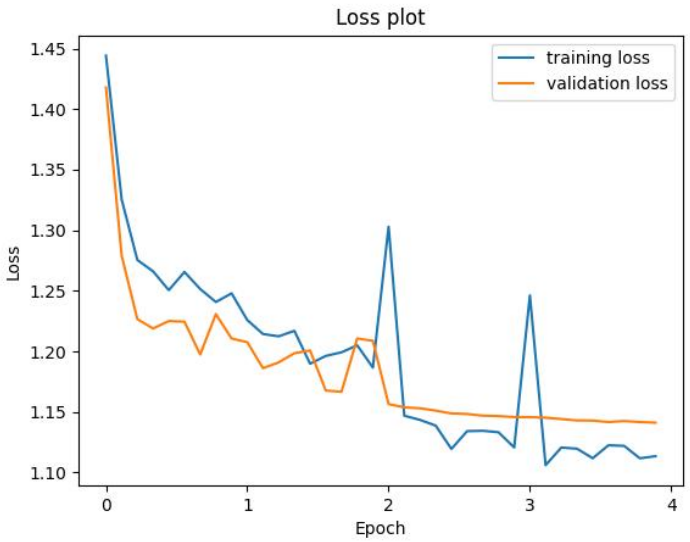

Figure B.1: Loss plot of AlexNet run for raw frames, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.350	0.202	0.256	893
Act_Adv	0.584	0.580	0.582	2486
Com_Rom	0.512	0.480	0.495	1604
Thr_Hor_Cri	0.452	0.571	0.505	1894
micro avg	0.505	0.505	0.505	6877
macro avg	0.475	0.458	0.459	6877
weighted avg	0.500	0.505	0.498	6877
samples avg	0.505	0.505	0.505	6877

Table B.1: Results of AlexNet run for raw frames, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	180	234	165	314
Act_Adv	124	1441	297	624
Com_Rom	94	368	770	372
Thr_Hor_Cri	116	425	272	1081

Table B.2: Confusion matrix of AlexNet run for raw frames, for 4 epochs

VGG-16

Additional figures and tables for VGG-16 experiments in 5.1 section.

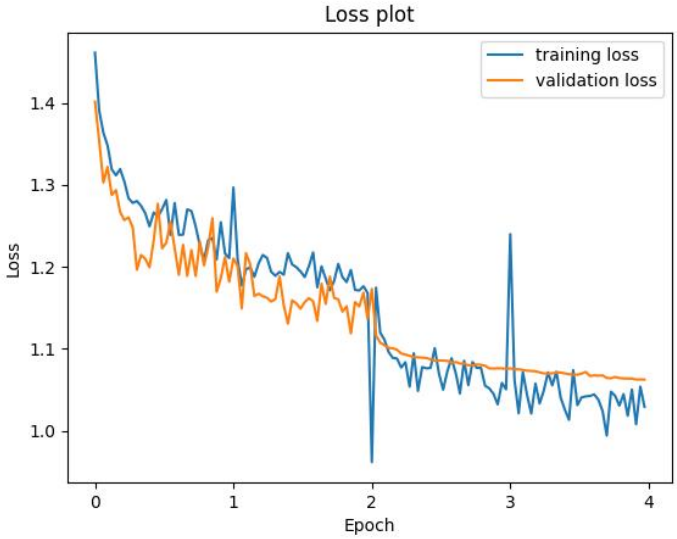

Figure B.2: Loss plot of VGG-16 run for raw frames, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.442	0.267	0.332	893
Act_Adv	0.632	0.631	0.632	2486
Com_Rom	0.512	0.531	0.522	1604
Thr_Hor_Cri	0.484	0.560	0.519	1894
micro avg	0.541	0.541	0.541	6877
macro avg	0.517	0.497	0.501	6877
weighted avg	0.538	0.541	0.536	6877
samples avg	0.541	0.541	0.541	6877

Table B.3: Results of VGG-16 run for raw frames, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	238	170	214	271
Act_Adv	113	1569	266	538
Com_Rom	89	340	852	323
Thr_Hor_Cri	99	404	331	1060

Table B.4: Confusion matrix of VGG-16 run for raw frames, for 4 epochs

ViT

Additional figures and tables for ViT experiments in 5.1 section.

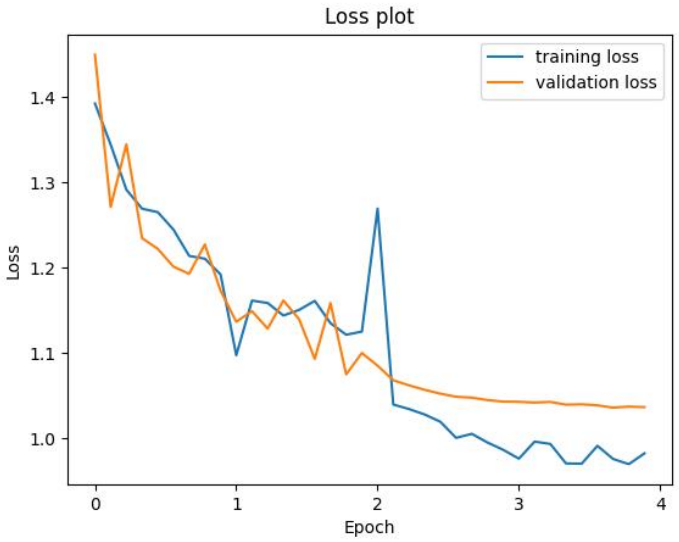

Figure B.3: Loss plot of ViT run for raw frames, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.466	0.302	0.367	893
Act_Adv	0.647	0.649	0.648	2486
Com_Rom	0.532	0.557	0.544	1604
Thr_Hor_Cri	0.508	0.570	0.537	1894
micro avg	0.561	0.561	0.561	6877
macro avg	0.538	0.520	0.524	6877
weighted avg	0.558	0.561	0.557	6877
samples avg	0.561	0.561	0.561	6877

Table B.5: Results of ViT run for raw frames, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	270	178	183	262
Act_Adv	106	1614	285	481
Com_Rom	97	310	893	304
Thr_Hor_Cri	107	392	316	1079

Table B.6: Confusion matrix of ViT run for raw frames, for 4 epochs

B.2.2 Vectorscope Representations - Random Distribution

AlexNet

Additional figures and tables for AlexNet experiments in 5.2 section.

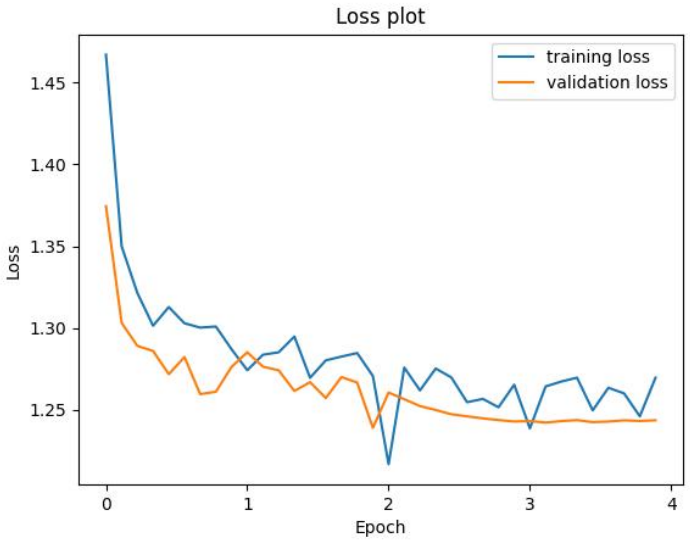

Figure B.4: Loss plot of AlexNet run for vectorscope representations, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.317	0.071	0.115	893
Act_Adv	0.513	0.464	0.487	2486
Com_Rom	0.407	0.489	0.444	1604
Thr_Hor_Cri	0.385	0.509	0.439	1894
micro avg	0.431	0.431	0.431	6877
macro avg	0.405	0.383	0.371	6877
weighted avg	0.428	0.431	0.416	6877
samples avg	0.431	0.431	0.431	6877

Table B.7: Results of AlexNet run for vectorscope representations, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	63	242	235	353
Act_Adv	29	1153	502	802
Com_Rom	40	397	785	382
Thr_Hor_Cri	67	455	408	964

Table B.8: Confusion matrix of AlexNet run for vectorscope representations, for 4 epochs

ResNet-50

Additional figures and tables for ResNet-50 experiments in 5.2 section.

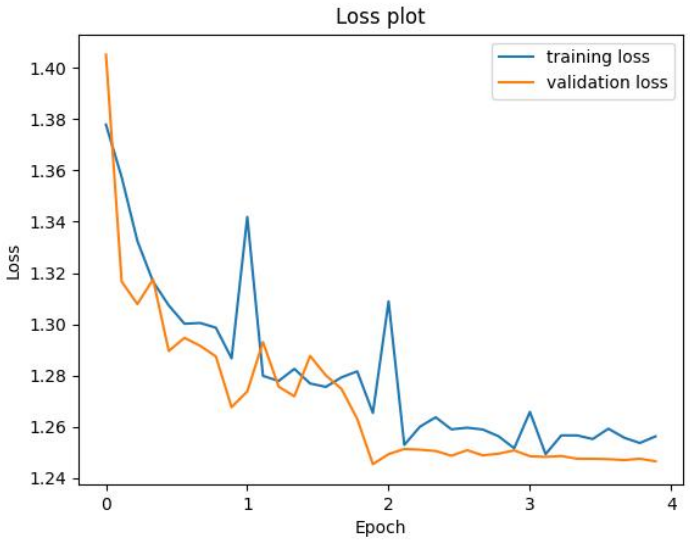

Figure B.5: Loss plot of ResNet-50 run for vectorscope representations, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.341	0.101	0.156	893
Act_Adv	0.482	0.488	0.485	2486
Com_Rom	0.403	0.440	0.421	1604
Thr_Hor_Cri	0.396	0.490	0.438	1894
micro avg	0.427	0.427	0.427	6877
macro avg	0.405	0.380	0.375	6877
weighted avg	0.421	0.427	0.414	6877
samples avg	0.427	0.427	0.427	6877

Table B.9: Results of ResNet-50 run for vectorscope representations, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	90	289	223	291
Act_Adv	58	1212	444	772
Com_Rom	41	502	706	355
Thr_Hor_Cri	75	511	380	928

Table B.10: Confusion matrix of ResNet-50 run for vectorscope representations, for 4 epochs

ViT

Additional figures and tables for ViT experiments in 5.2 section.

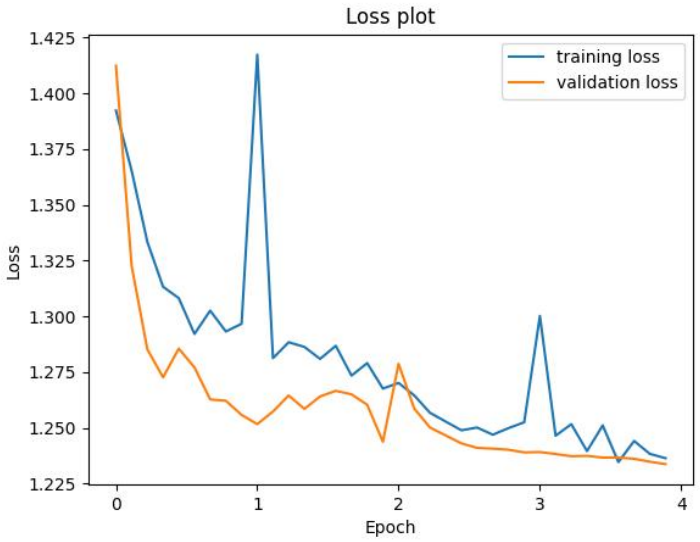

Figure B.6: Loss plot of ViT run for vectorscope representations, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.358	0.097	0.153	893
Act_Adv	0.489	0.512	0.500	2486
Com_Rom	0.419	0.461	0.439	1604
Thr_Hor_Cri	0.406	0.486	0.442	1894
micro avg	0.439	0.439	0.439	6877
macro avg	0.418	0.389	0.384	6877
weighted avg	0.433	0.439	0.425	6877
samples avg	0.439	0.439	0.439	6877

Table B.11: Results of ViT run for vectorscope representations, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	87	285	229	292
Act_Adv	43	1273	457	713
Com_Rom	40	483	739	342
Thr_Hor_Cri	73	562	339	920

Table B.12: Confusion matrix of ViT run for vectorscope representations, for 4 epochs

B.2.3 Raw Frames - "Separate" Distribution

AlexNet

Additional figures and tables for AlexNet experiments in 5.3 section.

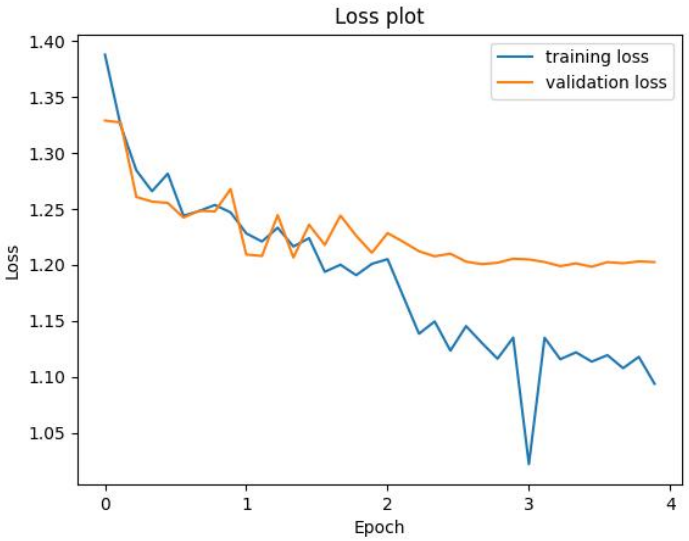

Figure B.7: Loss plot of AlexNet run for raw frames, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.302	0.207	0.245	750
Act_Adv	0.534	0.523	0.529	2503
Com_Rom	0.455	0.444	0.449	1607
Thr_Hor_Cri	0.446	0.516	0.479	2090
micro avg	0.469	0.469	0.469	6950
macro avg	0.434	0.422	0.426	6950
weighted avg	0.464	0.469	0.465	6950
samples avg	0.469	0.469	0.469	6950

Table B.13: Results of AlexNet run for raw frames, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	155	220	139	236
Act_Adv	134	1310	342	717
Com_Rom	112	395	713	387
Thr_Hor_Cri	113	526	372	1079

Table B.14: Confusion matrix of AlexNet run for raw frames, for 4 epochs

VGG-16

Additional figures and tables for VGG-16 experiments in 5.3 section.

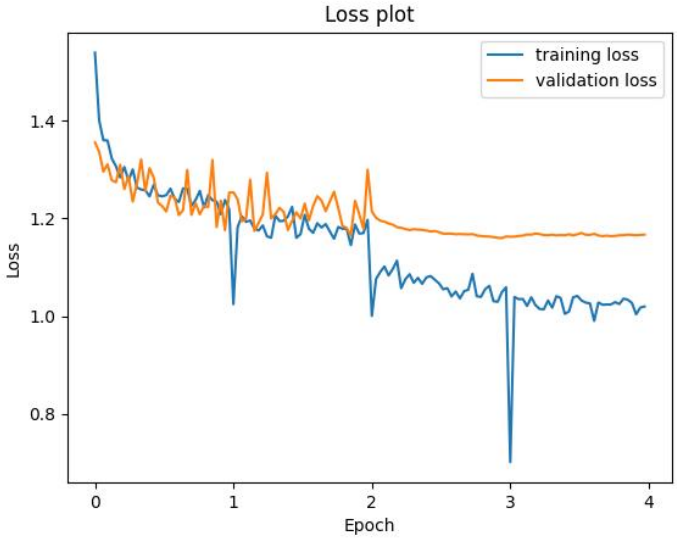

Figure B.8: Loss plot of VGG-16 run for raw frames, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.314	0.224	0.261	750
Act_Adv	0.586	0.545	0.565	2503
Com_Rom	0.464	0.478	0.471	1607
Thr_Hor_Cri	0.448	0.521	0.482	2090
micro avg	0.488	0.488	0.488	6950
macro avg	0.453	0.442	0.445	6950
weighted avg	0.487	0.488	0.485	6950
samples avg	0.488	0.488	0.488	6950

Table B.15: Results of VGG-16 run for raw frames, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	168	168	163	251
Act_Adv	117	1365	307	714
Com_Rom	118	346	768	375
Thr_Hor_Cri	132	451	418	1089

Table B.16: Confusion matrix of VGG-16 run for raw frames, for 4 epochs

ViT

Additional figures and tables for ViT experiments in 5.3 section.

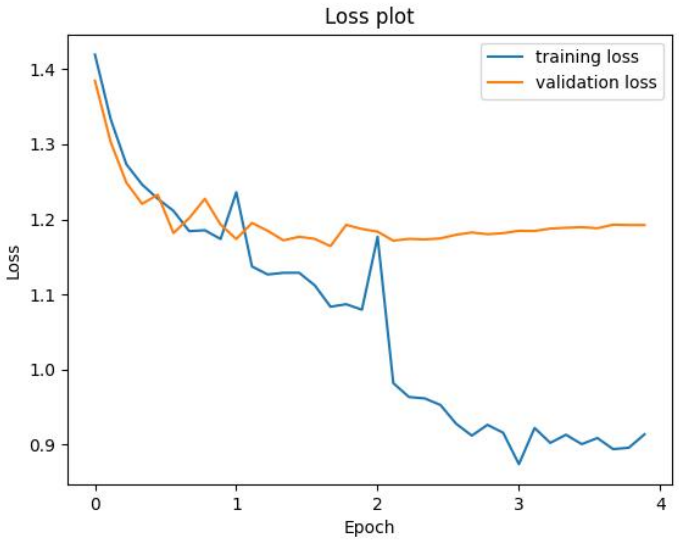

Figure B.9: Loss plot of ViT run for raw frames, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.340	0.308	0.323	750
Act_Adv	0.572	0.587	0.579	2503
Com_Rom	0.494	0.435	0.462	1607
Thr_Hor_Cri	0.478	0.523	0.500	2090
micro avg	0.502	0.502	0.502	6950
macro avg	0.471	0.463	0.466	6950
weighted avg	0.501	0.502	0.501	6950
samples avg	0.502	0.502	0.502	6950

Table B.17: Results of ViT run for raw frames, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	231	173	124	222
Act_Adv	171	1469	270	593
Com_Rom	142	389	699	377
Thr_Hor_Cri	136	538	323	1093

Table B.18: Confusion matrix of ViT run for raw frames, for 4 epochs

B.2.4 Vectorscope Representations - "Separate" Distribution

AlexNet

Additional figures and tables for AlexNet experiments in 5.4 section.

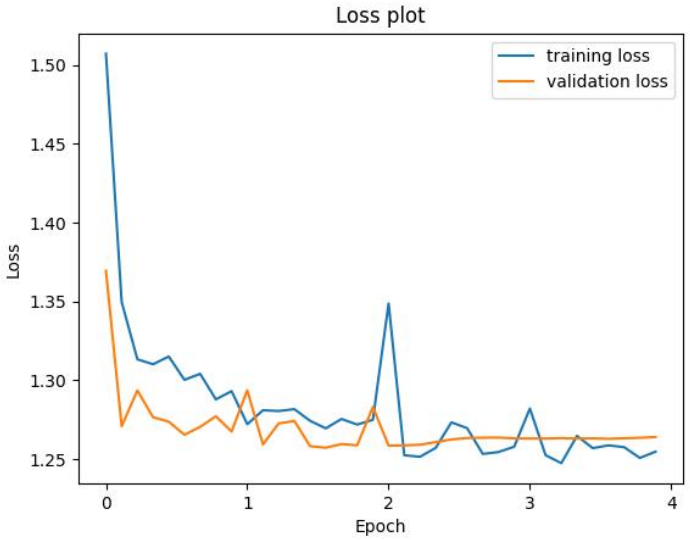

Figure B.10: Loss plot of AlexNet run for vectorscope representations, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.474	0.144	0.221	750
Act_Adv	0.464	0.409	0.435	2503
Com_Rom	0.385	0.479	0.426	1607
Thr_Hor_Cri	0.420	0.507	0.460	2090
micro avg	0.426	0.426	0.426	6950
macro avg	0.436	0.384	0.385	6950
weighted avg	0.434	0.426	0.417	6950
samples avg	0.426	0.426	0.426	6950

Table B.19: Results of AlexNet run for vectorscope representations, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	108	208	205	229
Act_Adv	66	1023	582	832
Com_Rom	28	411	769	399
Thr_Hor_Cri	26	561	444	1059

Table B.20: Confusion matrix of AlexNet run for vectorscope representations, for 4 epochs

ResNet-50

Additional figures and tables for ResNet-50 experiments in 5.4 section.

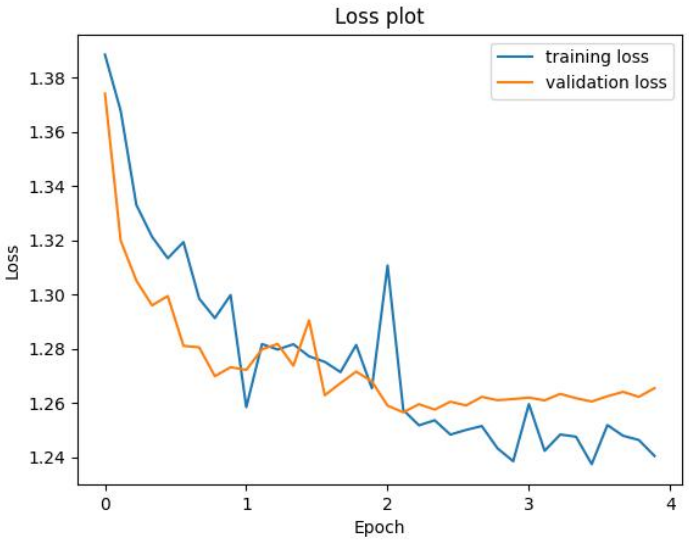

Figure B.11: Loss plot of ResNet-50 run for vectorscope representations, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.379	0.157	0.222	750
Act_Adv	0.448	0.453	0.451	2503
Com_Rom	0.402	0.412	0.407	1607
Thr_Hor_Cri	0.427	0.490	0.438	2090
micro avg	0.427	0.427	0.427	6950
macro avg	0.414	0.381	0.386	6950
weighted avg	0.424	0.427	0.419	6950
samples avg	0.427	0.427	0.427	6950

Table B.21: Results of ResNet-50 run for vectorscope representations, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	118	244	172	216
Act_Adv	84	1134	458	827
Com_Rom	56	522	662	367
Thr_Hor_Cri	53	630	355	1052

Table B.22: Confusion matrix of ResNet-50 run for vectorscope representations, for 4 epochs

ViT

Additional figures and tables for ViT experiments in 5.4 section.

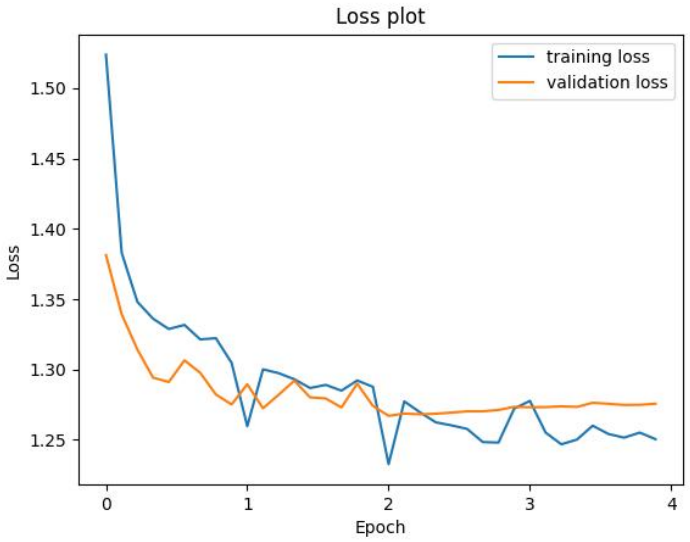

Figure B.12: Loss plot of ViT run for vectorscope representations, for 4 epochs

Genre	Precision	Recall	F1-score	Support
Drama	0.353	0.169	0.229	750
Act_Adv	0.452	0.415	0.433	2503
Com_Rom	0.391	0.395	0.393	1607
Thr_Hor_Cri	0.424	0.541	0.475	2090
micro avg	0.422	0.422	0.422	6950
macro avg	0.405	0.380	0.382	6950
weighted avg	0.419	0.422	0.414	6950
samples avg	0.422	0.422	0.422	6950

Table B.23: Results of ViT run for vectorscope representations, for 4 epochs

Genre	Drama	Act_Adv	Com_Rom	Thr_Hor_Cri
Drama	127	207	187	229
Act_Adv	102	1039	463	899
Com_Rom	59	504	635	409
Thr_Hor_Cri	72	550	338	1130

Table B.24: Confusion matrix of ViT run for vectorscope representations, for 4 epochs

B.2.5 Binary Classification

Additional figures and tables for Binary Classification experiments in 5.5 section.

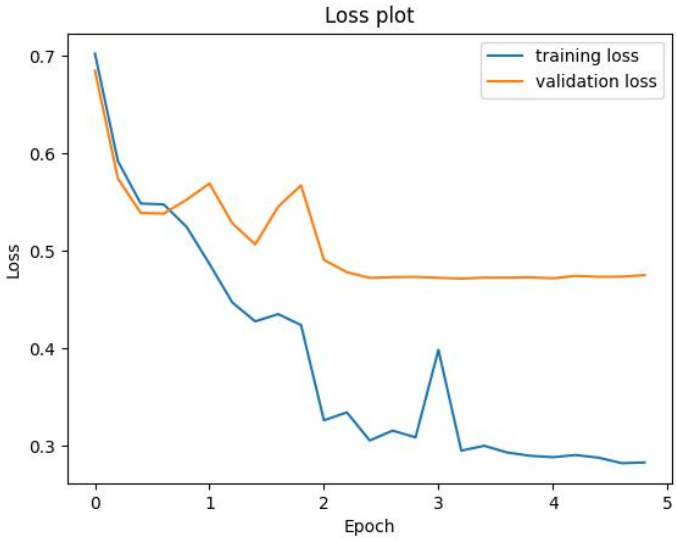

Figure B.13: Loss plot of ResNet-50 run for raw frames, for 5 epochs, random distribution

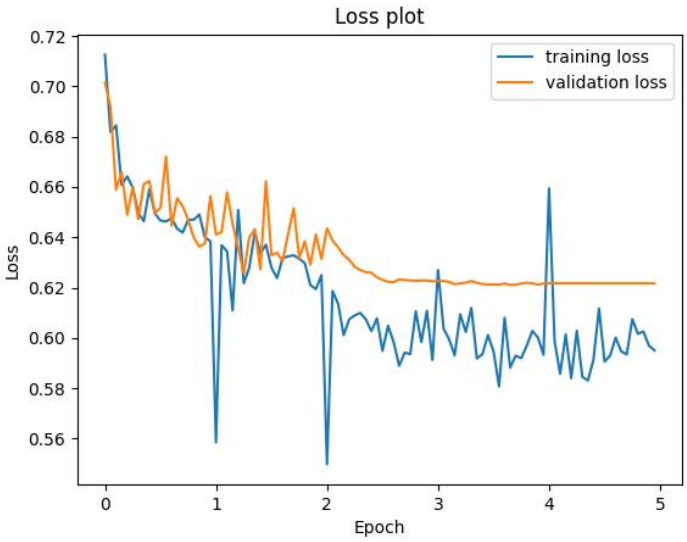

Figure B.14: Loss plot of VGG-16 run for vectorscope representations, for 5 epochs, random distribution

Figure B.15: Loss plot of ResNet-50 run for raw frames, for 5 epochs, "separate" distribution

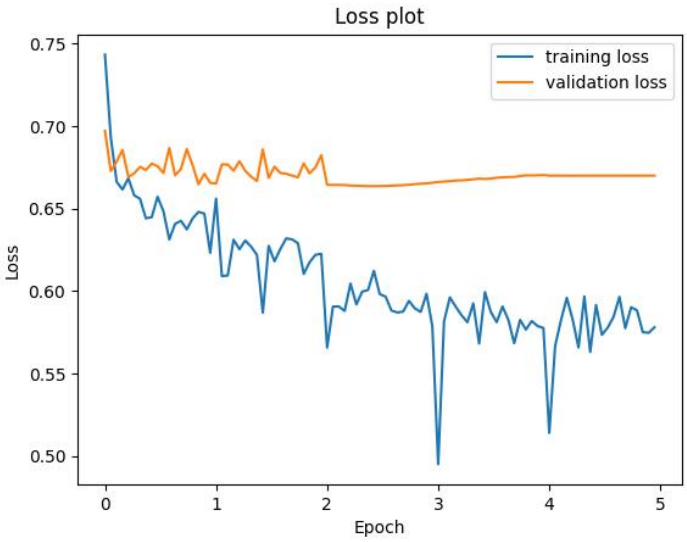

Figure B.16: Loss plot of VGG-16 run for vectorscope representations, for 5 epochs, "separate" distribution

Bibliography

[1] Zeeshan Rasheed and Mubarak Shah. «Movie Genre Classification By Exploiting Audio-visual Features Of Previews». In: *2002 IEEE International Conference on Communications*. Orlando, Florida, United States, 2002 (cit. on p. 5).

[2] Gabriel S. Simões, Jônatas Wehrmann, Rodrigo C. Barros, and Duncan D. Ruiz. «Movie Genre Classification with Convolutional Neural Networks». In: *2016 IEEE International Conference on Industrial Technology (ICIT)*. Porto Alegre, Rio Grande do Sui, Brazil, 2016 (cit. on p. 5).

[3] Marina Ivasic-Kos, Miran Pobar, and Luka Mikec. «Movie Posters Classification into Genres Based on Low-level Features». In: Rijeka , Croatia, May 2014 (cit. on p. 6).

[4] Nirman Dave. «Predicting Movie Genres from Movie Posters». In: Amherst, Massachusetts, United States (cit. on p. 6).

[5] Gabriel Barney and Kris Kaya. «Predicting Genre from Movie Posters». In: (cit. on p. 6).

[6] Kaggle. *The Movies Dataset*. 2017. URL: `https://www.kaggle.com/rounakbanik/the-movies-dataset/kernels` (cit. on p. 6).

[7] Nayeem Hossain, Md. Martuza Ahamad, Sakifa Aktar, and Mohammad Ali Moni. «Movie Genre Classification with Deep Neural Network using Poster Images». In: *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*. Dhaka, Bangladesh, Feb. 2021 (cit. on p. 6).

[8] Quan Hoang. «Predicting Movie Genres Based on Plot Summaries». In: Amherst, Massachusetts, United States, Jan. 2018 (cit. on p. 7).

[9] Bartolomeo Vacchetti, Tania Cerquitelli, and Riccardo Antonino. «Cinematographic Shot Classification through Deep Learning». In: *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. Torino, Italy, 2020 (cit. on p. 7).

[10] Jia Deng, Wei Dong, Richard Socherand Li-Jia Li, Kai Li, and Li Fei-Fei. «Imagenet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Miami, Florida, United States, June 2009, pp. 248–255 (cit. on pp. 7, 33).

[11] Bartolomeo Vacchetti and Tania Cerquitelli. «Cinematographic Shot Classification with Deep Ensemble Learning». In: (Mar. 2022) (cit. on p. 7).

[12] Bartolomeo Vacchetti and Tania Cerquitelli. «Movie Lens: Discovering and Characterizing Editing Patterns in the Analysis of Short Movie Sequences». In: Torino, Italy (cit. on p. 8).

[13] SHOTDECK. URL: `https://shotdeck.com/` (cit. on p. 9).

[14] Flim. URL: `https://beta.flim.ai/` (cit. on p. 9).

[15] SHOT.CAFE. URL: `https://shot.cafe/` (cit. on p. 9).

[16] Max Bain, Rémi Lacroix, Arsha Nagrani, and Andrew Brown. URL: `https://github.com/m-bain/CondensedMovies` (cit. on p. 9).

[17] Rotten Tomatoes. URL: `https://www.youtube.com/channel/UC3gNmTGu-TTbFPpfSs5kNkg` (cit. on p. 9).

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *ImageNet Large Scale Visual Recognition Challenge*. Toronto, Canada, Sept. 2012 (cit. on p. 30).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: *ICLR 2015, International Conference on Learning Representations 2015*. Oxford, England, May 2015 (cit. on p. 30).

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 (cit. on p. 30).

[21] Alexey Dosovitskiy et al. «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». In: *ICLR 2021, International Conference on Learning Representations 2021*. June 2021 (cit. on pp. 49, 50).