Predicting Spotify Genres

A. Main Document

i. The Spotify Data Set

For this project, we used a dataset derived from HuggingFace, which contains a tabular dataset of Spotify tracks in CSV format with 114,000 rows and 20 columns. It was collected using the Spotify Web API and Python. The unit of observation is one Spotify track (or song). The dataset includes the following features:

- **track_id**: The Spotify ID for the track
- **artists**: The artists' names who performed the track
- **album_name**: The album name in which the track appears
- **track_name**: Name of the track
- **popularity**: A value between 0 and 100, with 100 being the most popular
- **duration_ms**: The track length in milliseconds
- **explicit**: Whether or not the track has explicit lyrics
- **danceability**: Suitability of a track is for dancing. A value of 0.0 is least danceable and 1.0 is most danceable.
- **energy**: A measure from 0.0 to 1.0 that represents a perceptual measure of intensity and activity.
- **key**: The key the track is in.
- **loudness**: The overall loudness of a track in decibels (dB)
- **mode**: Modality (major or minor) of a track
- **speechiness**: The presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.
- **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic
- **instrumentalness**: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content
- **liveness**: Detects the presence of an audience in the recording.
- **valence**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
- **tempo**: The overall estimated tempo of a track in beats per minute (BPM).
- **time_signature**: An estimated time signature.
- **track_genre**: The genre in which the track belongs.

The target variable is **track_genre**, representing the track's genre.

ii. Overview of the Problem

The focus of this project is to classify the genre of a track based on its feature attributes, and ultimately to predict the genre of a new track given its feature attributes. The data set includes a total of 114 unique genres at relatively balanced counts. However, to reduce the complexity of the problem and enable accurate predictions, we reduce the data to tracks belonging to twenty different genres, selected from the original pool of 114. This set of twenty was selected because they are the twenty most populated classes after preprocessing the data, further detailed in Appendix ii.

iii. Key Methodology

The key methodology that proved most effective for addressing the genre prediction problem began with rigorous data preprocessing. This step involved removing null values, duplicate rows, and irrelevant features such as artist names and track IDs, which do not contribute meaningful information for genre prediction. Feature selection was then performed using LASSO logistic regression and random forests, both of which are robust methods for identifying the most important predictors in high-dimensional datasets. By selecting the top 10 features that had the greatest impact on genre classification, we reduced model complexity while retaining the most relevant information. Finally, we trained a random forest model, which has proven to be a strong classifier for multi-class classification tasks, yielding an accuracy of 68%. Random forests work by aggregating the predictions of multiple decision trees, each trained on a different subset of the data, which helps to reduce overfitting and improve model generalizability. This ensemble approach effectively captures complex relationships in the data and improves classification accuracy.

iv. Results

Model	Test Accuracy	Validation Accuracy
Logistic Regression	55.0%	53.1%
Random Forests	**65.0%**	**64.1%**
Decision Trees	55.2%	50.1%
4 Layer Neural Network	57.7%	57.4%
6 Layer Neural Network	61.0%	60.4%
10 Layer Neural Network	59.1%	51.9%

We find that Random Forests yield the highest test and validation accuracy among the six models evaluated. To ensure robust performance comparisons, we employed 5-fold cross-validation, which splits the dataset into five subsets, iteratively training on four and validating on the fifth. This approach helps mitigate overfitting and provides a reliable estimate of model performance on unseen data.

Random Forests excelled due to their ability to handle high-dimensional data, robustness to noise, and the inclusion of feature randomness, which reduces overfitting compared to simpler models like decision trees. Logistic regression assumes a linear relationship between the features and the target variable, which may not capture complex, nonlinear patterns in the data. While it is efficient and interpretable, its performance is limited in datasets with intricate interactions among features. In contrast, Random Forests excel in such scenarios by creating nonlinear decision boundaries through multiple decision trees, allowing them to better adapt to complex data structures. On the other hand, neural networks often require a significant amount of data, computational resources, and hyperparameter tuning to achieve their full potential. In our case, the dataset size may have been insufficient to leverage the advantages of neural networks fully, leading to overfitting or suboptimal generalization.

Additionally, the model provided insights into feature importance, helping to identify key drivers of prediction. However, despite its strengths, the Random Forest method has limitations. It requires careful tuning of hyperparameters (e.g., number of trees, maximum depth) to optimize performance and is computationally intensive compared to linear models. Furthermore, its ensemble nature makes it less interpretable, which may not align with applications requiring clear, explainable models.

Future improvements could include dimensionality reduction techniques or feature engineering, as this could further enhance performance by simplifying the feature space and reducing noise. By addressing these limitations and exploring complementary methods, we aim to refine our predictive capabilities further.

B. Appendix

1. Exploratory Data Analysis

The descriptive statistics of the pre-processed data set shows that most of the features ranges between [0,1] while the rest vary drastically: some ranges between [0,100] (e.g. `popularity`), some [0,11] (e.g. `key`), or even to negative values (e.g. `loudness`). Thus, it will be important to standardize and/or mean center these features accordingly for the remainder of the project.

Like some of the categorical predictor variables, many of the continuous predictor variables (duration_ms, danceabilty, energy, loudness, speechiness, acousticness, and instrumentalness) follow non-normal distributions with a handful of outliers as illustrated by their overlaid density and box plots. This will be addressed by scaling and selecting features appropriately depending on the assumptions made by different modeling techniques (e.g. distance-based clustering methods may be sensitive to outliers, linear regression assumes normality of residuals and homoscedasticity).

Suggested by the heatmap of each feature's mean value per genre, some of the predictor feature class imbalance may be associated with specific classes in the response variable `track_genre`. For instance, `explicit`-ness is especially present in genres like grindcore, black-metal, and hardstyle while genres like sertanejo, gospel, and disney lean toward being on a major scale. This indicates that applying imputation to accommodate for these imbalances may negatively affect the class balance in the response variable "track_genre," which is arguably more important than that of predictor variables.

The correlation matrix between predictor variables found multicollinearity. Specifically the absolute correlation coefficient was greater than 0.5 between valence and danceability (0.52), energy and loudness (-0.73), energy and acousticness (-0.73), and loudness and acousticness (-0.53). Their scatterplots further illustrate how their relationships are non-linear. It is important to note these highly-correlated features since modeling techniques (e.g. logistic regression) can make assumptions about the absence of multicollinearity.

We explored how each numerical feature's distribution varies according to the categorical features "key," "explicit," and "mode" via category-annotated density plots. The numerical features, except perhaps "energy" and "acoustic", follow relatively similar distributions between the 11 "key" classes. Likewise, the distributions were similar between the 2 "explicit" classes and the 2 "mode" classes. We also explored how each categorical feature's distribution varies across the top 20 genres via stacked bar graphs.

2. Pre-processing

The dataset was given by 114000 observations across 21 features. After familiarizing ourselves with the feature descriptions and the basic structure of the data, we dropped the column "Unnamed: 0" unique for each observation and removed a single observation with null values along with 894 complete duplicates. However, we found that "track ID" was not unique for each observation, indicating there were duplicates in terms of "track ID." We discovered the reason behind this was that each unique track, instead of observation, was associated with a track ID. After isolating each feature as a potential explanation for the track ID duplicates, we found that there were 38948 tracks were duplicates of each other by being listed under multiple genres, and likewise for 9238 tracks

under different albums, 293 tracks under different popularity ratings, 2 tracks under different track names, and 2 tracks under different tempo. We left the genre duplicates intact to preserve information for genre classification. The highest popularity rating was kept for identical tracks with multiple popularity ratings to best reflect their performance. Identical tracks with multiple track names, album names, and tempo were handled as well by keeping their first occurrences.

The “track_name”, “album_name”, and “artists” features were dropped following this process since they are not within the scope of this project. This leaves us with 106811 observations across 15 features, excluding the response variable “track_genre.” The ordinal features “popularity", "key", and "time_signature" were given by numerical values, thus, are already encoded into quasi-interval variables and no further action is needed. The nominal features "explicit" and "mode" were one-hot encoded since their categories lack a natural order. These categorical features all exhibit some degree of class imbalance. Notably, “explicit” is dominated by its “non-explicit” class, “time_signature” is dominated by its “4/4” class, and “popularity” is dominated by its “0” rating class. However, not only are large proportions of data missing on these features, their missingness are associated with specific genres. Thus, imputation may not be reliable in this case, especially when our response variable “track_genre” is already very balanced. Observations for the top 20 most common genres were kept for the project. After preprocessing the dataset, a version with the original values were kept for conducting EDA, a z-score normalized version and a min-max scaled version with 80/20 train-test split were prepared for modeling techniques with different data preferences

3. Regression Analysis

Since the main problem we address with this project involves classification and not regression, a linear regression analysis is not directly useful for our goals. However, in exploring the data set and settling on a topic for the primary problem to explore, we conducted a regression analysis using loudness as the response variable and energy as the predictor variable.

The initial model achieved an R-squared value of approximately 0.593 on the training data, meaning that 59.3% of the variability in loudness is explained by energy. Considering that this model uses only one predictor variable, the model is quite good, though the analysis is brief and can be greatly expanded upon.

We attempted to improve the initial model by employing ridge regression with an lambda value of 1. Ridge regression applies an L2 penalty on the least squares loss function and ultimately prevents overfitting by encouraging the coefficient parameters toward zero. The R-squared value of this ridge regression model was also approximately 0.593, indicating that no accuracy was lost. This is explained by comparing mean-squared errors (MSE) of the initial model fit on the training and testing sets. The quantities were similar (7.809 vs 8.350, respectively), indicating that there was little need for regularization against overfitting. Thus, the ridge regression model did not change much from the original model.

The prediction of loudness using other/multiple predictors is a topic that could be further explored, but is outside of the scope of this project.

4. Logistic Regression

Logistic regression, when compared to standard regression analysis, was much more applicable to the research problem we were looking to solve. Since logistic regression aims to predict a value from 0 to 1 for the probability for a class label, we were able to leverage it to predict the genre of each song in the dataset.

Logistic Regression gave us our first insights into the importance of each of the features for each genre and collectively after averaging the absolute values of each feature importance. Features like popularity and loudness were consistently important for each of the genres, with features like acousticness and danceability also playing a heavy role in the predictive ability of the model. Features such as tempo, mode, time signature, and key had very low importance to the performance of the model.

Our logistic regression model ended up having an accuracy of 0.55, which when compared to some of our other models was a bit low. Later cross validation gave us an accuracy of 0.53. However, the information regarding which features were most important in determining the genre of a song was valuable information that would show up more than once in our research.

5. Random Forest

Random forest uses a large number of decision trees to come to a decision about the most likely class label of a particular observation based on the value of a predictor, whether it is above/below a certain threshold or a true/false feature.

Our random forest model mirrored some earlier results seen in the logistic regression analysis, with features like popularity and acousticness once again being features with importance while time signature and explicit were once again at the bottom of the list.

However, in contrast with the logistic regression, our random forest model resulted in the highest accuracy of every model we fitted. We were able to reach an accuracy of 65% and a cross validation accuracy of 64.1%.

This model ended up being the best performance of each of the ones we fitted. Random forest's handling of outliers and collinearity most likely contributed to its success in comparison to other models. It also fits the context of our problem, with several numeric and boolean variables that we are aiming to predict a categorical variable from. Its process of using multiple decision trees is vital for preventing overfitting. However, random forest uses a significant amount of resources to fit the model, so runtimes ended up being pretty long in our application.

6. Clustering Analysis

Principal Component Analysis is a technique for dimensionality reduction when a data set is very high-dimensional (i.e., contains many features). The goal is to simplify the data set while retaining as much information or variability in the data as possible.

Clustering is an unsupervised machine learning technique that creates groups in the data. For this clustering analysis, we attempt to cluster based on time signature.

Clustering is susceptible to a phenomenon known as the curse of dimensionality, in which a data set is so high-dimensional and complex that clustering is difficult to perform and largely inaccurate, as the more complex a data set is, the less meaningful distance metrics become. Thus, reducing the dimension of the data set may aid in clustering effectiveness. Our approach involves performing k-means clustering on the

data, then performing principal component analysis to create a transformed (simpler) data set, performing k-means clustering again on the new PCA-transformed data, and finally comparing evaluation metrics for the two clustering schemes.

Results showed that the first two principal components capture about 34.16% of the variability in the data, and after that, each principal component made a small, consistent contribution. Unfortunately, keeping only the first two principal components would simply result in a data set that does not capture nearly enough information from the original data to yield generalizable insights from analysis. Moreover, if we want to retain most of the information in the original data, let's say 90%, then we would need to keep the first ten principal components, which is not a very successful dimensionality reduction down from fourteen original features.

Keeping the first nine principal components and clustering with the new PCA transformed data set, we obtained only a slightly higher silhouette score of 0.1296, compared to a score of 0.1098 before performing PCA. Silhouette score is a measure of how tightly and distinctly the data is clustered, where 1 is tightly clustered and 0 is loosely (and indistinctively) clustered. It is one measure of effectiveness for a clustering algorithm. The calculated silhouette scores for both clustering schemes suggests that the effects of PCA are minimal for this data set, so we did not employ it in the main analysis of our classification of track genres.

The same approach may be taken to cluster the data based on other categorical features such as key, though we did not explore these techniques further so we could focus on the key methodology.

7. Neural Network

Neural networks are a powerful tool for multi-class classification tasks, capable of modeling complex patterns in data through their layered architectures. This makes them suitable for predicting Spotify genres, given the non-sequential, tabular nature of the dataset. However, due to the lack of spatial, temporal, or time-sensitive data, architectures like CNNs and RNNs, commonly used for image or sequence processing, are not ideal for this task. Instead, we designed three feedforward neural network architectures with 4, 6, and 10 layers, incorporating dropout and batch normalization layers to mitigate overfitting and stabilize training. To further optimize model performance, we tuned key hyperparameters, including batch size, dropout rate, and learning rate, through systematic experimentation.

The choice of architecture depth was driven by a desire to balance model capacity and training efficiency. The 4-layer model served as a baseline, providing a simpler structure to establish performance benchmarks. The 6-layer model offered additional depth, hypothesized to capture more complex relationships in the data without introducing significant overfitting. The 10-layer model, the most complex of the three, was designed to evaluate the trade-off between capacity and generalization. To ensure robust training, we employed ReLU activation functions for non-linearity, softmax for the output layer, and categorical cross-entropy as the loss function, given the multi-class nature of the task.

The results demonstrated that model performance improved with increased depth up to a point, beyond which overfitting became apparent. The 4-layer model achieved an accuracy of 57.7%, establishing a reliable baseline with stable convergence and minimal overfitting. The 6-layer model outperformed the

others with an accuracy of 61.0%, benefiting from its enhanced capacity to learn complex patterns while maintaining generalization. The 10-layer model, while theoretically more powerful, took significantly longer to train despite only achieving an accuracy of 59.1%, which is lower than the 6-layer model. This suggests that the dataset's size and complexity were insufficient to fully leverage the additional depth.

Overall, the 6-layer model proved to be the most effective architecture, achieving the highest accuracy and demonstrating a good trade-off between capacity and generalization. These findings underscore the importance of tailoring architecture and hyperparameter selection to the characteristics of the dataset. Future work could explore advanced regularization techniques, such as early stopping, to further enhance the performance of deeper architectures. Additionally, expanding the dataset or augmenting it with derived features could better support the capacity of more complex models.

8. Hyperparameter Tuning

Hyperparameter tuning played a pivotal role in optimizing the models. For the neural networks, batch size impacts how often weight updates occur; smaller batch sizes often generalize better but require more computational resources. Dropout rates were adjusted to regulate overfitting by randomly deactivating neurons during training, encouraging robustness. Learning rate, a critical parameter controlling the step size of weight updates, was carefully tuned to balance convergence speed and stability.

Across all deep learning architectures, the optimal hyperparameters varied. The 4 layer model performed best with a learning rate of 0.001, batch size of 64, and dropout rate of 0.2. Both the 6 and 10 layer model performed best with a learning rate 0.001, batch size of 32, and dropout rate of 0.1. This makes sense, since more complex models require smaller learning rates and smaller batch sizes to achieve stable training and reduce overfitting.

For the random forest, decision tree, and logistic regression models, we utilized grid search to systematically explore combinations of hyperparameters and employed cross-validation to ensure robust evaluation. In the random forest, the number of trees (n_estimators), maximum tree depth (max_depth), and the minimum samples per split and leaf (min_samples_split and min_samples_leaf) were tuned. The optimal configuration balanced model complexity and generalization, achieving higher accuracy and lower variance. Similarly, decision trees were optimized for depth, and leaf node size, improving predictive power without overfitting. Logistic regression benefited from regularization parameter tuning (C) to control overfitting, ensuring a balance between model simplicity and accuracy. These fine-tuning efforts collectively enhanced model performance, enabling more reliable predictions and insights from the data.

The Spotify Tracks Dataset:

This is a dataset of Spotify tracks over a range of 114 different genres. Each track is associated with the following features including the artist, track name, and various audio features. We decided to focus on predicting `track_genre` given the dataset's numerical features while setting aside the categorical features like `artists`, `album_name`, and `track_name` which may not be as informative.

</br>

`track_id`: The Spotify ID for the track

`artists`: The artists' names who performed the track. If there is more than one artist, they are separated by a ;

`album_name`: The album name in which the track appears

`track_name`: Name of the track

`popularity`: The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity.

`duration_ms`: The track length in milliseconds

`explicit`: Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown)

`danceability`: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable

`energy`: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale

`key`: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1

`loudness`: The overall loudness of a track in decibels (dB)

`mode`: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0

`speechiness`: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks

`acousticness`: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic instrumentalness: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content

`liveness` : Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live

`valence` : A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry)

`tempo` : The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration

`time_signature` : An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.

`track_genre` : The genre in which the track belongs

I. Data Pre-Processing

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn import preprocessing

pd.set_option('display.max_columns', None)
pd.options.display.max_colwidth = 500
pd.options.display.max_rows = 100
```

```
/Users/peiyuanlee/miniforge3/envs/myenv/lib/python3.11/site-packages/pandas/core/arrays/masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.
3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
# Load the dataset
df_raw = pd.read_csv("hf://datasets/maharshipandya/spotify-tracks-dataset/dataset.csv")
```

In [ ]:
```
# Examine the first 10 rows (song tracks)
df_raw.head(10)
```

Out[ ]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	vale
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	230666	False	0.676	0.4610	1	-6.746	0	0.1430	0.0322	0.000001	0.3580	0.7
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	149610	False	0.420	0.1660	1	-17.235	1	0.0763	0.9240	0.000006	0.1010	0.2
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	210826	False	0.438	0.3590	0	-9.734	1	0.0557	0.2100	0.000000	0.1170	0.1
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Soundtrack)	Can't Help Falling In Love	71	201933	False	0.266	0.0596	0	-18.515	1	0.0363	0.9050	0.000071	0.1320	0.1
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	198853	False	0.618	0.4430	2	-9.681	1	0.0526	0.4690	0.000000	0.0829	0.1
5	5	01MVOl9KtVTNfFiBU9I7dc	Tyrone Wells	Days I Will Remember	Days I Will Remember	58	214240	False	0.688	0.4810	6	-8.807	1	0.1050	0.2890	0.000000	0.1890	0.6
6	6	6Vc5wAMmXdKIAM7WUoEb7N	A Great Big World;Christina Aguilera	Is There Anybody Out There?	Say Something	74	229400	False	0.407	0.1470	2	-8.822	1	0.0355	0.8570	0.000003	0.0913	0.0
7	7	1EzrEOXmMH3G43AXT1y7pA	Jason Mraz	We Sing. We Dance. We Steal Things.	I'm Yours	80	242946	False	0.703	0.4440	11	-9.331	1	0.0417	0.5590	0.000000	0.0973	0.7
8	8	0IktbUcnAGrvD03AWnz3Q8	Jason Mraz;Colbie Caillat	We Sing. We Dance. We Steal Things.	Lucky	74	189613	False	0.625	0.4140	0	-8.700	1	0.0369	0.2940	0.000000	0.1510	0.6
9	9	7k9GuJYLp2AzqokyEdwEw2	Ross Copperman	Hunger	Hunger	56	205594	False	0.442	0.6320	1	-6.770	1	0.0295	0.4260	0.004190	0.0735	0.1

In [ ]:

```
# Examine a random sample of 10 rows (song tracks)
df_raw.sample(n=10)
```

Out[ ]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness
3647	3647	27sytaeEm6TDVMpdExyVfd	O Rappa	Só as Melhores do Pop Rock Brasileiro	A feira	0	239533	False	0.713	0.886	7	-7.750	1	0.0411	0.0012	0.000008
56533	56533	0w2emroZUEoocjdVbK5F69	Claire Rosinkranz	Die For You	Boy In A Billion	0	207678	False	0.513	0.458	10	-7.300	1	0.1420	0.4370	0.000000
63240	63240	0qjhMbCmCSZC2f4qohgcxa	Eikichi Yazawa	the Name Is... (50th Anniversary Remastered)	アリよさらば - Remastered 2022	38	255893	False	0.682	0.828	11	-8.009	0	0.0571	0.2410	0.000000
17958	17958	1bYscy2XwW0fKJoNrALWP7	Deathpact	SPLIT // PERSONALITY	SONG SIX	37	290666	False	0.541	0.891	4	-4.399	0	0.0595	0.0703	0.381000
54714	54714	6ApQUXDhO9qcqVz6Q4OCJY	Datassette	Existenzmaximum - EP	Holiday 88	10	282125	False	0.713	0.707	4	-14.870	0	0.0511	0.0827	0.879000
78606	78606	5wOby0SgajxMlOFce5HLyh	voXXclub	Hitmedley	Hitmedley	27	358653	False	0.407	0.936	3	-4.503	1	0.1950	0.0280	0.000000
97310	97310	4nzbkdiJ0GzxNUhVFz43j4	Atitude 67	Atitude 67 (Ao Vivo)	Casal Do Ano (Plutão) - Ao Vivo	51	207853	False	0.704	0.865	4	-5.494	1	0.0678	0.4890	0.000000
106326	106326	3tXgGYRE0spiBVxaa9Xr79	Lars Winnerbäck	Hosianna	Utkast till ett brev	41	278106	False	0.662	0.897	4	-4.742	0	0.0356	0.0173	0.005950
81986	81986	4tjLYTXFqZhkUDga4bQ0yl	Neha Kakkar;Dhvani Bhanushali;Ikka;Tanishk Bagchi	Dilbar (From "Satyameva Jayate")	Dilbar (From "Satyameva Jayate")	66	184432	False	0.725	0.912	9	-3.665	0	0.0851	0.1550	0.000077
84776	84776	5gOnivVq0hLxPvIPC00ZhF	T. Rex	Electric Warrior	Cosmic Dancer	58	266533	False	0.363	0.803	0	-8.089	1	0.0600	0.0117	0.006010

In [ ]:
```
# Examine the number of features and observations
df_raw.shape
```

Out[ ]:
```
(114000, 21)
```

Checking for feature relevance, duplicates, and missing data:

In [ ]:
```
# Examine features given by the dataset
df_raw.columns
```

Out[ ]:
```
Index(['Unnamed: 0', 'track_id', 'artists', 'album_name', 'track_name',
       'popularity', 'duration_ms', 'explicit', 'danceability', 'energy',
       'key', 'loudness', 'mode', 'speechiness', 'acousticness',
       'instrumentalness', 'liveness', 'valence', 'tempo', 'time_signature',
       'track_genre'],
      dtype='object')
```

In [ ]:
```
# Examine feature data types and missingness
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        114000 non-null  int64  
 1   track_id          114000 non-null  object 
 2   artists           113999 non-null  object 
 3   album_name        113999 non-null  object 
 4   track_name        113999 non-null  object 
 5   popularity        114000 non-null  int64  
 6   duration_ms       114000 non-null  int64  
 7   explicit          114000 non-null  bool   
 8   danceability      114000 non-null  float64
 9   energy            114000 non-null  float64
 10  key               114000 non-null  int64  
 11  loudness          114000 non-null  float64
 12  mode              114000 non-null  int64  
 13  speechiness       114000 non-null  float64
 14  acousticness      114000 non-null  float64
 15  instrumentalness  114000 non-null  float64
 16  liveness          114000 non-null  float64
 17  valence           114000 non-null  float64
 18  tempo             114000 non-null  float64
 19  time_signature    114000 non-null  int64  
 20  track_genre       114000 non-null  object 
dtypes: bool(1), float64(9), int64(6), object(5)
memory usage: 17.5+ MB
```

We see that the datset has 114000 samples for 21 features with the only missing data being `artists`, `album_name`, and `track_name` each having 1 null observation. Since this is not many observations, it is safe to just drop them.

In [3]:
```
df_raw.dropna(axis=0,inplace=True)
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 113999 entries, 0 to 113999
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Unnamed: 0        113999 non-null  int64
 1   track_id          113999 non-null  object
 2   artists           113999 non-null  object
 3   album_name        113999 non-null  object
 4   track_name        113999 non-null  object
 5   popularity        113999 non-null  int64
 6   duration_ms       113999 non-null  int64
 7   explicit          113999 non-null  bool
 8   danceability      113999 non-null  float64
 9   energy            113999 non-null  float64
 10  key               113999 non-null  int64
 11  loudness          113999 non-null  float64
 12  mode              113999 non-null  int64
 13  speechiness       113999 non-null  float64
 14  acousticness      113999 non-null  float64
 15  instrumentalness  113999 non-null  float64
 16  liveness          113999 non-null  float64
 17  valence           113999 non-null  float64
 18  tempo             113999 non-null  float64
 19  time_signature    113999 non-null  int64
 20  track_genre       113999 non-null  object
dtypes: bool(1), float64(9), int64(6), object(5)
memory usage: 18.4+ MB
```

In [ ]:
```python
# Ensuring that 'track_id' and 'Unnamed: 0" are entirely arbitrary
print("track_id: ",df_raw.track_id.nunique(), "/",113999)
print("Unnamed: 0: ",df_raw['Unnamed: 0'].nunique(), "/",113999)
```

```
track_id:  89740 / 113999
Unnamed: 0:  113999 / 113999
```

The feature `Unnamed: 0` is unique per track, thus, can be removed. However, `track_id` seems to have duplicates, perhaps in terms of nominal variables like `explicit`, `mode`, `key`, or `track_genre` since there can be different versions of the same song in terms of these variables. We will isolate each feature as a potential explanation for the duplicates.

In [4]:
```python
# Dropping the feature 'Unnamed: 0"
df_raw.drop('Unnamed: 0', axis=1, inplace=True)
```

In [5]:
```python
# Sample the first 20 rows that have duplicated track IDs
dup_tracks = df_raw[df_raw.duplicated(subset=['track_id'], keep=False)].sort_values(by='track_id')
dup_tracks.head(20)
```

Out[5]:

	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence
15028	001APMDOl3qtx1526T11n1	Pink Sweat$;Kirby	New RnB	Better	0	176320	False	0.613	0.471	1	-6.644	0	0.1070	0.31600	0.000001	0.1170	0.406
103211	001APMDOl3qtx1526T11n1	Pink Sweat$;Kirby	New RnB	Better	0	176320	False	0.613	0.471	1	-6.644	0	0.1070	0.31600	0.000001	0.1170	0.406
85578	001YQlnDSduXd5LgBd66gT	Soda Stereo	Soda Stereo (Remastered)	El Tiempo Es Dinero - Remasterizado 2007	38	177266	False	0.554	0.921	2	-4.589	1	0.0758	0.01940	0.088100	0.3290	0.700
100420	001YQlnDSduXd5LgBd66gT	Soda Stereo	Soda Stereo (Remastered)	El Tiempo Es Dinero - Remasterizado 2007	38	177266	False	0.554	0.921	2	-4.589	1	0.0758	0.01940	0.088100	0.3290	0.700
91801	003vvx7Niy0yvhvHt4a68B	The Killers	Hot Fuss	Mr. Brightside	86	222973	False	0.352	0.911	1	-5.230	1	0.0747	0.00121	0.000000	0.0995	0.236
3257	003vvx7Niy0yvhvHt4a68B	The Killers	Hot Fuss	Mr. Brightside	86	222973	False	0.352	0.911	1	-5.230	1	0.0747	0.00121	0.000000	0.0995	0.236
2106	003vvx7Niy0yvhvHt4a68B	The Killers	Hot Fuss	Mr. Brightside	86	222973	False	0.352	0.911	1	-5.230	1	0.0747	0.00121	0.000000	0.0995	0.236
33178	004h8smbloAkUNDJvVKwkG	Ouse;Powfu	Loners Diary	Lovemark	58	219482	True	0.808	0.331	5	-13.457	1	0.0557	0.13100	0.000000	0.2250	0.337
94239	004h8smbloAkUNDJvVKwkG	Ouse;Powfu	Loners Diary	Lovemark	58	219482	True	0.808	0.331	5	-13.457	1	0.0557	0.13100	0.000000	0.2250	0.337
97533	006rHBBNLJMpQs8fRC2GDe	Calcinha Preta;Gusttavo Lima	CP 25 Anos (Ao Vivo em Aracaju)	Agora Estou Sofrendo - Ao Vivo	47	260510	False	0.605	0.678	0	-3.257	1	0.0311	0.64200	0.000000	0.1570	0.439
77391	006rHBBNLJMpQs8fRC2GDe	Calcinha Preta;Gusttavo Lima	CP 25 Anos (Ao Vivo em Aracaju)	Agora Estou Sofrendo - Ao Vivo	47	260510	False	0.605	0.678	0	-3.257	1	0.0311	0.64200	0.000000	0.1570	0.439
35138	006rHBBNLJMpQs8fRC2GDe	Calcinha Preta;Gusttavo Lima	CP 25 Anos (Ao Vivo em Aracaju)	Agora Estou Sofrendo - Ao Vivo	47	260510	False	0.605	0.678	0	-3.257	1	0.0311	0.64200	0.000000	0.1570	0.439
112131	006tmNZLXEXPqdb23wwSN1	İlhan İrem	Bezginin Gizli Mektupları	Yemyeşil Bir Deniz	44	358173	False	0.486	0.568	9	-9.199	0	0.0417	0.65200	0.000000	0.8340	0.650
64662	006tmNZLXEXPqdb23wwSN1	İlhan İrem	Bezginin Gizli Mektupları	Yemyeşil Bir Deniz	44	358173	False	0.486	0.568	9	-9.199	0	0.0417	0.65200	0.000000	0.8340	0.650
62346	006tmNZLXEXPqdb23wwSN1	İlhan İrem	Bezginin Gizli Mektupları	Yemyeşil Bir Deniz	44	358173	False	0.486	0.568	9	-9.199	0	0.0417	0.65200	0.000000	0.8340	0.650
63142	006tmNZLXEXPqdb23wwSN1	İlhan İrem	Bezginin Gizli Mektupları	Yemyeşil Bir Deniz	44	358173	False	0.486	0.568	9	-9.199	0	0.0417	0.65200	0.000000	0.8340	0.650
64246	00970cTs7LnxWt0d5Qk08m	Ella Fitzgerald	Weihnachtslieder 2022	Sleigh Ride	0	175986	False	0.593	0.287	1	-12.472	1	0.0469	0.76400	0.000000	0.1530	0.639
8095	00970cTs7LnxWt0d5Qk08m	Ella Fitzgerald	Weihnachtslieder 2022	Sleigh Ride	0	175986	False	0.593	0.287	1	-12.472	1	0.0469	0.76400	0.000000	0.1530	0.639
71588	00B7SBwrjbycLMOgAmeIU8	Red Hot Chili Peppers	Return of the Dream Canteen	Reach Out	66	251588	False	0.663	0.710	11	-5.550	0	0.0599	0.00745	0.005590	0.1470	0.487
2870	00B7SBwrjbycLMOgAmeIU8	Red Hot Chili Peppers	Return of the Dream Canteen	Reach Out	66	251588	False	0.663	0.710	11	-5.550	0	0.0599	0.00745	0.005590	0.1470	0.487

In [6]:
```
# number of complete duplicates
dup_num = dup_tracks[dup_tracks.duplicated(keep=False)].shape[0]
dup_num
```

Out[6]:
```
894
```

In [7]:
```
# Dropping the feature 'track_id"
df_raw.drop('track_id', axis=1, inplace=True)

# Remove the duplicates
df_raw.drop_duplicates(inplace=True)
```

In [8]:
```
print("number of duplicates in terms of all other features except for...")
cols_to_check = list(dup_tracks.columns)
dup_cols = list(dup_tracks.columns)
for i  in cols_to_check:
  dup_cols.remove(i)
  print(i,': ', dup_tracks[dup_tracks.duplicated(subset=dup_cols, keep=False)].shape[0]-dup_num)
  dup_cols.append(i)
```

```
number of duplicates in terms of all other features except for...
track_id :  151
artists :  0
album_name :  0
track_name :  0
popularity :  0
duration_ms :  0
explicit :  0
danceability :  0
energy :  0
key :  0
loudness :  0
mode :  0
speechiness :  0
acousticness :  0
instrumentalness :  0
liveness :  0
valence :  0
tempo :  0
time_signature :  0
track_genre :  38948
```

The duplicates seems to be the exact same tracks listed under either multiple genres (38948 of these) or listed under different track IDs (151). We will remove the tracks with duplicated track_IDs but keep the tracks listed under multiple genres since `track_genre` is our response variable.

In [9]:
```
print("number of duplicates in terms of all other features except for...")
cols_to_check = list(df_raw.columns)
dup_cols = list(df_raw.columns)
for i  in cols_to_check:
  dup_cols.remove(i)
  print(i,': ', df_raw[df_raw.duplicated(subset=dup_cols, keep=False)].shape[0])
  dup_cols.append(i)
```

```
number of duplicates in terms of all other features except for...
artists :  0
album_name :  9238
track_name :  2
popularity :  293
duration_ms :  0
explicit :  0
danceability :  0
energy :  0
key :  0
loudness :  0
mode :  0
speechiness :  0
acousticness :  0
instrumentalness :  0
liveness :  0
valence :  0
tempo :  2
time_signature :  0
track_genre :  38967
```

There are tracks that are exactly the same but received different `popularity` ratings. According to the features documentation, `popularity` is "calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are." In order to reflect the performance of the tracks, we will keep the observation with the highest popularity rating and remove the remaining duplicates.

In [10]:
```
# sort in descending order by popularity
df_raw.sort_values(by='popularity',ascending=False).head(20)

# keep only the first occurance of the duplicate (i.e. observation with the max popularity value)
dup_cols = list(df_raw.columns)
dup_cols.remove('popularity')
df_raw.drop_duplicates(subset=dup_cols, keep='first', inplace=True)
```

There are also tracks with the exact same values for all features except `track_name`, `album_name`, and `tempo`. We will examine these duplications to ensure it is fair to remove their duplicates.

In [11]:
```
# same track under different names??
dup_cols = list(df_raw.columns)
dup_cols.remove('track_name')
df_raw[df_raw.duplicated(subset=dup_cols, keep=False)].head(10)
```

Out[11]:

	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
49371	UVIQUE	InfeXious Euphoric - Chapter One	Falling	0	178374	False	0.43	0.781	3	-5.601	1	0.0334	0.0108	0.734	0.0818	0.206	75.017	4	hardstyle
49376	UVIQUE	InfeXious Euphoric - Chapter One	Falling - Radio Mix	0	178374	False	0.43	0.781	3	-5.601	1	0.0334	0.0108	0.734	0.0818	0.206	75.017	4	hardstyle

In [12]:
```
# same track under different album names??
dup_cols = list(df_raw.columns)
dup_cols.remove('album_name')
df_raw[df_raw.duplicated(subset=dup_cols, keep=False)].head(10)
```

Out[12]:

	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
26	Jason Mraz	Holly Jolly Christmas	Winter Wonderland	0	131760	False	0.620	0.309	5	-9.209	1	0.0495	0.788	0.000000	0.1460	0.664	145.363	4	acoustic
28	Jason Mraz	Christmas Time	Winter Wonderland	0	131760	False	0.620	0.309	5	-9.209	1	0.0495	0.788	0.000000	0.1460	0.664	145.363	4	acoustic
29	Jason Mraz	Perfect Christmas Hits	Winter Wonderland	0	131760	False	0.620	0.309	5	-9.209	1	0.0495	0.788	0.000000	0.1460	0.664	145.363	4	acoustic
30	Jason Mraz	Merry Christmas	Winter Wonderland	0	131760	False	0.620	0.309	5	-9.209	1	0.0495	0.788	0.000000	0.1460	0.664	145.363	4	acoustic
31	Jason Mraz	Christmas Music - Holiday Hits	Winter Wonderland	0	131760	False	0.620	0.309	5	-9.209	1	0.0495	0.788	0.000000	0.1460	0.664	145.363	4	acoustic
33	Brandi Carlile;Sam Smith	Human - Best Adult Pop Tunes	Party of One	0	259558	False	0.296	0.206	0	-11.799	1	0.0412	0.782	0.000225	0.0959	0.202	165.400	4	acoustic
34	Brandi Carlile;Sam Smith	Feeling Good - Adult Pop Favorites	Party of One	0	259558	False	0.296	0.206	0	-11.799	1	0.0412	0.782	0.000225	0.0959	0.202	165.400	4	acoustic
35	Brandi Carlile;Sam Smith	Mellow Bars R'n'B	Party of One	0	259558	False	0.296	0.206	0	-11.799	1	0.0412	0.782	0.000225	0.0959	0.202	165.400	4	acoustic
36	KT Tunstall	Chill Christmas Dinner	Lonely This Christmas	0	257493	False	0.409	0.153	6	-10.740	0	0.0306	0.939	0.000026	0.1080	0.180	85.262	4	acoustic
39	KT Tunstall	sadsadchristmas	Lonely This Christmas	0	257493	False	0.409	0.153	6	-10.740	0	0.0306	0.939	0.000026	0.1080	0.180	85.262	4	acoustic

In [13]:
```
# same track with different tempo??
dup_cols = list(df_raw.columns)
dup_cols.remove('tempo')
df_raw[df_raw.duplicated(subset=dup_cols, keep=False)].head(10)
```

Out[13]:

	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
59208	AMONGST THE ASHES	Agonizing Awakening	Exordium of Sickness	0	80948	False	0.423	0.853	1	-10.133	1	0.0382	0.00044	0.69	0.145	0.107	89.980	4	iranian
59916	AMONGST THE ASHES	Agonizing Awakening	Exordium of Sickness	0	80948	False	0.423	0.853	1	-10.133	1	0.0382	0.00044	0.69	0.145	0.107	89.977	4	iranian

In [14]:
```
# it is fair to remove their duplicates
dup_cols = list(df_raw.columns)
dup_cols.remove('track_name')
df_raw.drop_duplicates(subset=dup_cols, keep='first', inplace=True)

dup_cols = list(df_raw.columns)
dup_cols.remove('album_name')
df_raw.drop_duplicates(subset=dup_cols, keep='first', inplace=True)
```

```
dup_cols = list(df_raw.columns)
dup_cols.remove('tempo')
df_raw.drop_duplicates(subset=dup_cols, keep='first', inplace=True)
```

In [15]:
```
# No negative values except for "loudness" which is reasonable since decibels can be negative
(df_raw.select_dtypes(exclude='object')<0).any()
```

Out[15]:
```
popularity          False
duration_ms         False
explicit            False
danceability        False
energy              False
key                 False
loudness             True
mode                False
speechiness         False
acousticness        False
instrumentalness    False
liveness            False
valence             False
tempo               False
time_signature      False
dtype: bool
```

We are now left with 106811 observations after removing duplicates and obervations with missing values.

In [16]:
```
df_raw.shape
```

Out[16]:
```
(106811, 19)
```

Handling the categorical features:

The features `artists`, `album_name`, and `track_name` cannot be encoded by category in a meaningful way. Since analyzing text is out of scope for this project, we will not be considering these features.

In [17]:
```
# Unique name values
print("artists: ",df_raw.artists.nunique(), "/",106811)
print("album_name: ",df_raw.album_name.nunique(), "/",106811)
print("track_name: ",df_raw.track_name.nunique(), "/",106811)
```

```
artists:  31437 / 106811
album_name:  46512 / 106811
track_name:  73607 / 106811
```

The features `popularity`, `explicit`, `key`, `mode`, and `time_signature` are categorical variables given by numerical values. We will one-hot encode the features `explicit` and `mode` since they are nominal, meaning, their categories lack a natural order (`mode` is already encoded). On the other hand, `popularity`, `key`, and `time_signature` are quasi-interval variables so we will leave them alone to preserve their natural order.

In [18]:
```
# Encode 'explicit' as 0 or 1
label_encoder = preprocessing.LabelEncoder()
df_raw['explicit']=label_encoder.fit_transform(df_raw['explicit'])
df_raw.sample(n=20)
```

Out[18]:

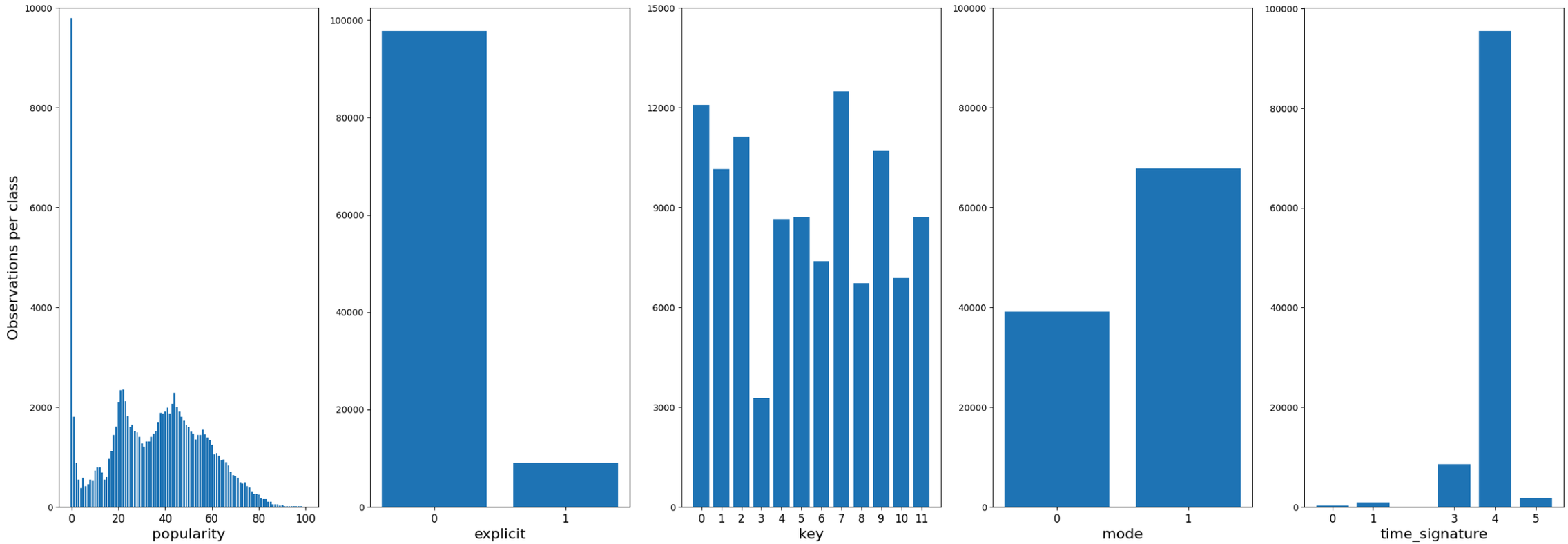
	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	tra
23084	Mark Knight;Armand Van Helden	Toolroom Amsterdam 2022	The Music Began To Play	5	145984	0	0.738	0.936	4	-3.451	0	0.2190	0.001510	0.707000	0.6170	0.2700	127.022	4	de
54092	Rival Consoles	Now Is	Running	39	251188	0	0.584	0.613	11	-13.823	1	0.0418	0.113000	0.855000	0.1080	0.0511	118.883	4	
35937	Canindé	Ao Vivo	Borbulhas de Amor (Tenho um Coração) - Burbujas de Amor	37	232266	0	0.815	0.571	0	-5.032	1	0.0302	0.308000	0.000000	0.9130	0.6150	120.093	4	
75741	Paul Cardall	Sacred Piano	Redeemer	20	352373	0	0.218	0.107	7	-18.676	1	0.0384	0.912000	0.398000	0.1180	0.0392	134.568	4	
57833	Maisie Peters	All Bops	This Is on You	0	195253	1	0.715	0.508	7	-6.899	1	0.0889	0.560000	0.000000	0.1240	0.2740	95.981	4	
9628	Aline Barros;Comunidade Evangélica Internacional da Zona Sul	Rompendo em Fé	Rompendo em Fé	44	289459	0	0.436	0.673	9	-4.640	1	0.0311	0.348000	0.000000	0.1100	0.2320	133.844	4	
48257	A Tribe Called Quest	People's Instinctive Travels and the Paths of Rhythm (25th Anniversary Edition)	Can I Kick It?	70	251573	0	0.848	0.666	0	-6.547	1	0.2740	0.173000	0.000699	0.1290	0.7440	96.662	4	
85659	ELLEGARDEN	ジターバグ	Cakes And Ale And Everlasting Laugh	36	184173	0	0.382	0.945	2	-2.897	1	0.0670	0.000486	0.000000	0.0481	0.3140	154.960	4	
87555	Terno Rei	Violeta	São Paulo	43	171000	0	0.657	0.436	4	-9.036	0	0.0306	0.267000	0.000258	0.1110	0.2650	122.000	4	
7483	The Infamous Stringdusters	Silver Sky	Rockets	23	183686	0	0.526	0.576	4	-6.392	1	0.0286	0.233000	0.000056	0.1020	0.4240	119.924	3	
59809	From The Vastland	Mar-Tiya-Khvara	Mar-Tiya-Khvara	3	375500	0	0.273	0.923	8	-3.873	1	0.0955	0.000069	0.003300	0.0877	0.0416	119.302	4	
27957	Sub Focus;Alice Gold	Torus	Out The Blue	48	277840	0	0.423	0.912	8	-5.271	0	0.0478	0.003200	0.003430	0.2090	0.2610	174.021	4	c
70809	Joker Xue	渡	像風一樣	47	255111	0	0.514	0.418	2	-9.053	1	0.0619	0.355000	0.000000	0.0604	0.1470	117.705	4	r
32031	DJ Snake;Selena Gomez;Ozuna;Cardi B	Carte Blanche	Taki Taki (feat. Selena Gomez, Ozuna & Cardi B)	73	212500	1	0.842	0.801	8	-4.167	0	0.2280	0.157000	0.000005	0.0642	0.6170	95.881	4	
61219	Nogizaka46	帰り道は遠回りしたくなる	帰り道は遠回りしたくなる	26	269706	0	0.510	0.857	1	-2.715	1	0.0403	0.457000	0.000000	0.0794	0.6930	138.064	4	

	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	tra
96655	Tim Maia	Sufocante	Bons Momentos	38	292322	0	0.583	0.360	11	-13.415	0	0.0330	0.601000	0.000054	0.1140	0.2230	126.218	4	
56817	Nikhil D'Souza	Boss	Har Kisi Ko	39	304025	0	0.463	0.788	3	-6.523	1	0.0518	0.336000	0.000000	0.3370	0.3820	179.968	4	
85260	Joan Jett & the Blackhearts	Bad Reputation (Expanded Edition)	Bad Reputation	68	169186	0	0.378	0.974	6	-4.055	1	0.1940	0.001920	0.013900	0.0588	0.8240	203.715	4	
103988	Anita Baker	Jazz Ballads Classics	Body and Soul	42	342000	0	0.532	0.432	8	-10.481	0	0.0387	0.550000	0.000000	0.0754	0.2190	109.112	3	
43956	JADED;MIRAMAR	Overtime (Remixes)	Overtime (MIRAMAR Remix)	39	212957	0	0.773	0.897	2	-4.855	1	0.1390	0.003010	0.612000	0.1180	0.3720	123.978	4	

Additionally, we will check for class imbalance in these categorical features.

In [19]:
```
# Visualize the class imbalance in the categorical features
fig, axs = plt.subplots(1,5)
features = ['popularity','explicit','key', 'mode', 'time_signature']
col = 0
for i in features:
  val_count = df_raw[i].value_counts().rename_axis(i).reset_index(name='count')
  axs[col].bar(val_count[i], val_count['count'])
  axs[col].set_xlabel(i, fontsize=16)
  axs[col].set_xticks(val_count[i])
  axs[col].set_yticks(np.arange(0,106811,20000))
  axs[col].tick_params(axis='x', which='major', labelsize=12)

  if i=='popularity':
    axs[col].set_xticks(np.arange(0,101,20))
    axs[col].set_yticks(np.arange(0, 10001, 2000))
    axs[col].set_ylim(0, 10000)

  if i == 'key':
    axs[col].set_yticks(np.arange(0, 15001, 3000))
    axs[col].set_ylim(0, 15000)

  col += 1


axs[0].set_ylabel('Observations per class', fontsize=16)
plt.suptitle("Number of observations per class for each categorical feature", fontsize=20)
fig.set_figwidth(30)
fig.set_figheight(10)
fig.show()
```

```
/var/folders/h8/frp0f1bd0v32l04p93kbg3f00000gn/T/ipykernel_12674/2397168134.py:29: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```



Summary of class imbalances:

`popularity` : On a scale from 0 to 100, majority of the tracks were labeled as 0.

`explicit` : The non-explicit (`0`) class significantly dominates explicit (`1`) class.

`key` : Relatively balanced.

`mode` : The major scale (`1`) class somewhat dominates the minor scale (`0`) class.

`time_signature` : Tracks with time signature 4/4 (`4`) significantly outnumbers rest, followed by the time signature 3/4 (`3`).

Examine the reponse variable `track_genre` :

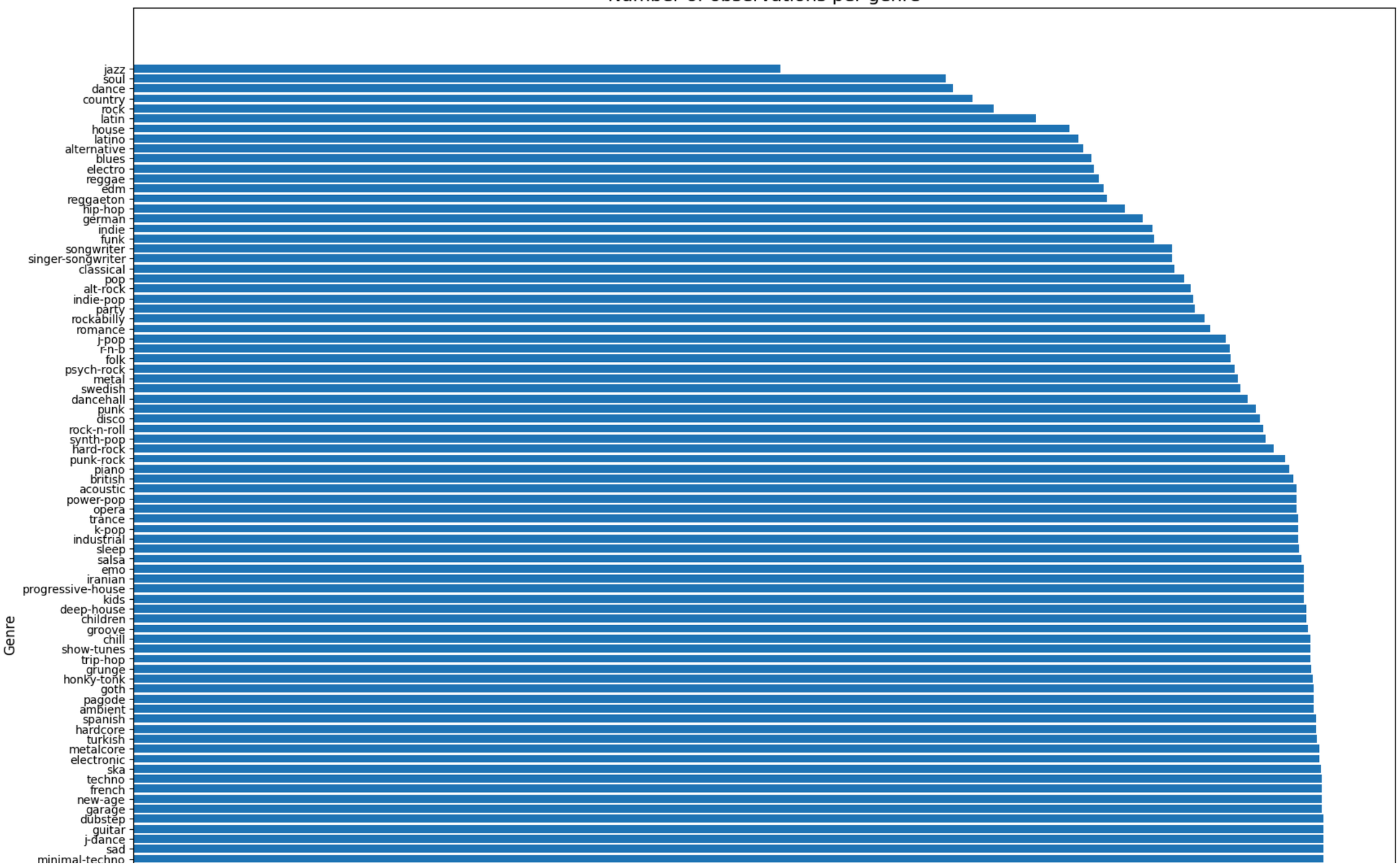
In [ ]:
```
df_raw.track_genre.nunique()
```

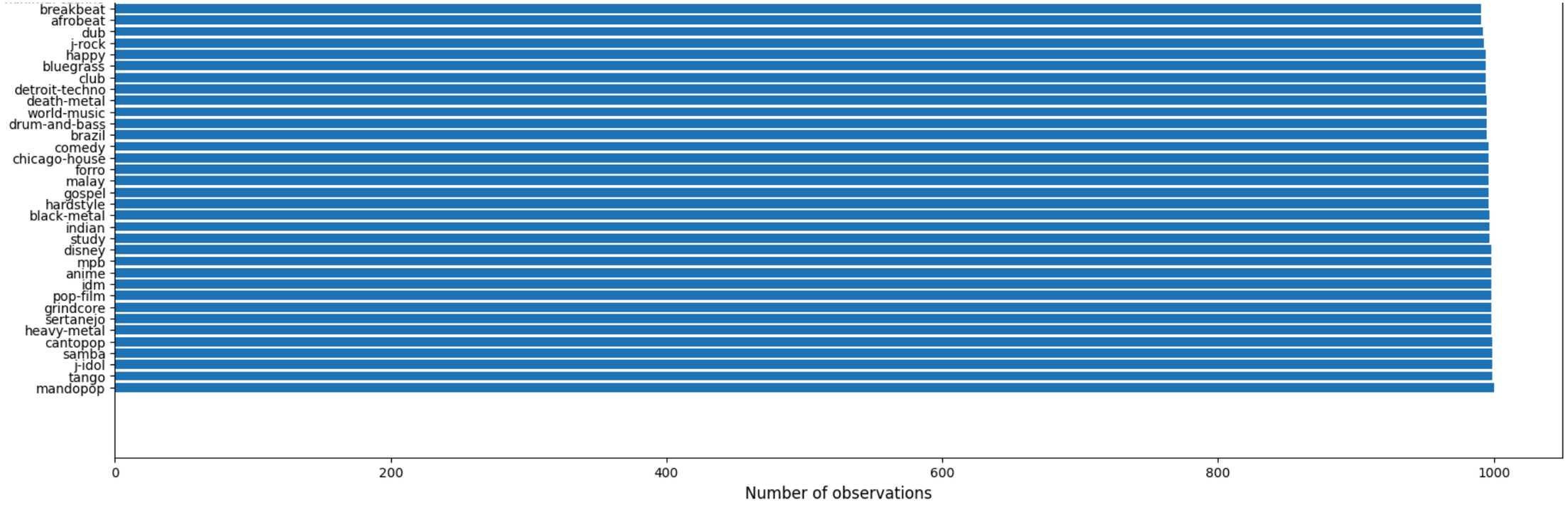
Out[ ]:
```
114
```

In [ ]:
```
df_raw.track_genre.unique()
```

Out[ ]:
```
array(['acoustic', 'afrobeat', 'alt-rock', 'alternative', 'ambient',
       'anime', 'black-metal', 'bluegrass', 'blues', 'brazil',
       'breakbeat', 'british', 'cantopop', 'chicago-house', 'children',
       'chill', 'classical', 'club', 'comedy', 'country', 'dance',
       'dancehall', 'death-metal', 'deep-house', 'detroit-techno',
       'disco', 'disney', 'drum-and-bass', 'dub', 'dubstep', 'edm',
       'electro', 'electronic', 'emo', 'folk', 'forro', 'french', 'funk',
       'garage', 'german', 'gospel', 'goth', 'grindcore', 'groove',
       'grunge', 'guitar', 'happy', 'hard-rock', 'hardcore', 'hardstyle',
       'heavy-metal', 'hip-hop', 'honky-tonk', 'house', 'idm', 'indian',
       'indie-pop', 'indie', 'industrial', 'iranian', 'j-dance', 'j-idol',
       'j-pop', 'j-rock', 'jazz', 'k-pop', 'kids', 'latin', 'latino',
       'malay', 'mandopop', 'metal', 'metalcore', 'minimal-techno', 'mpb',
       'new-age', 'opera', 'pagode', 'party', 'piano', 'pop-film', 'pop',
       'power-pop', 'progressive-house', 'psych-rock', 'punk-rock',
       'punk', 'r-n-b', 'reggae', 'reggaeton', 'rock-n-roll', 'rock',
       'rockabilly', 'romance', 'sad', 'salsa', 'samba', 'sertanejo',
       'show-tunes', 'singer-songwriter', 'ska', 'sleep', 'songwriter',
       'soul', 'spanish', 'study', 'swedish', 'synth-pop', 'tango',
       'techno', 'trance', 'trip-hop', 'turkish', 'world-music'],
      dtype=object)
```

In [ ]:
```
# Visualize any class imbalance in the response variable 'track_genre
from matplotlib import container

fig, ax = plt.subplots()

val_count = df_raw['track_genre'].value_counts().rename_axis('track_genre').reset_index(name='count')
ax.barh(val_count['track_genre'], val_count['count'])
ax.set_ylabel('Genre', fontsize=12)
ax.set_xlabel('Number of observations', fontsize=12)
ax.tick_params(axis='both', which='major', labelsize=10)

ax.set_title("Number of observations per genre", fontsize=16)
fig.set_figwidth(20)
fig.set_figheight(20)
fig.show()
```


Number of observations per genre
Genre
jazz
soul
dance
country
rock
latin
house
latino
alternative
blues
electro
reggae
edm
reggaeton
hip-hop
german
indie
funk
songwriter
singer-songwriter
classical
pop
alt-rock
indie-pop
party
rockabilly
romance
j-pop
r-n-b
folk
psych-rock
metal
swedish
dancehall
punk
disco
rock-n-roll
synth-pop
hard-rock
punk-rock
piano
british
acoustic
power-pop
opera
trance
k-pop
industrial
sleep
salsa
emo
iranian
progressive-house
kids
deep-house
children
groove
chill
show-tunes
trip-hop
grunge
honky-tonk
goth
pagode
ambient
spanish
hardcore
turkish
metalcore
electronic
ska
techno
french
new-age
garage
dubstep
guitar
j-dance
sad
minimal-techno



In [ ]:
```
pd.set_option('display.max_rows', None)
df_raw['track_genre'].value_counts()
```

Out[ ]:

track_genre	count
mandopop	1000
tango	999
j-idol	999
samba	999
cantopop	999
heavy-metal	998
sertanejo	998
grindcore	998
pop-film	998
idm	998
anime	998
mpb	998
disney	998
study	997
indian	997
black-metal	997
hardstyle	996
gospel	996
malay	996
forro	996
chicago-house	996
comedy	996
brazil	995
drum-and-bass	995
world-music	995
death-metal	995
detroit-techno	994
club	994
bluegrass	994
happy	994
j-rock	993

track_genre	count
dub	992
afrobeat	991
breakbeat	991
minimal-techno	990
sad	990
j-dance	990
guitar	990
dubstep	990
garage	989
new-age	989
french	989
techno	989
ska	988
electronic	987
metalcore	987
turkish	985
hardcore	984
spanish	984
ambient	982
pagode	982
goth	982
honky-tonk	981
grunge	980
trip-hop	979
show-tunes	979
chill	979
groove	977
children	976
deep-house	976
kids	974
progressive-house	974

track_genre	count
iranian	974
emo	974
salsa	972
sleep	970
industrial	969
k-pop	969
trance	969
opera	968
power-pop	968
acoustic	968
british	965
piano	962
punk-rock	958
hard-rock	949
synth-pop	942
rock-n-roll	940
disco	937
punk	934
dancehall	927
swedish	921
metal	919
psych-rock	916
folk	913
r-n-b	912
j-pop	909
romance	896
rockabilly	891
party	883
indie-pop	882
alt-rock	880
pop	874

track_genre	count
classical	866
singer-songwriter	864
songwriter	864
funk	849
indie	848
german	840
hip-hop	825
reggaeton	810
edm	807
reggae	803
electro	799
blues	797
alternative	790
latino	786
house	779
latin	751
rock	716
country	698
dance	682
soul	676
jazz	538

dtype: int64

The data is somewhat imbalanced in terms of the response variable `track_genre`. However, the ratio between the smallest class ("jazz") and largest class ("mandopop") is 538:1000, which is relatively acceptable rate.

In preparation for EDA, we'll make a new copy of the data set with the columns `album_name`, `track_name`, and `artists` dropped and will keep only observations belonging to the top 20 genres by count in the data in order to preserve the amount of data we have to work with and allow for more robust and effective classification of track genre. With 114 classes and 15 features, most models are unable to accurately classify observations.

In [20]:
```
# Isolate the numerical data (i.e. drop the album, track, and artist names) and the response variable 'track_genre'
df_raw = df_raw.drop(columns=['album_name','track_name','artists'])
top20 = df_raw['track_genre'].value_counts(ascending=False)[:20].index
df_raw = df_raw[df_raw['track_genre'].isin(top20)]
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19955 entries, 5000 to 108999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   popularity        19955 non-null  int64
 1   duration_ms       19955 non-null  int64
 2   explicit          19955 non-null  int64
 3   danceability      19955 non-null  float64
 4   energy            19955 non-null  float64
 5   key               19955 non-null  int64
 6   loudness          19955 non-null  float64
 7   mode              19955 non-null  int64
 8   speechiness       19955 non-null  float64
 9   acousticness      19955 non-null  float64
 10  instrumentalness  19955 non-null  float64
 11  liveness          19955 non-null  float64
 12  valence           19955 non-null  float64
 13  tempo             19955 non-null  float64
 14  time_signature    19955 non-null  int64
 15  track_genre       19955 non-null  object
dtypes: float64(9), int64(6), object(1)
memory usage: 2.6+ MB
```

In [21]:

```
# Create a standardized version of the data for modeling purposes after EDA
ss = preprocessing.StandardScaler()
df = df_raw.copy()
num_cols = df.drop(columns='track_genre').columns
df[num_cols] = ss.fit_transform(df.drop(columns='track_genre'))
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19955 entries, 5000 to 108999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   popularity        19955 non-null  float64
 1   duration_ms       19955 non-null  float64
 2   explicit          19955 non-null  float64
 3   danceability      19955 non-null  float64
 4   energy            19955 non-null  float64
 5   key               19955 non-null  float64
 6   loudness          19955 non-null  float64
 7   mode              19955 non-null  float64
 8   speechiness       19955 non-null  float64
 9   acousticness      19955 non-null  float64
 10  instrumentalness  19955 non-null  float64
 11  liveness          19955 non-null  float64
 12  valence           19955 non-null  float64
 13  tempo             19955 non-null  float64
 14  time_signature    19955 non-null  float64
 15  track_genre       19955 non-null  object
dtypes: float64(15), object(1)
memory usage: 2.6+ MB
```

In [22]:
```
# Create a scaled version of the data so that all values are between -1 and 1
mm = preprocessing.MinMaxScaler()
df_mm = df_raw.copy()
num_cols = df_mm.drop(columns='track_genre').columns
df_mm[num_cols] = mm.fit_transform(df_mm.drop(columns='track_genre'))
df_mm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19955 entries, 5000 to 108999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   popularity        19955 non-null  float64
 1   duration_ms       19955 non-null  float64
 2   explicit          19955 non-null  float64
 3   danceability      19955 non-null  float64
 4   energy            19955 non-null  float64
 5   key               19955 non-null  float64
 6   loudness          19955 non-null  float64
 7   mode              19955 non-null  float64
 8   speechiness       19955 non-null  float64
 9   acousticness      19955 non-null  float64
 10  instrumentalness  19955 non-null  float64
 11  liveness          19955 non-null  float64
 12  valence           19955 non-null  float64
 13  tempo             19955 non-null  float64
 14  time_signature    19955 non-null  float64
 15  track_genre       19955 non-null  object 
dtypes: float64(15), object(1)
memory usage: 2.6+ MB
```

II. Exploratory Data Analysis

In [24]:
```
df.sample(10)
```

Out[24]:

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
54384	-1.412613	0.316920	-0.181294	0.976702	0.599810	1.061682	-0.819268	-1.349109	-0.442136	-0.690465	2.234677	-0.249663	-0.256109	1.196783	0.241705	idm
55210	1.650196	1.244444	-0.181294	-0.971826	0.111061	1.625871	0.179987	-1.349109	-0.361902	0.790679	-0.542043	0.218154	-0.180384	-1.195361	0.241705	indian
74026	1.139728	0.059853	-0.181294	-0.277663	-0.897968	-0.912980	-0.103597	0.741230	-0.540667	0.672792	-0.542049	-0.593647	-0.905745	-1.062472	-2.113381	mpb
12824	1.139728	0.147428	-0.181294	0.696601	0.970313	-0.912980	0.648232	-1.349109	0.374272	-0.636056	-0.542049	-0.753255	0.564904	-1.046021	0.241705	cantopop
69101	0.437834	0.440066	-0.181294	0.501748	0.682582	0.779587	0.606388	0.741230	-0.292930	-0.563510	-0.542049	-0.737661	0.796063	-0.371189	-2.113381	malay
50085	0.054983	-0.497352	-0.181294	-1.726880	1.135857	1.061682	0.993446	0.741230	-0.108535	-1.130767	-0.541642	0.264018	0.843889	-1.036955	0.241705	heavy-metal
42170	-1.157379	-0.741845	5.515904	-1.288461	1.399939	-1.195074	1.223362	0.741230	0.768399	-1.128732	-0.173831	0.135598	-0.036906	-0.397892	0.241705	grindcore
97389	0.884494	-0.646145	-0.181294	1.518636	0.934839	-0.630885	0.855406	-1.349109	0.472804	-0.046621	-0.542049	2.126113	0.871788	0.137524	0.241705	sertanejo
6034	-0.519294	-0.877322	-0.181294	-2.031337	1.151623	0.215398	0.733740	0.741230	0.080084	-1.131743	2.285986	0.470408	1.138816	1.330332	0.241705	black-metal
26933	-0.902145	-0.314015	-0.181294	-1.178857	-1.248764	0.497493	-1.004383	-1.349109	-0.613863	1.615888	1.766859	-0.368911	-1.160817	-1.410998	0.241705	disney

In [25]:
```
# Examine the descriptive statistics of the numerical data
df.describe()
```

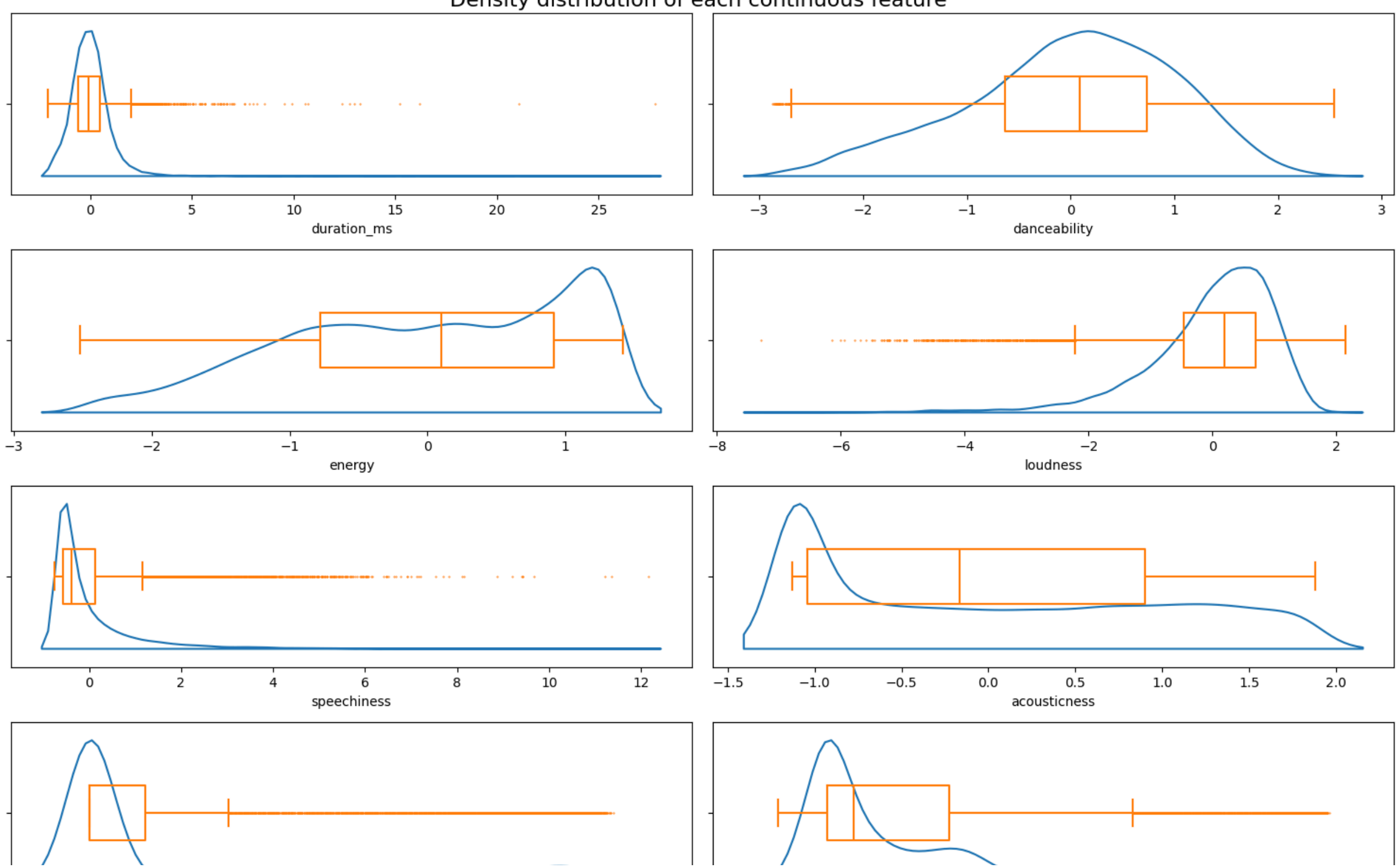
Out[25]:

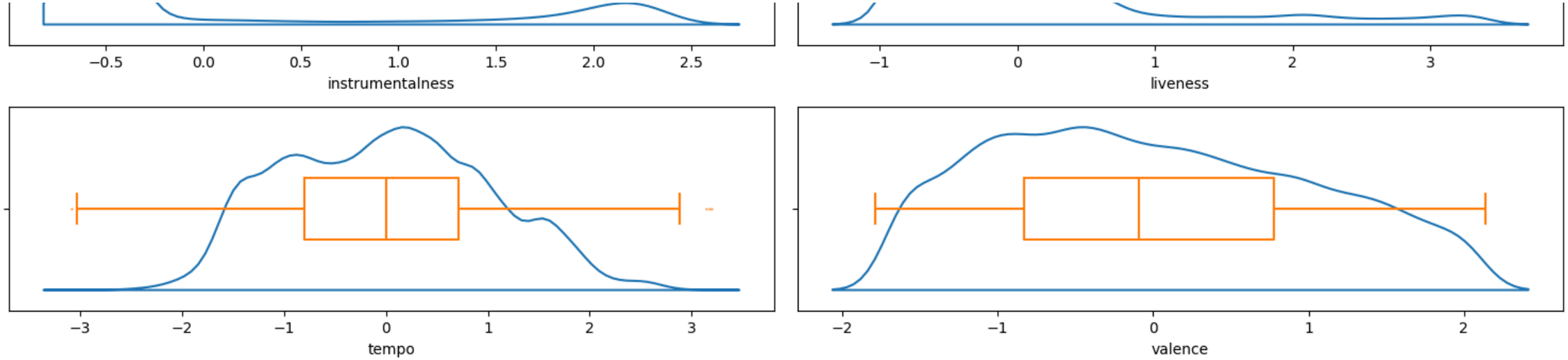
	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	tim
count	1.995500e+04	19955.000000	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	1.995500e+04	19955.000000	1.995500e+04	1.995500e+04	1
mean	2.050978e-16	0.000000	-3.988012e-17	-3.304353e-16	2.050978e-16	8.759384e-17	-4.557728e-17	8.830599e-17	-5.412302e-17	-9.115457e-17	5.127444e-17	0.000000	1.709148e-16	4.158927e-16	-
std	1.000025e+00	1.000025	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025e+00	1.000025	1.000025e+00	1.000025e+00	1
min	-2.178315e+00	-2.097496	-1.812939e-01	-2.872858e+00	-2.521597e+00	-1.477168e+00	-7.289416e+00	-1.349109e+00	-7.503996e-01	-1.131785e+00	-5.420494e-01	-1.063757	-1.786541e+00	-3.078958e+00	-9
25%	-9.021446e-01	-0.583862	-1.812939e-01	-6.338779e-01	-7.797228e-01	-9.129795e-01	-4.655263e-01	-1.349109e+00	-5.660042e-01	-1.043219e+00	-5.420494e-01	-0.657857	-8.300205e-01	-7.976322e-01	
50%	1.826001e-01	-0.062605	-1.812939e-01	8.768604e-02	9.923667e-02	-6.669615e-02	1.902211e-01	7.412297e-01	-3.830164e-01	-1.645077e-01	-5.419570e-01	-0.442294	-8.871751e-02	7.469095e-03	
75%	7.568767e-01	0.461311	-1.812939e-01	7.392248e-01	9.190733e-01	7.795872e-01	7.005375e-01	7.412297e-01	1.293495e-01	8.994976e-01	-2.221223e-01	0.337401	7.801215e-01	7.138510e-01	
max	3.436834e+00	27.786392	5.515904e+00	2.541613e+00	1.419647e+00	1.625871e+00	2.155076e+00	7.412297e-01	1.215587e+01	1.878866e+00	2.473113e+00	3.433249	2.139177e+00	3.191446e+00	2

Most of the features ranges between [0,1] while the rest vary drastically: some ranges betweem [0,100] (e.g. `popularity`), some [0,11] (e.g. `key`), or even to negative values (e.g. `loudness`). Thus, it will be important to standardize and/or mean center these features accordingly for the remainder of the project.

In [ ]:
```
# Plot the density distribution for each continuous variable
cts_cols = ['duration_ms','danceability','energy','loudness','speechiness','acousticness','instrumentalness','liveness','tempo','valence']

fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 13))
col = 0
row = 0

for i in cts_cols:
  sns.violinplot(data=df, x=i, ax=axes[row][col], inner=None, fill=False, split=True) #inner_kws=dict(box_width=15, whis_width=2)
```

```
  sns.boxplot(data=df, x=i, fill=False, width=0.3, fliersize=0.5, boxprops={'zorder': 2}, ax=axes[row][col])
  axes[row][col].set_xlabel(i, fontsize=10)
  col+=1
  if col==2:
    col=0
    row+=1

plt.suptitle('Density distribution of each continuous feature', fontsize=16)
plt.tight_layout()
```

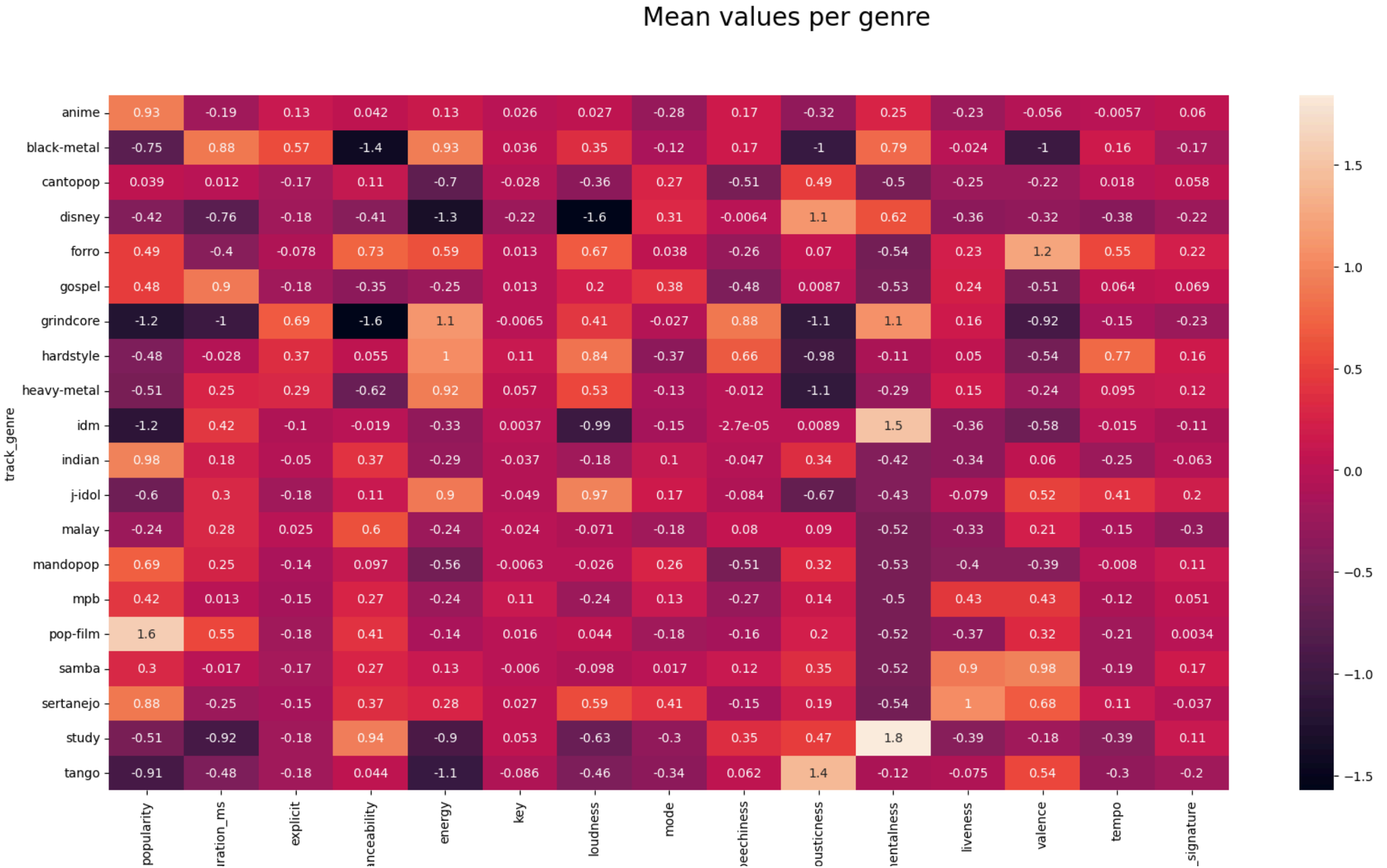

Density distribution of each continuous feature
duration_ms
danceability
energy
loudness
speechiness
acousticness



Like some of the categorical predictor variables, many of the continuous predictor variables are heavily skewed with a handful of outliers. This should be addressed depending on the assumptions made by different modeling techiques (e.g. distance-based clustering methods may be sensitive to outliers, linear regression assumes normality and homoscedasticity).

In [ ]:
```
# Examine the predictor variable means per genre
fig, ax = plt.subplots(figsize=(20, 10))
sns.heatmap(df.groupby('track_genre').mean(),annot=True)
plt.suptitle('Mean values per genre', fontsize=20)
```

Out[ ]:
```
Text(0.5, 0.98, 'Mean values per genre')
```

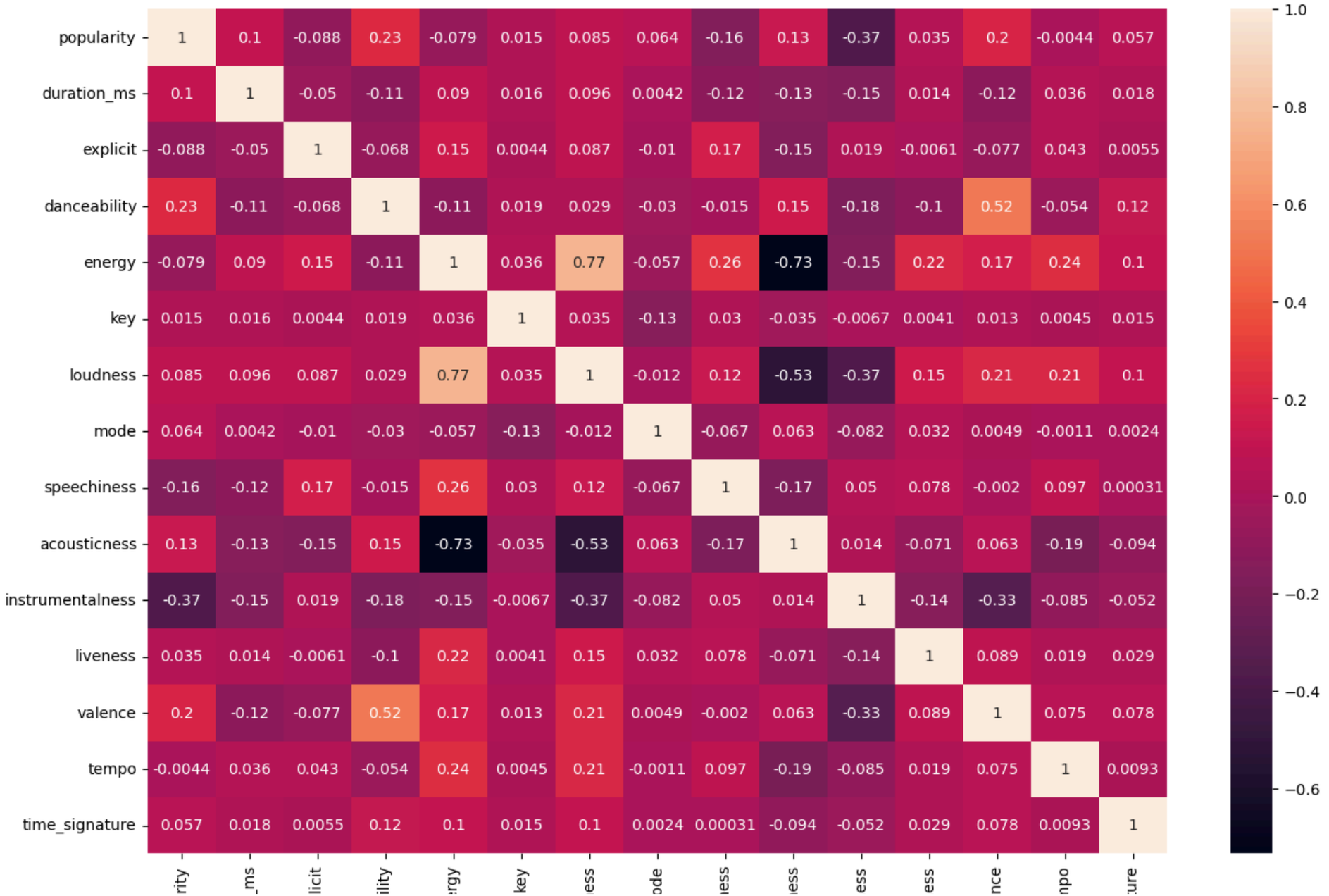

Mean values per genre
track_genre
anime 0.93 -0.19 0.13 0.042 0.13 0.026 0.027 -0.28 0.17 -0.32 0.25 -0.23 -0.056 -0.0057 0.06
black-metal -0.75 0.88 0.57 -1.4 0.93 0.036 0.35 -0.12 0.17 -1 0.79 -0.024 -1 0.16 -0.17
cantopop 0.039 0.012 -0.17 0.11 -0.7 -0.028 -0.36 0.27 -0.51 0.49 -0.5 -0.25 -0.22 0.018 0.058
disney -0.42 -0.76 -0.18 -0.41 -1.3 -0.22 -1.6 0.31 -0.0064 1.1 0.62 -0.36 -0.32 -0.38 -0.22
forro 0.49 -0.4 -0.078 0.73 0.59 0.013 0.67 0.038 -0.26 0.07 -0.54 0.23 1.2 0.55 0.22
gospel 0.48 0.9 -0.18 -0.35 -0.25 0.013 0.2 0.38 -0.48 0.0087 -0.53 0.24 -0.51 0.064 0.069
grindcore -1.2 -1 0.69 -1.6 1.1 -0.0065 0.41 -0.027 0.88 -1.1 1.1 0.16 -0.92 -0.15 -0.23
hardstyle -0.48 -0.028 0.37 0.055 1 0.11 0.84 -0.37 0.66 -0.98 -0.11 0.05 -0.54 0.77 0.16
heavy-metal -0.51 0.25 0.29 -0.62 0.92 0.057 0.53 -0.13 -0.012 -1.1 -0.29 0.15 -0.24 0.095 0.12
idm -1.2 0.42 -0.1 -0.019 -0.33 0.0037 -0.99 -0.15 -2.7e-05 0.0089 1.5 -0.36 -0.58 -0.015 -0.11
indian 0.98 0.18 -0.05 0.37 -0.29 -0.037 -0.18 0.1 -0.047 0.34 -0.42 -0.34 0.06 -0.25 -0.063
j-idol -0.6 0.3 -0.18 0.11 0.9 -0.049 0.97 0.17 -0.084 -0.67 -0.43 -0.079 0.52 0.41 0.2
malay -0.24 0.28 0.025 0.6 -0.24 -0.024 -0.071 -0.18 0.08 0.09 -0.52 -0.33 0.21 -0.15 -0.3
mandopop 0.69 0.25 -0.14 0.097 -0.56 -0.0063 -0.026 0.26 -0.51 0.32 -0.53 -0.4 -0.39 -0.008 0.11
mpb 0.42 0.013 -0.15 0.27 -0.24 0.11 -0.24 0.13 -0.27 0.14 -0.5 0.43 0.43 -0.12 0.051
pop-film 1.6 0.55 -0.18 0.41 -0.14 0.016 0.044 -0.18 -0.16 0.2 -0.52 -0.37 0.32 -0.21 0.0034
samba 0.3 -0.017 -0.17 0.27 0.13 -0.006 -0.098 0.017 0.12 0.35 -0.52 0.9 0.98 -0.19 0.17
sertanejo 0.88 -0.25 -0.15 0.37 0.28 0.027 0.59 0.41 -0.15 0.19 -0.54 1 0.68 0.11 -0.037
study -0.51 -0.92 -0.18 0.94 -0.9 0.053 -0.63 -0.3 0.35 0.47 1.8 -0.39 -0.18 -0.39 0.11
tango -0.91 -0.48 -0.18 0.044 -1.1 -0.086 -0.46 -0.34 0.062 1.4 -0.12 -0.075 0.54 -0.3 -0.2
popularity
uration_ms
explicit
anceability
energy
key
loudness
mode
peechiness
ousticness
entalness
liveness
valence
tempo
_signature
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

As illustrated by the mean values per feature per genre, some of the class imbalance in the predictor variables are associated with specific classes in the response variable `track_genre`. For instance, `explicit`-ness is expecially present in genres like grindcore, black-metal, an hardstyle while genres like sertanejo, gospel, and disney lean toward being on a major scale.

In [ ]:
```
# Plot the correlation matrix between predictor variables
fig, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(df.drop(columns=['track_genre']).corr(),annot=True)
plt.suptitle("Correlation Matrix between Predictor Variables", fontsize=20)
```

Out[ ]:
```
Text(0.5, 0.98, 'Correlation Matrix between Predictor Variables')
```

Correlation Matrix between Predictor Variables

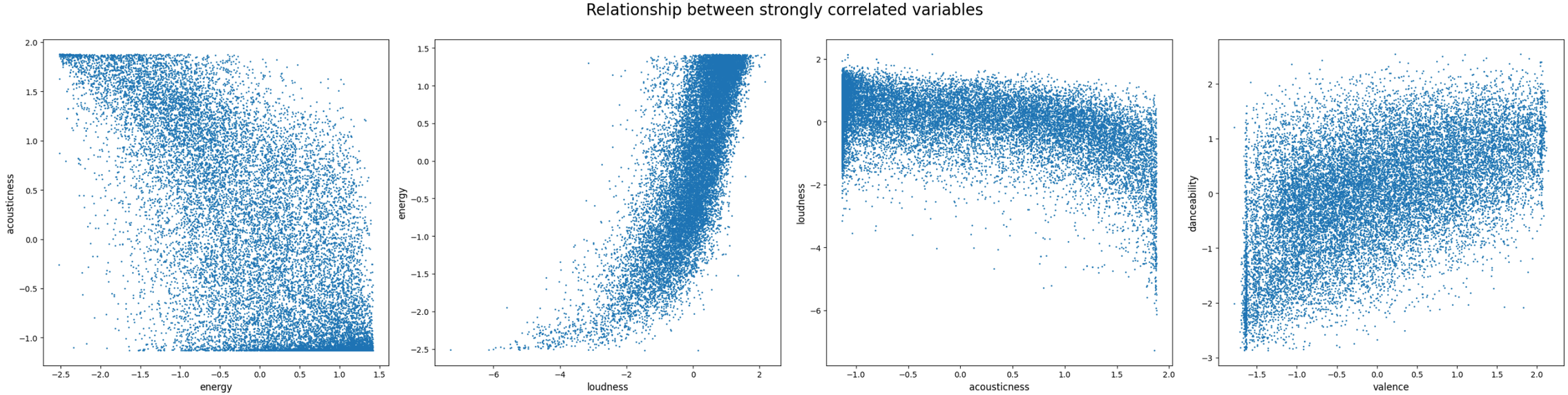


It is important to note the following highly-correlated features since modeling techinques (e.g. logistic regression) can make assumptions about the absence of multicollinearity.

- valence and danceability: 0.52
- energy and loudness: -0.73
- energy and acousticness: -0.73
- loudness and acousticness: -0.53

The scatterplots below illutrates how their relationships are non-linear.

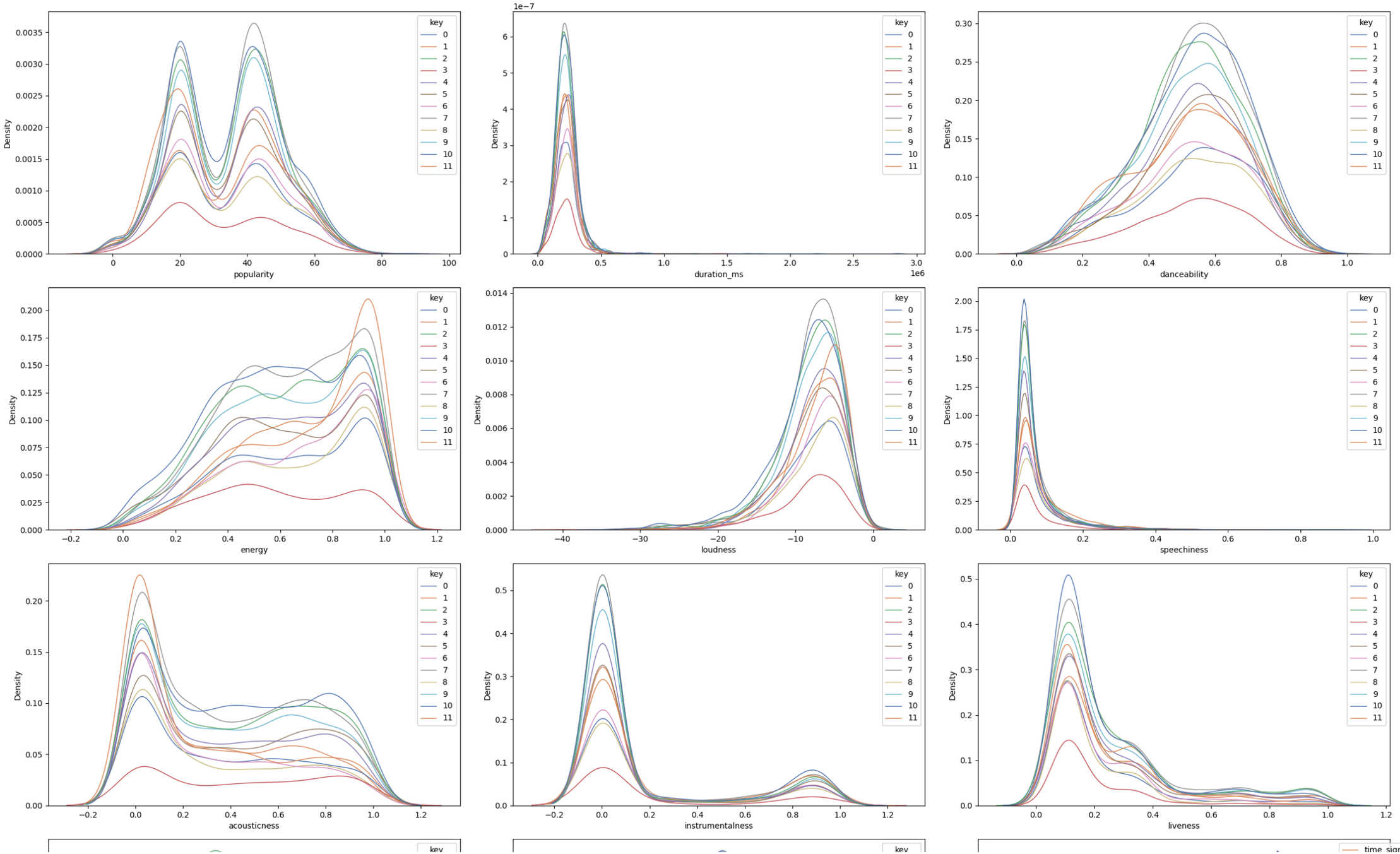
In [ ]:
```
fig, axs = plt.subplots(1,4, figsize=(28,7))
axs[0].scatter(df['energy'], df['acousticness'], s=1)
axs[0].set_xlabel('energy', fontsize=12)
axs[0].set_ylabel('acousticness', fontsize=12)
axs[1].scatter(df['loudness'], df['energy'], s=1)
axs[1].set_xlabel('loudness', fontsize=12)
axs[1].set_ylabel('energy', fontsize=12)
axs[2].scatter(df['acousticness'], df['loudness'], s=1)
axs[2].set_xlabel('acousticness', fontsize=12)
axs[2].set_ylabel('loudness', fontsize=12)
axs[3].scatter(df['valence'], df['danceability'], s=1)
axs[3].set_xlabel('valence', fontsize=12)
axs[3].set_ylabel('danceability', fontsize=12)

plt.suptitle("Relationship between strongly correlated variables", y=1.01, fontsize=20)
plt.tight_layout()
plt.show()
```

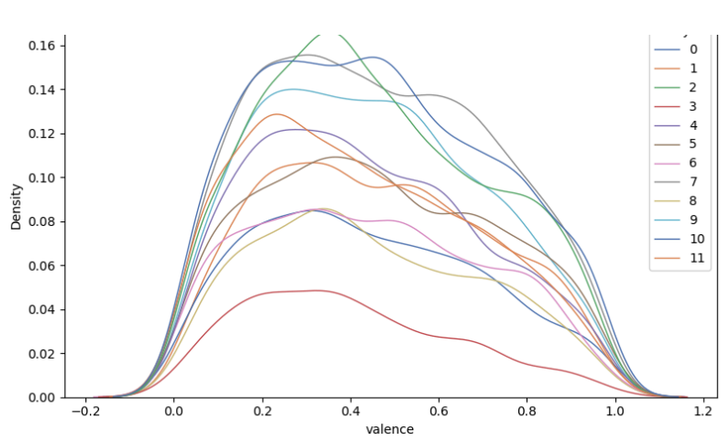


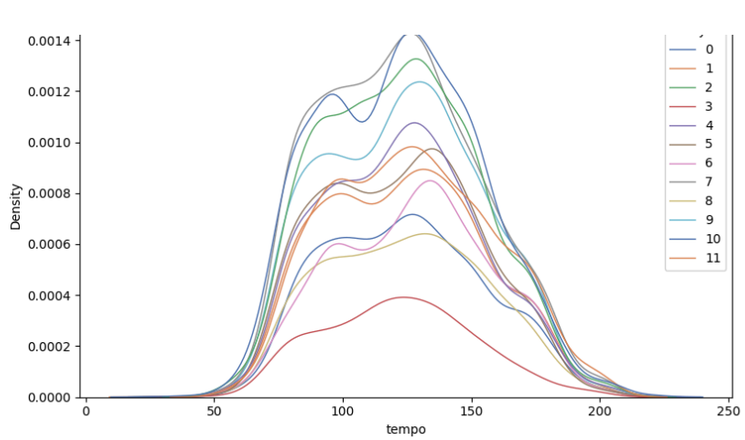
We can further explore the relationship between each numerical feature's distribution and each categorical feature via the density plots below.

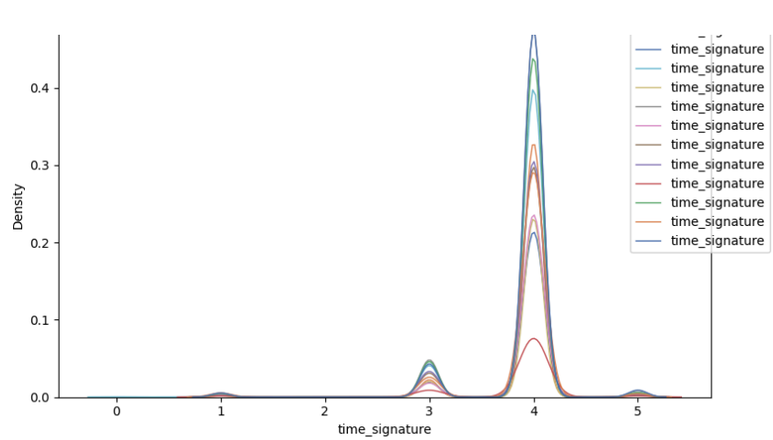
In [ ]:
```
# Create a figure and axes
fig, ax = plt.subplots(4,3, figsize=(25,20))
i=0
j=0
# Loop through the columns and create a KDE plot for each
for col in df_raw.drop(columns=['track_genre','mode','explicit','key']).columns:
    sns.kdeplot(df_raw[[col,'key']], x=col, ax=ax[i][j], label=col, lw=1, palette='deep', hue='key')
    j+=1
    if j==3:
      j=0
      i+=1

# Add a legend and title
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1))
plt.suptitle("Density Plot for Each Numerical Feature Categorized by 'Key'", y=1.01, fontsize=20)
plt.tight_layout()
plt.show()
```

Density Plot for Each Numerical Feature Categorized by 'Key'





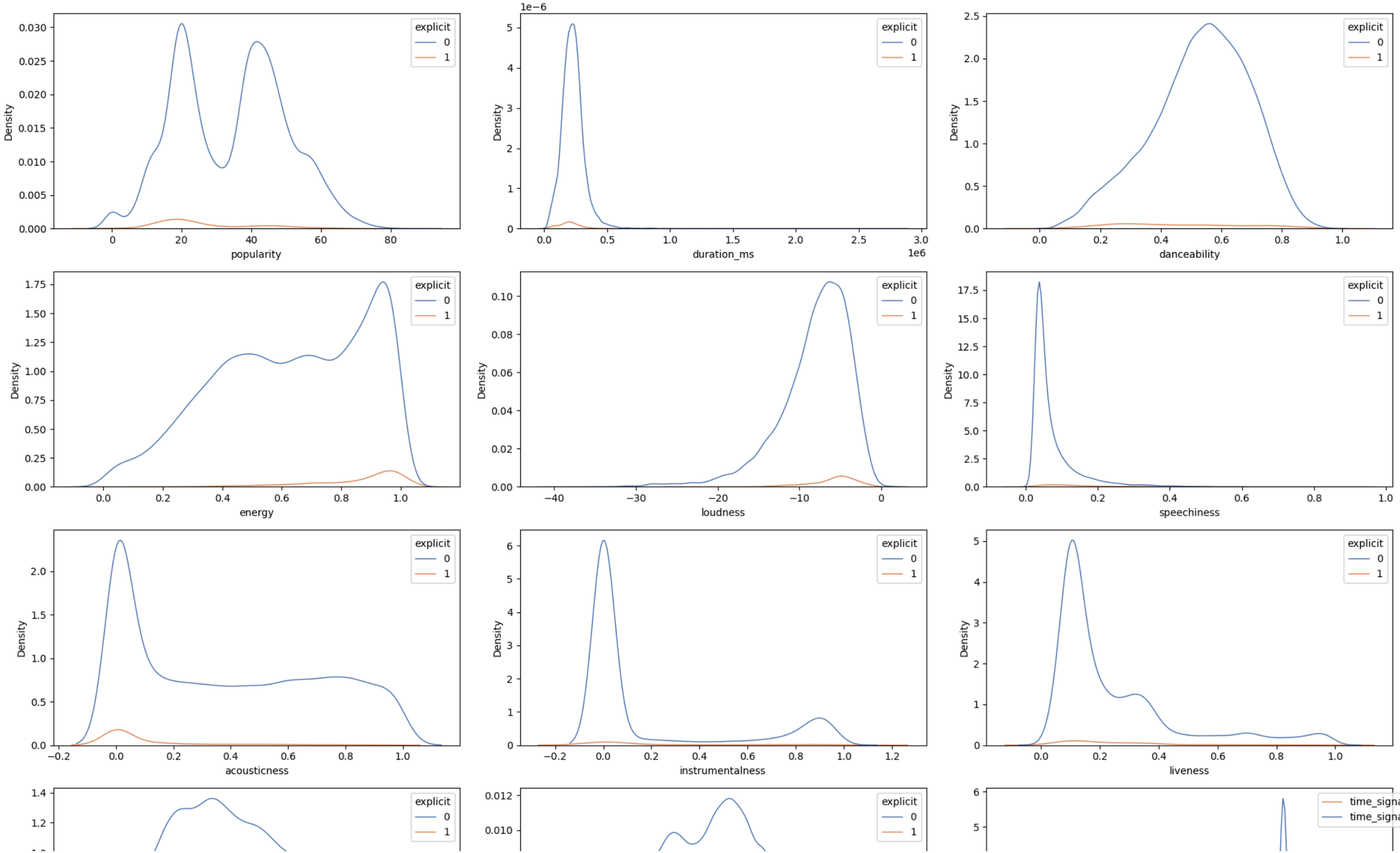


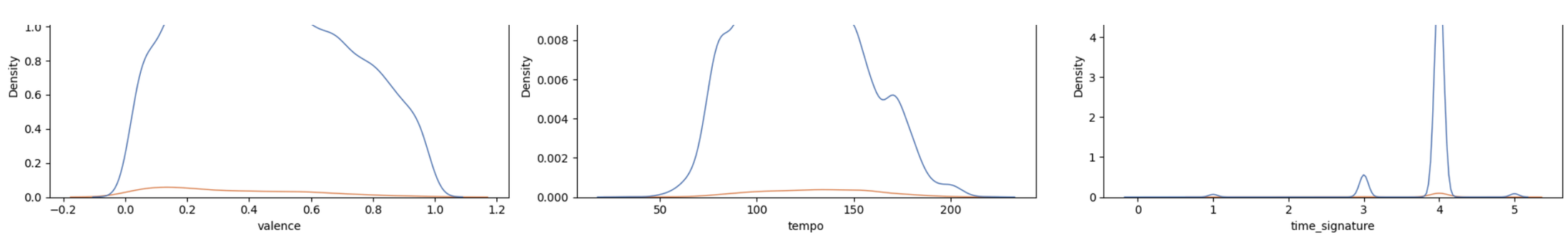


In [ ]:

```
# Create a figure and axes
fig, ax = plt.subplots(4,3, figsize=(20,15))
i=0
j=0
# Loop through the columns and create a KDE plot for each
for col in df_raw.drop(columns=['track_genre','mode','explicit','key']).columns:
    sns.kdeplot(df_raw[[col,'explicit']], x=col, ax=ax[i][j], label=col, lw=1, palette='deep', hue='explicit')
    j+=1
    if j==3:
      j=0
      i+=1

# Add a legend and title
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1))
plt.suptitle("Density Plot for Each Numerical Feature Categorized by 'Explicit'", y=1.01, fontsize=16)
plt.tight_layout()
plt.show()
```

Density Plot for Each Numerical Feature Categorized by 'Explicit'

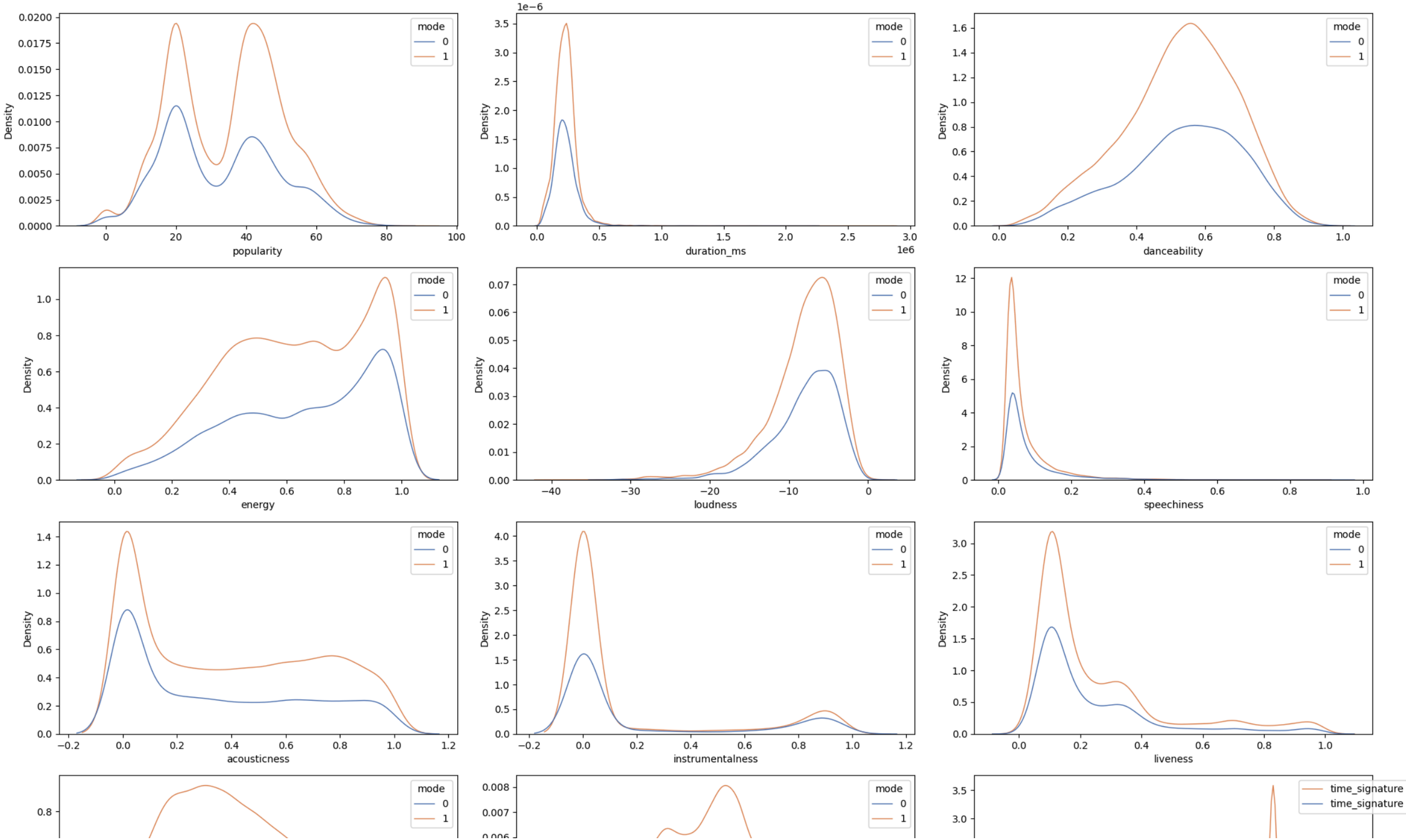


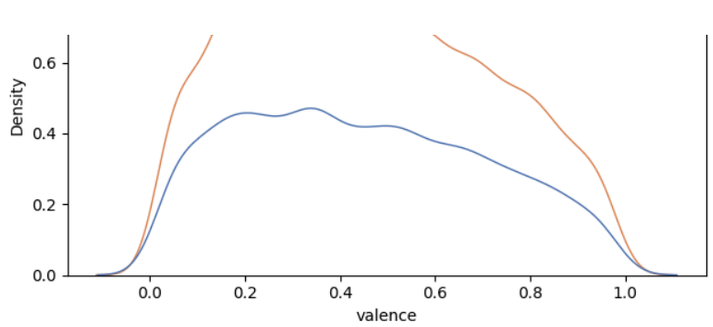


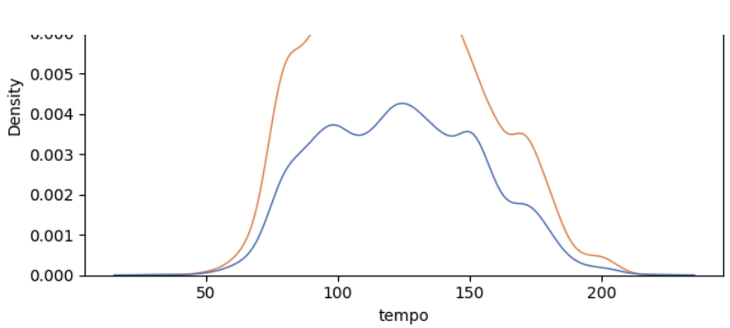
In [ ]:

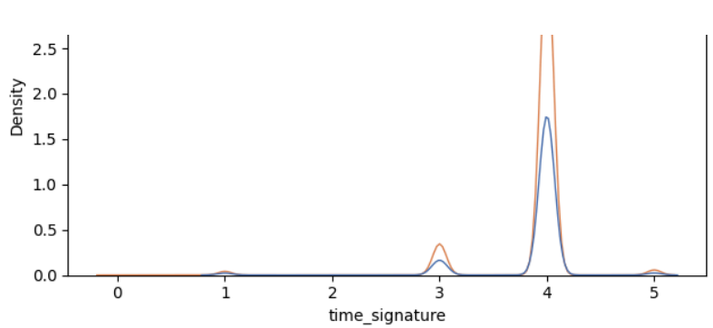
```
# Create a figure and axes
fig, ax = plt.subplots(4,3, figsize=(20,15))
i=0
j=0
# Loop through the columns and create a KDE plot for each
for col in df_raw.drop(columns=['track_genre','mode','explicit','key']).columns:
    sns.kdeplot(df_raw[[col,'mode']], x=col, ax=ax[i][j], label=col, lw=1, palette='deep', hue='mode')
    j+=1
    if j==3:
      j=0
      i+=1

# Add a legend and title
plt.legend(loc='upper right',bbox_to_anchor=(1.1, 1))
plt.suptitle("Density Plot for Each Numerical Feature Categorized by 'Modality'", y=1.01, fontsize=16)
plt.tight_layout()
plt.show()
```

Density Plot for Each Numerical Feature Categorized by 'Modality'



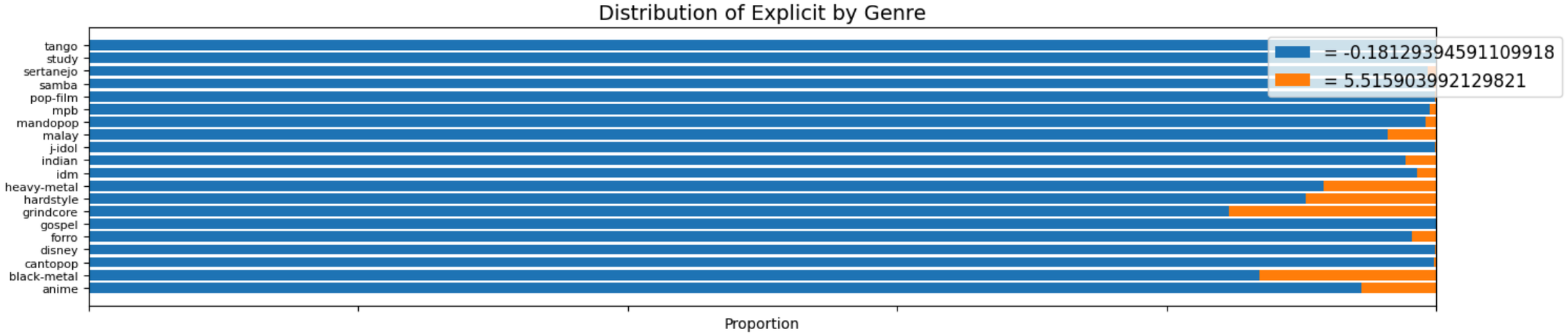


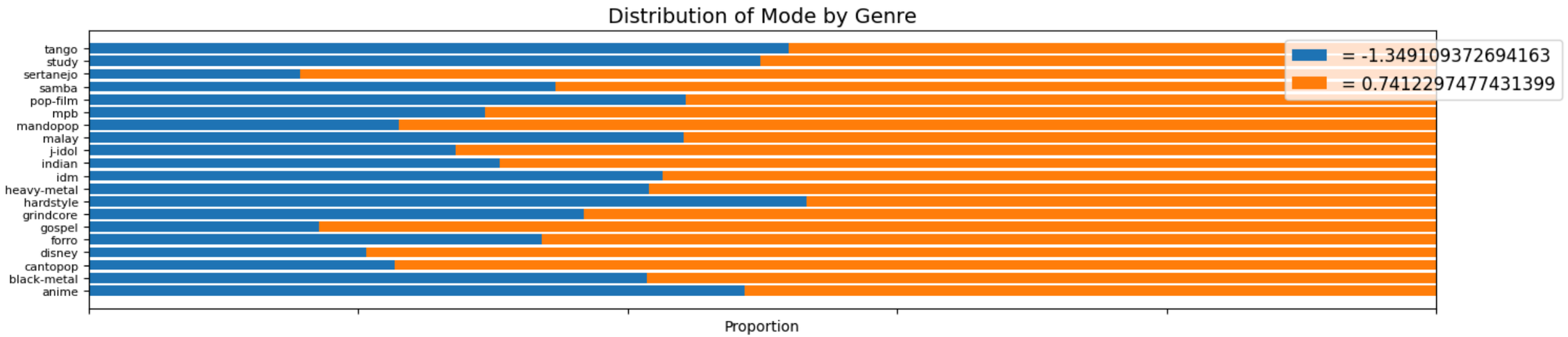


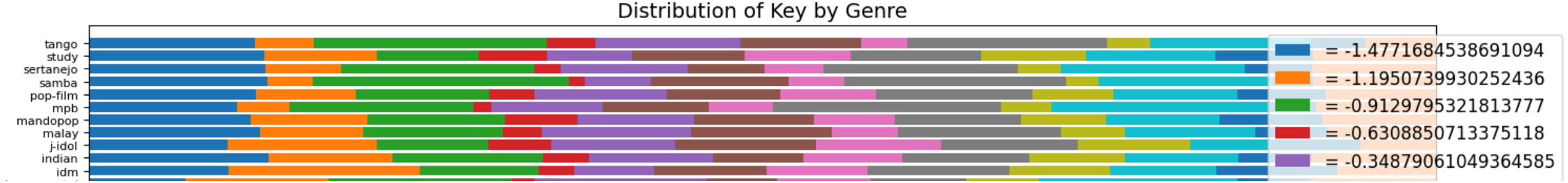


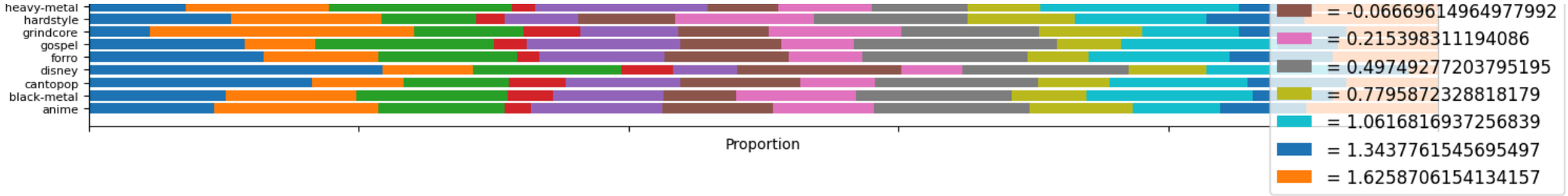
We can also visualize the class distribution per genre for every categorical feature.

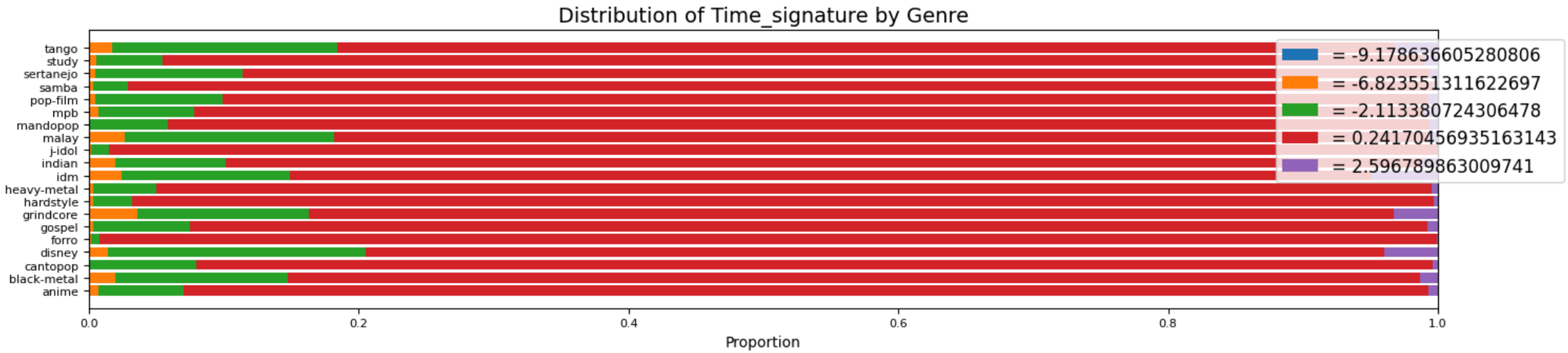
In [ ]:
```
# calculate the class proportions for each categorical feature
proportions = {}
for col in ['explicit', 'mode', 'key', 'time_signature']:
    proportions[col] = (df.groupby('track_genre')[col].value_counts(normalize=True).unstack(fill_value=0))


fig, axes = plt.subplots(4, 1, figsize=(15, 15), sharex=True)
genres = df['track_genre'].unique()  # 20 unique genres

# plot each category in a separate subplot
for ax, (col, prop_df) in zip(axes, proportions.items()):
    left_pos = np.zeros(20) # bars intialized at leftmost position

    # plot each class as a stacked bar
    for col_class in prop_df.columns:
        ax.barh(genres,
            prop_df[col_class].reindex(genres, fill_value=0),  # index replaced with genres
            left=left_pos,  # stack this class's bars on top of previous bars
            label=f"= {col_class}") # legend label
        left_pos += prop_df[col_class].reindex(genres, fill_value=0)

    ax.set_title(f"Distribution of {col.capitalize()} by Genre", fontsize=14)
    ax.set_xlabel("Proportion", fontsize=10)
    ax.set_xlim(0, 1)
    ax.tick_params(axis='both', which='major', labelsize=8)
    ax.legend(loc='upper right',bbox_to_anchor=(1.1, 1),fontsize=12)

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


Distribution of Explicit by Genre
tango
study
sertanejo
samba
pop-film
mpb
mandopop
malay
j-idol
indian
idm
heavy-metal
hardstyle
grindcore
gospel
forro
disney
cantopop
black-metal
anime
= -0.18129394591109918
= 5.515903992129821
Proportion


Distribution of Mode by Genre
tango
study
sertanejo
samba
pop-film
mpb
mandopop
malay
j-idol
indian
idm
heavy-metal
hardstyle
grindcore
gospel
forro
disney
cantopop
black-metal
anime
= -1.349109372694163
= 0.7412297477431399
Proportion


Distribution of Key by Genre
tango
study
sertanejo
samba
pop-film
mpb
mandopop
malay
j-idol
indian
idm
= -1.4771684538691094
= -1.1950739930252436
= -0.9129795321813777
= -0.6308850713375118
= -0.34879061049364585





iii. Regression Analysis

Since our main problem to address involves the classification of track genres, there is not much use in performing linear regression. However, in exploring the data and conducting some preliminary analyses before we settled on the primary topic of our project, we did use linear regression to predict the loudness of a song from its energy, seeing as they were fairly correlated from EDA. We also compared the fit to a ridge regression model, though the results indicate that ridge regression had minimal effect on the regression. The original model was simple enough to not have overfitted to the training data, as indicated by the similar mean-squared errors of the training and testing sets. Therefore, applying ridge regression did not impact the model, as no regularization was really needed.

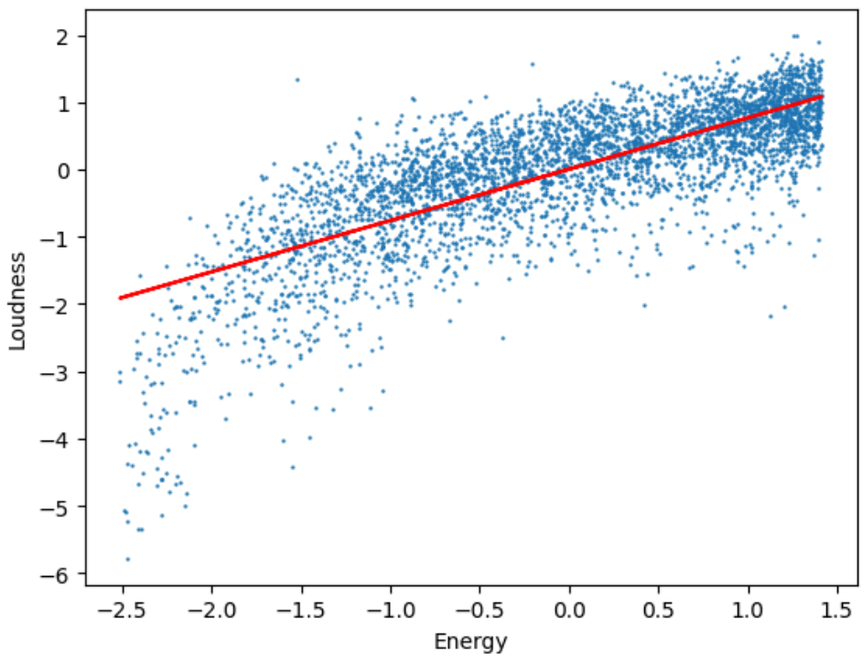
In [26]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
```

In [27]:
```
X = df[['energy']]
y = df[['loudness']]

# create the training and testing (validation) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# fit a linear regression model regressing loudness on energy
```

```
reg = LinearRegression().fit(X_train, y_train)
reg.score(X_test, y_test)
```

Out[27]:
```
0.5928131827456873
```

In [ ]:
```
# plot a scatterplot of the data and the regression line
plt.plot(X_test, reg.predict(X_test), color = 'r')
plt.scatter(X_test, y_test, s = 0.5)
plt.xlabel("Energy")
plt.ylabel("Loudness")
plt.show()
```



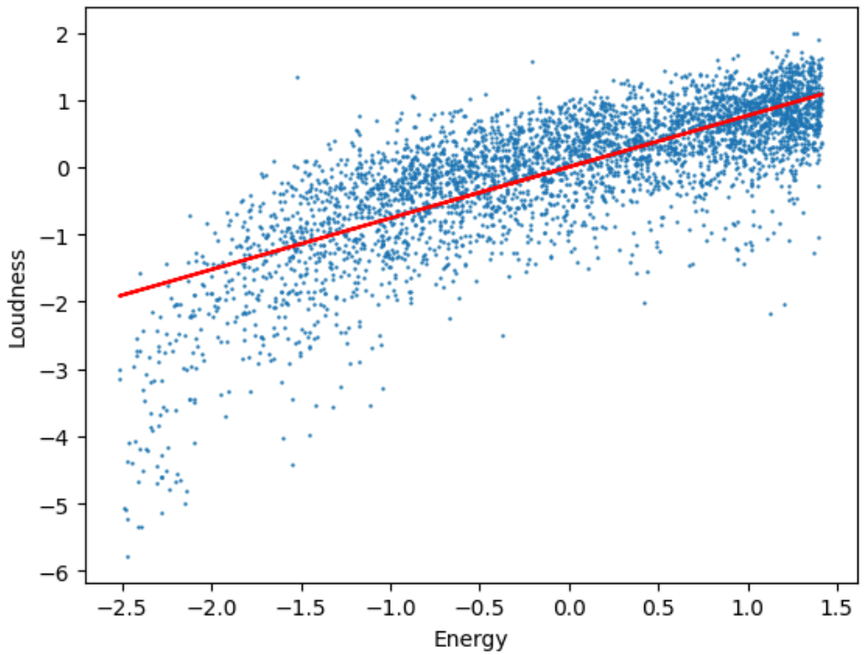
In [ ]:
```
train_MSE = mean_squared_error(y_train, reg.predict(X_train))
test_MSE = mean_squared_error(y_test, reg.predict(X_test))
print('Train MSE: ', train_MSE)
print('Test MSE: ', test_MSE)
```

```
Train MSE:  0.403851304649083
Test MSE:  0.4318246345503502
```

In [ ]:
```
from sklearn.linear_model import Ridge

# employ ridge regression with alpha = 1
lin_with_regularization = Ridge(alpha = 1)
lin_with_regularization.fit(X_train, y_train)
lin_with_regularization.score(X_test, y_test)
```

Out[ ]:
```
0.5928105584823308
```

In [ ]:
```
plt.plot(X_test, lin_with_regularization.predict(X_test), color = 'r')
plt.scatter(X_test, y_test, s = 0.5)
plt.xlabel("Energy")
plt.ylabel("Loudness")
plt.show()
```



In [ ]:
```
train_MSE = mean_squared_error(y_train, lin_with_regularization.predict(X_train))
test_MSE = mean_squared_error(y_test, lin_with_regularization.predict(X_test))
print('Train MSE: ', train_MSE)
print('Test MSE: ', test_MSE)
```

```
Train MSE:  0.40385130693463783
Test MSE:  0.43182741760107024
```

IV. Logistic Regression Analysis

How was logistic regression analysis applied in your project? What did you learn about your data set from this analysis and were you able to use this analysis for feature importance? Was regularization needed?

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score
from tabulate import tabulate
```

```
rando = 1
df_logistic = df.copy()

# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    df_logistic.drop(['track_genre'], axis=1),
    df_logistic.track_genre, test_size=0.2, random_state=rando)
```

We take a subset of the whole data set to be able to build the logistic regression model and visualize the results efficiently. We also split the data into training and testing sets, with a split of 80/20.

In [ ]:
```
# Build the logistic regression model
pipeline = make_pipeline(
    StandardScaler(),
    LogisticRegression(penalty='l1', solver='saga', max_iter=10000)
)
pipeline.fit(X_train, y_train)

# Determine the model's accuracy
accuracy = pipeline.score(X_test, y_test)
print(f"Model accuracy: {accuracy:.2f}")
```

```
Model accuracy: 0.55
```

In [ ]:
```
from sklearn.model_selection import cross_val_score
cv_score_logistic = cross_val_score(pipeline, X_test, y_test, cv=5, scoring='accuracy')
cv_score_logistic
```

Out[ ]:
```
array([0.53441802, 0.5112782 , 0.53258145, 0.52756892, 0.54761905])
```

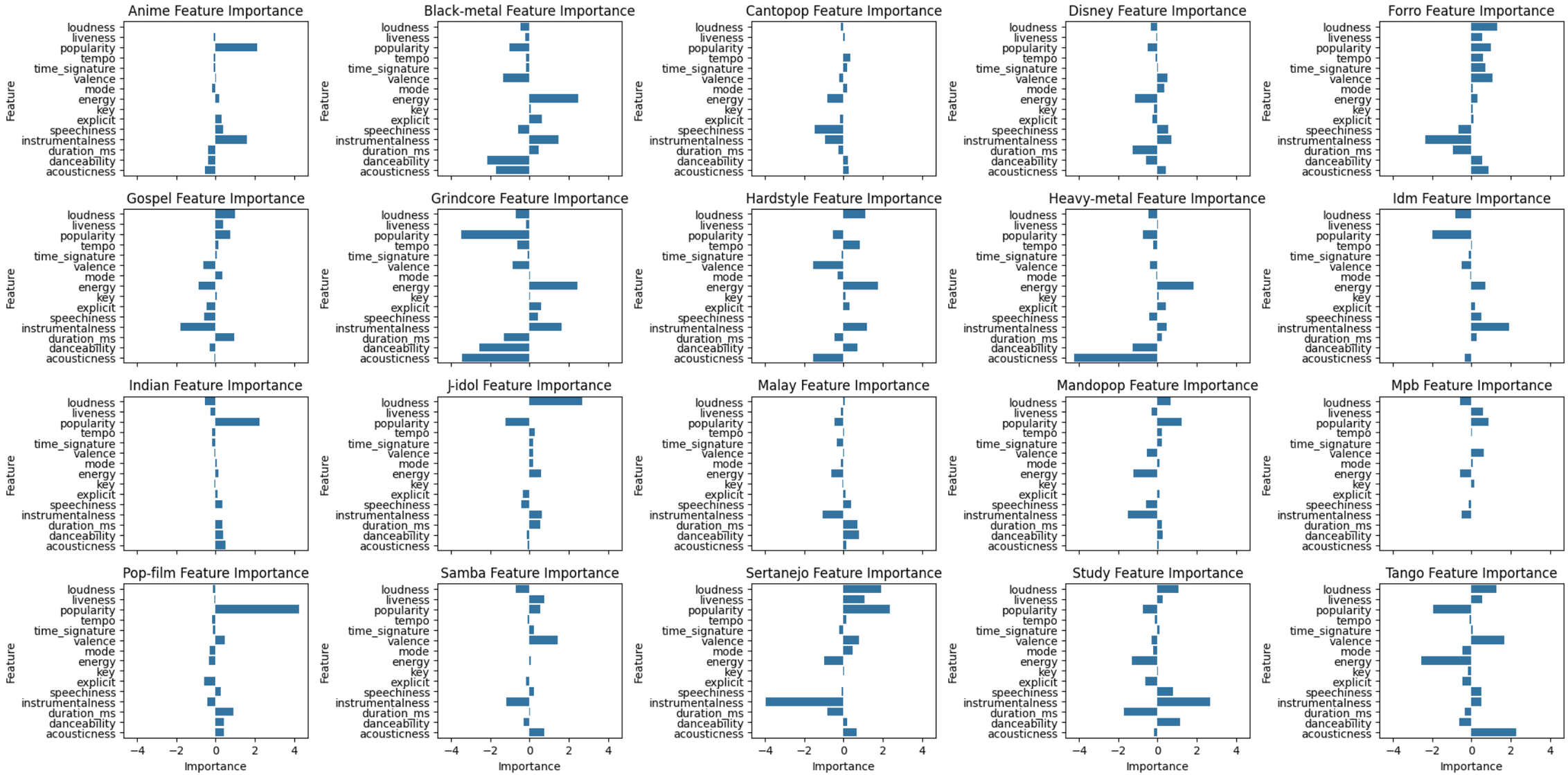
In [ ]:
```
np.mean(cv_score_logistic)
```

Out[ ]:
```
0.5306931283151558
```

We can get an idea of how each feature is used in the model. The following bar plots are separated by genre, and show the feature importance in each.

In [ ]:
```
# Determine the importance of each of the features
model = pipeline.named_steps['logisticregression']

feature_importance = pd.DataFrame(model.coef_, columns=X_train.columns, index=genres)
feature_importance['Genre'] = model.classes_

feature_importance_melted = feature_importance.melt(id_vars='Genre', var_name='Feature', value_name='Importance')

average_importance = feature_importance_melted.groupby('Feature')['Importance'].mean().sort_values(ascending=False).index
feature_importance_melted['Feature'] = pd.Categorical(feature_importance_melted['Feature'], categories=average_importance, ordered=True)

# Build bar plots for each of the genres, showing the importance of each feature
fig, axes = plt.subplots(4, 5, figsize=(20, 10), sharex=True)

for i, genre in enumerate(genres):
    ax = axes[i // 5, i % 5]
    sns.barplot(x='Importance', y='Feature', data=feature_importance_melted[feature_importance_melted['Genre'] == genre], ax=ax)
    ax.set_title(f'{genre.capitalize()} Feature Importance')
```

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

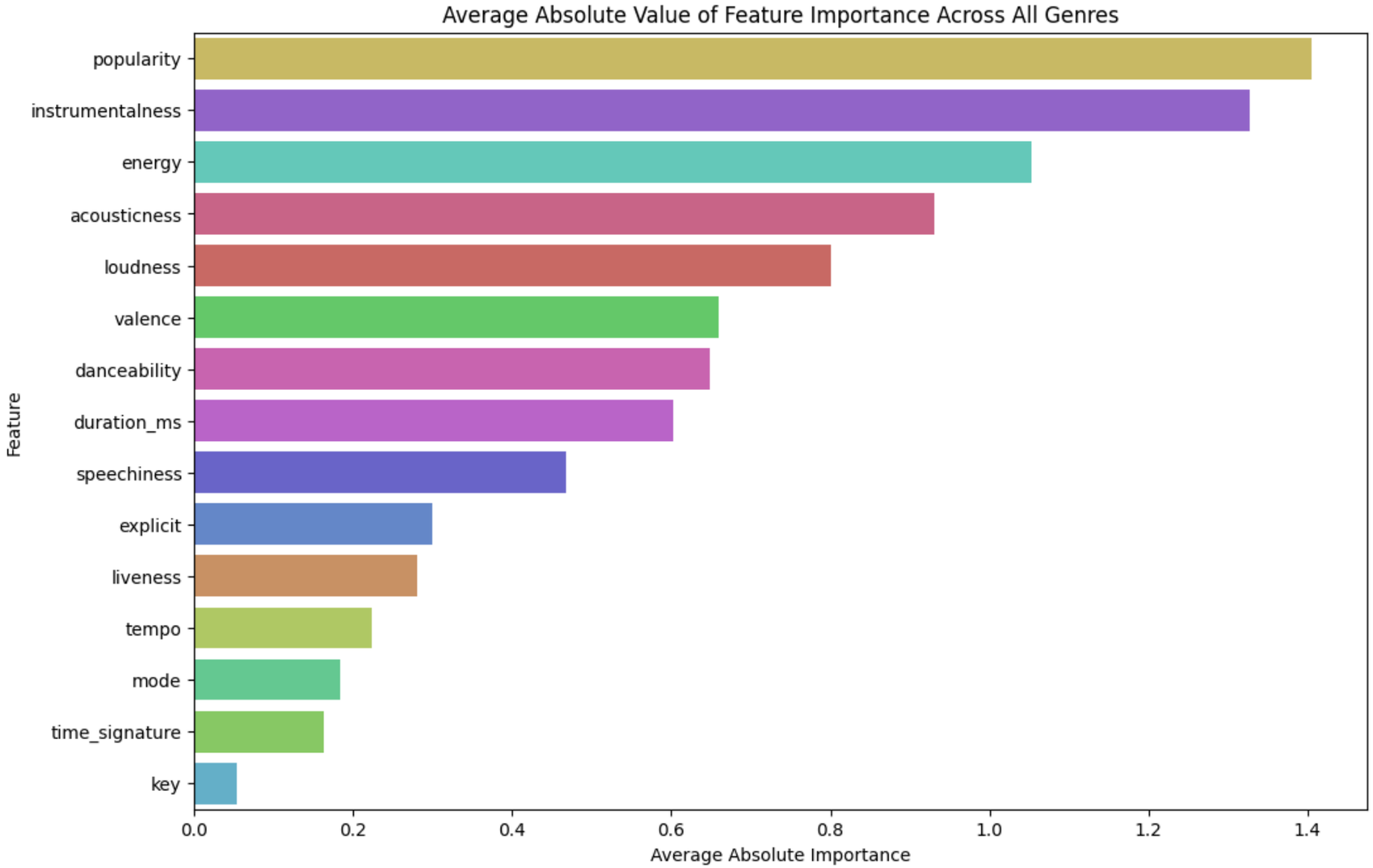


We can see that there is a clear difference in the importance of features like popularity, instrumetalness, and energy versus the less important features like mode, time signature, and key.

In [ ]:
```
# Print out the importance values for each of the features by genre
print(tabulate(feature_importance.round(4), headers='keys', tablefmt='pretty', numalign='center', stralign='center'))

# Calculate the average absolute value of importance for each feature
```

```
average_abs_importance = feature_importance_melted.groupby('Feature', observed=False)['Importance'].apply(lambda x: np.mean(np.abs(x)))

# Create a DataFrame for plotting
average_abs_importance_df = average_abs_importance.reset_index().sort_values(by='Importance', ascending=False)

# Plot the average absolute value of importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=average_abs_importance_df, hue='Feature', palette='hls', order=average_abs_importance_df.sort_values('Importance', ascending=False).Feature)
plt.title('Average Absolute Value of Feature Importance Across All Genres')
plt.xlabel('Average Absolute Importance')
plt.ylabel('Feature')
plt.show()
```

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	Genre
anime	2.1316	-0.3808	0.3165	-0.3596	0.1665	0.0	0.0001	-0.1936	0.391	-0.5373	1.617	-0.088	0.0087	-0.103	-0.0874	anime
black-metal	-1.0447	0.4631	0.6202	-2.1651	2.5019	0.0479	-0.4789	-0.0264	-0.5885	-1.7283	1.4762	-0.2153	-1.3642	-0.169	-0.1915	black-metal
cantopop	0.0007	-0.2555	-0.1831	0.2412	-0.836	-0.016	-0.127	0.192	-1.4654	0.2759	-0.9579	0.0744	-0.2097	0.3544	0.1888	cantopop
disney	-0.5169	-1.2507	-0.2651	-0.564	-1.1548	-0.1894	-0.3387	0.3371	0.5554	0.438	0.7108	-0.0611	0.5155	-0.103	3	0.0156	disney
forro	0.983	-0.9405	0.1038	0.5427	0.3142	0.0471	1.32	0.0628	-0.6794	0.8743	-2.3476	0.5531	1.057	0.5858	0.6936	forro
gospel	0.7435	0.9416	-0.4717	-0.2983	-0.8794	0.0542	0.9947	0.3437	-0.5776	-0.0434	-1.8046	0.3817	-0.6405	0.1565	0.0826	gospel
grindcore	-3.5029	-1.2876	0.5992	-2.577	2.4482	0.0338	-0.6968	0.0344	0.4163	-3.4497	1.6563	-0.1926	-0.8745	-0.628	8	-0.1171	grindcore
hardstyle	-0.5284	-0.444	0.2981	0.7089	1.7588	0.0973	1.13	-0.3026	-0.0007	-1.5636	1.1769	0.0	-1.55	0.8337	-0.0828	hardstyle
heavy-metal	-0.7325	0.2398	0.4251	-1.2829	1.8308	0.0657	-0.459	-0.0675	-0.4323	-4.2579	0.4739	0.0159	-0.3957	-0.224	-0.0271	heavy-metal
idm	-2.0062	0.2517	0.1924	0.0	0.7154	-0.0157	-0.8198	-0.0485	0.4971	-0.3473	1.901	-0.0172	-0.5192	0.0325	-0.1375	idm
indian	2.2401	0.356	0.1056	0.3691	0.1462	-0.0624	-0.5279	0.0671	0.3553	0.5241	0.0	-0.2534	-0.0694	-0.168	7	-0.1605	indian
j-idol	-1.2269	0.53	-0.3255	-0.1292	0.5764	0.0	2.6981	0.2018	-0.4145	-0.0801	0.6424	-0.0304	0.1831	0.2526	0.2016	j-idol
malay	-0.4787	0.7064	0.1012	0.8023	-0.6332	-0.0556	0.0538	-0.1519	0.3707	0.1254	-1.0551	-0.1508	0.007	0.0055	-0.3317	malay
mandopop	1.2276	0.2394	0.1098	0.2696	-1.2424	-0.0069	0.6748	0.1086	-0.5715	0.0674	-1.5242	-0.2964	-0.5245	0.2129	0.2403	mandopop
mpb	0.8549	-0.0	-0.0003	0.0	-0.5648	0.1303	-0.5722	0.0816	-0.1225	-0.0001	-0.4882	0.5911	0.6414	0.0151	0.001	mpb
pop-film	4.2766	0.9233	-0.5702	0.4121	-0.3538	-0.0144	-0.1334	-0.2809	0.2664	0.415	-0.4023	-0.0553	0.4495	-0.159	7	-0.1266	pop-film
samba	0.5434	0.0075	-0.1894	-0.2865	0.0831	-0.0101	-0.7104	-0.0122	0.233	0.7557	-1.1707	0.7376	1.4322	-0.090	3	0.2164	samba
sertanejo	2.3734	-0.8286	-0.0132	0.1969	-0.9746	0.0345	1.925	0.4757	-0.0877	0.6597	-3.9668	1.0533	0.7711	0.1465	-0.2121	sertanejo
study	-0.7327	-1.6919	-0.615	1.1455	-1.3264	0.0178	1.0567	-0.2225	0.7994	-0.1942	2.6698	0.2668	-0.3075	-0.148	3	0.1071	study
tango	-1.9592	-0.3235	-0.4797	-0.6334	-2.5664	-0.171	1.2928	-0.4566	0.5234	2.2749	0.5191	0.5607	1.6812	-0.082	5	0.0458	tango



V. KNN, Decision Trees, and Random Forest

KNN

We can fit a KNN model to the Spotify data by encoding the categorical response variable and using split data into 80/20 training and testing.

In [31]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.preprocessing import LabelEncoder

df_knn = df.copy()
label_encoder = LabelEncoder()

df_knn['track_genre'] = label_encoder.fit_transform(df_knn['track_genre'])
X_train, X_test, y_train, y_test = train_test_split(df_knn.drop(['track_genre'], axis=1), df_knn.track_genre, test_size=0.2)
X_train['explicit'] = X_train['explicit'].astype(int)
X_test['explicit'] = X_test['explicit'].astype(int)

k = 14
knn = KNeighborsRegressor(n_neighbors=k)
knn.fit(X_train, y_train)

knn_train_pred = knn.predict(X_train)
mse = mean_squared_error(y_train, knn_train_pred)
rmse = sqrt(mse)
print(f"rMSE: {rmse}")

knn_score = knn.score(X_test, y_test)
print(f"kNN Accuracy: {knn_score}")
```

```
rMSE: 4.155109436772109
kNN Accuracy: 0.3839119863538719
```

Cross Validation:

In [32]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1, 26)}
grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get best k
best_k = grid_search.best_params_['n_neighbors']
print(f"Optimal k: {best_k}")

# Train the model with the optimal k
knn_optimal = KNeighborsRegressor(n_neighbors=best_k)
knn_optimal.fit(X_train, y_train)

# Predict and evaluate
knn_train_pred_optimal = knn_optimal.predict(X_train)
mse_optimal = mean_squared_error(y_train, knn_train_pred_optimal)
rmse_optimal = sqrt(mse_optimal)
print(f"Optimal rMSE: {rmse_optimal}")

knn_test_pred_optimal = knn_optimal.predict(X_test)
knn_score_optimal = knn_optimal.score(X_test, y_test)
print(f"Optimal kNN Accuracy: {knn_score_optimal}")
```

```
Optimal k: 11
Optimal rMSE: 4.083738010674413
Optimal kNN Accuracy: 0.37882312215287484
```

Decision Tree

The usefulness of a Decision tree was explored in the following section. We set some parameters, such as a max depth of 20 splits, and a minimum sample in each of 20. Gini is used as a criterion. We ended up with a 53% overall accuracy.

In [33]:

```
import plotly.figure_factory as ff
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn import tree
from joblib import Parallel, delayed
from itertools import product
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image

X_train, X_test, y_train, y_test = train_test_split(df_knn.drop(['track_genre'], axis=1),
                                                    df_knn.track_genre, test_size=0.3)

music_tree = DecisionTreeClassifier(criterion='gini', min_samples_split=20, max_depth=20)
music_tree.fit(X_train, y_train)

music_tree_pred = music_tree.predict(X_test)

print("Confusion Matrix: \n", confusion_matrix(y_test, music_tree_pred))
print("Accuracy: \n", metrics.accuracy_score(y_test, music_tree_pred))
print(classification_report(y_test, music_tree_pred))
```

```
Confusion Matrix:
 [[132   2  11  13   4   6   0  20   6   5  15  12   9  11   5  14   6   8
   26   1]
 [  7 188   3   2   0   0  26  10  37   3   0   7   0   0   1   0   0   0
    2   1]
 [  6   1 104  37   2  14   1   5   2   3  16   3  32  44   8   4   4   4
    2   6]
 [  7   3  26 174   4   0   2   2   4  10   7   3  16   6   2   3   9   6
   10  21]
 [  7   0   5   0 161  19   0   2   0   0   2   1   0   2  18   0  38  40
    0   0]
 [  4   1  16   0  17 158   1   2   0   0   9   1   5   8  34   1   7   9
    0   0]
 [  1  32   0   0   0   0 246   4   6  12   0   1   2   0   0   0   0   0
    0   1]
 [ 15   5   5   3   5   5   2 192  16   6   4  21  11   3   0   1   0   1
    0   0]
 [  8  33   7   3   0   2   1  20 176   4   4  11   8   2   5   1   3   1
    0   1]
 [  7   6   2  10   0   0   7   7   0 221  10   0   7   2   0   1   0   0
   20   6]
 [ 16   0  19   5  14  19   0   3   0   4  93   0   9  21  28  47   5  10
    4   0]
 [  8  10  10   3   2   3   4  30  27   1   3 178  15   5   0   1   6   0
    0   1]
 [  8   1  27  23  11  13   1   4   9   5  21  14 109   9   8   3  18   2
    3  12]
 [  7   0  78   9   5  18   1   7   2   1  30   5  29 100   8  17   0   7
    0   6]
 [  7   1  15   1  33  23   0   2   0   2  28   2   8   9  94   0  58  18
    4   0]
 [ 14   0   4  10   3   2   1   2   1   2  60   0   6  24   1 176   0   3
    0   2]
 [  1   0   7   1  30  15   0   2   0   2   8   3  14   4  69   0 119  19
    1   1]
 [ 10   0  10   1  23   9   0   2   0   0  15   0   1  14  18   3   9 165
    0   0]
 [ 13   0   4   7   0   0   0   3   0  28   9   0   5   0   6   0   2   0
  219   4]
 [  2   4  14  30   1   1   0   2   4   6   1   2   7   1   1   0   0   0
    9 205]]
Accuracy:
 0.5361616836479038
              precision    recall  f1-score   support

           0       0.47      0.43      0.45       306
           1       0.66      0.66      0.66       287
           2       0.28      0.35      0.31       298
           3       0.52      0.55      0.54       315
           4       0.51      0.55      0.53       295
           5       0.51      0.58      0.54       273
           6       0.84      0.81      0.82       305
           7       0.60      0.65      0.62       295
           8       0.61      0.61      0.61       290
           9       0.70      0.72      0.71       306
          10       0.28      0.31      0.29       297
```

```
          11       0.67      0.58      0.62       307
          12       0.37      0.36      0.37       301
          13       0.38      0.30      0.34       330
          14       0.31      0.31      0.31       305
          15       0.65      0.57      0.60       311
          16       0.42      0.40      0.41       296
          17       0.56      0.59      0.58       280
          18       0.73      0.73      0.73       300
          19       0.76      0.71      0.73       290

    accuracy                           0.54      5987
   macro avg       0.54      0.54      0.54      5987
weighted avg       0.54      0.54      0.54      5987
```

In [ ]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': ['gini'],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

dt = DecisionTreeClassifier(random_state=123)


grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)

# Perform grid search
grid_search.fit(X_train, y_train)

# Print best parameters and best score
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

# Use the best model to predict on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate the model
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
Best Cross-Validation Accuracy: 0.5552691935380942
              precision    recall  f1-score   support

           0       0.51      0.47      0.49       287
           1       0.70      0.65      0.67       298
           2       0.31      0.28      0.29       298
           3       0.60      0.44      0.51       315
           4       0.58      0.59      0.59       315
           5       0.51      0.58      0.54       288
           6       0.86      0.85      0.85       294
           7       0.65      0.64      0.64       291
           8       0.63      0.58      0.60       302
           9       0.76      0.71      0.74       302
          10       0.29      0.30      0.30       297
          11       0.63      0.62      0.62       291
          12       0.41      0.34      0.37       293
          13       0.34      0.49      0.40       299
          14       0.33      0.36      0.34       298
          15       0.62      0.77      0.69       296
          16       0.51      0.48      0.49       337
          17       0.55      0.62      0.59       303
          18       0.70      0.73      0.71       294
          19       0.81      0.64      0.72       289

    accuracy                           0.56      5987
   macro avg       0.56      0.56      0.56      5987
weighted avg       0.56      0.56      0.56      5987
```

In [ ]:
```
cv_score_dt = cross_val_score(dt, X_test, y_test, cv=5, scoring='accuracy')
cv_score_dt
```

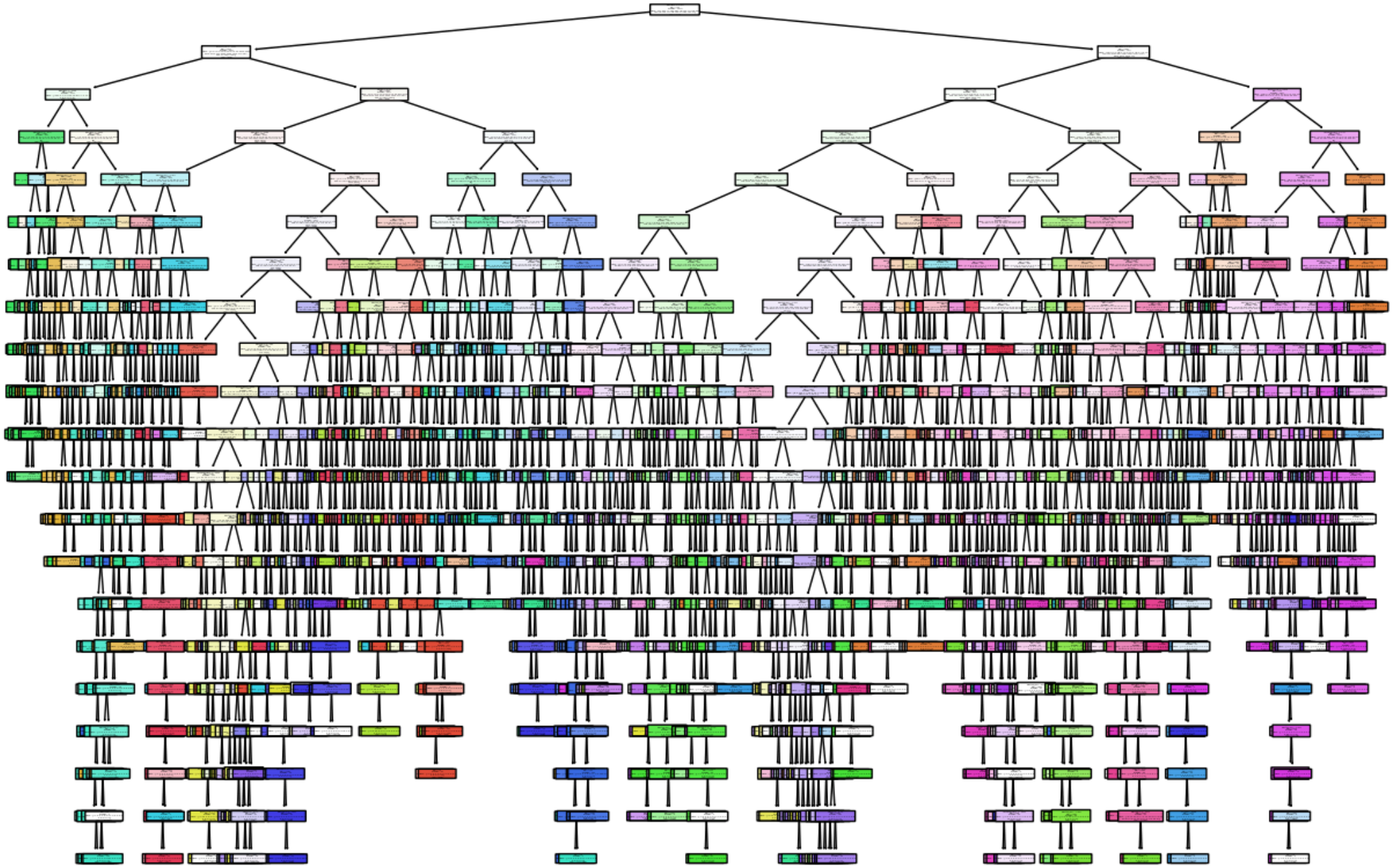
Out[ ]:
```
array([0.4933222 , 0.47495826, 0.46616541, 0.50459482, 0.48788638])
```

In [ ]:
```
np.mean(cv_score_dt)
```

Out[ ]:
```
0.4853854167974193
```

The following plot shows the process of the decision tree. The different labels are color coded in each of the boxes, with the strength of the predictions represented by the transparency of the color.

In [ ]:
```
plt.figure(figsize=(15,10))
plot_tree(music_tree, filled=True, feature_names=list(X_train.columns), class_names=genres, rounded=True)
plt.show()
```



Random Forest

When fitting the Random Forest model, we can see some similarity in the feature importance to the Logistic Regression.

In [34]:
```
music_rf = RandomForestClassifier(random_state=123)
music_rf.fit(X_train, y_train)

music_rf_pred = music_rf.predict_proba(X_test)

music_rf_importance = pd.DataFrame({
    'feature': X_train.columns,
    'importance': music_rf.feature_importances_
}).sort_values('importance', ascending=False).reset_index().drop('index', axis=1)
music_rf_importance
```

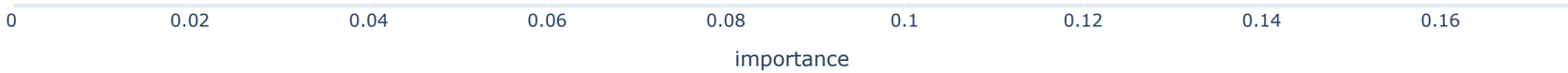
Out[34]:

	feature	importance
0	popularity	0.166057
1	acousticness	0.100013
2	duration_ms	0.099923
3	loudness	0.080884
4	energy	0.080626
5	danceability	0.078105
6	valence	0.077792
7	instrumentalness	0.075068
8	tempo	0.068005
9	speechiness	0.063770
10	liveness	0.056186
11	key	0.032148
12	mode	0.011425
13	time_signature	0.007267
14	explicit	0.002732

The Random Forest feature importance is represented in the bar plot shown below.

In [35]:
```
import plotly.express as px
import plotly.offline as pyo
pyo.init_notebook_mode(connected=True)

px.bar(music_rf_importance, y='feature', x='importance',
       orientation='h', title='Impurity Importance for Random Forest', height=1000)
```


Impurity Importance for Random Forest
feature
explicit
time_signature
mode
key
liveness
speechiness
tempo
instrumentalness
valence
danceability
energy
loudness
duration_ms
acousticness
popularity



In [ ]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report


rf = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4],
    'max_features': ['sqrt', 'log2'],
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, scoring='accuracy', verbose=2, n_jobs=-1)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

y_pred = grid_search.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning:

invalid value encountered in cast
```

```
Best Parameters: {'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Best Score: 0.6642326749484407
              precision    recall  f1-score   support

           0       0.62      0.53      0.57       287
           1       0.75      0.75      0.75       298
           2       0.49      0.38      0.43       298
           3       0.77      0.62      0.69       315
           4       0.63      0.68      0.65       315
           5       0.57      0.72      0.64       288
           6       0.88      0.89      0.89       294
           7       0.82      0.76      0.79       291
           8       0.70      0.70      0.70       302
           9       0.84      0.81      0.82       302
          10       0.46      0.38      0.41       297
          11       0.73      0.74      0.74       291
          12       0.53      0.53      0.53       293
          13       0.45      0.38      0.41       299
          14       0.37      0.41      0.38       298
          15       0.65      0.84      0.74       296
          16       0.54      0.54      0.54       337
          17       0.65      0.67      0.66       303
          18       0.80      0.93      0.86       294
          19       0.81      0.86      0.83       289

    accuracy                           0.65      5987
   macro avg       0.65      0.66      0.65      5987
weighted avg       0.65      0.65      0.65      5987
```

In [ ]:
```
cv_score_rf = cross_val_score(rf, X_test, y_test, cv=5, scoring='accuracy')
cv_score_rf
```

Out[ ]:
```
array([0.63856427, 0.61936561, 0.63074353, 0.64828739, 0.63324979])
```

In [ ]:
```
np.mean(cv_score_rf)
```

Out[ ]:
```
0.63404211697859
```

VI. PCA and Clustering

Principal Component Analysis is a technique for dimensionality reduction when a data set is very high-dimensional (i.e., contains many features). The goal is to simplify the data set while retaining as much information or variability in the data as possible.

Clustering is an unsupervised machine learning technique that creates groups in the data. For this clustering analysis, we attempt to cluster based on time signature.

Clustering is susceptible to a phenomenon known as the curse of dimensionality, in which data set is so high-dimensional and complex that clustering is difficult to perform and largely inaccurate, as the more complex a data set is, the less meaningful distance metrics become. Thus, reducing the dimension of the data set may aid in clustering effectiveness. Our approach involves performing k-means clustering on the data, then performing principal component analysis to create a transformed (simpler) data set, performing k-means clustering again on the new PCA-transformed data, and finally comparing evaluation metrics for the two clustering schemes.

In [36]:
```
import plotly.express as px
import plotly.graph_objects as go
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, silhouette_samples, rand_score, adjusted_rand_score
```

In [37]:
```
ss = preprocessing.StandardScaler()
df_clustering = df_raw.drop(columns='track_genre')
# Create a standardized version of the data for modeling purposes after EDA
num_cols = df_clustering.columns.values.tolist()
num_cols.remove('time_signature')
df_clustering[num_cols] = ss.fit_transform(df_clustering.drop(columns='time_signature'))
```

In [38]:
```
set(df_clustering['time_signature'])
```

Out[38]:
```
{0, 1, 3, 4, 5}
```

In [39]:
```
kmeans = KMeans(n_clusters=5)
y_kmeans = kmeans.fit_predict(df_clustering.drop(columns='time_signature'))
```

```
/Users/peiyuanlee/miniforge3/envs/myenv/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

In [40]:
```
for i in np.arange(0, len(kmeans.labels_)):
  if kmeans.labels_[i] > 1:
    kmeans.labels_[i] += 1
set(kmeans.labels_)
```

Out[40]:
```
{0, 1, 3, 4, 5}
```

In [41]:
```
print(silhouette_score(df_clustering.drop(columns='time_signature'), kmeans.labels_))
```

```
0.12619622421023385
```

In [42]:
```
pca_U, pca_d, pca_V = np.linalg.svd(df_clustering.drop(columns='time_signature'))
```

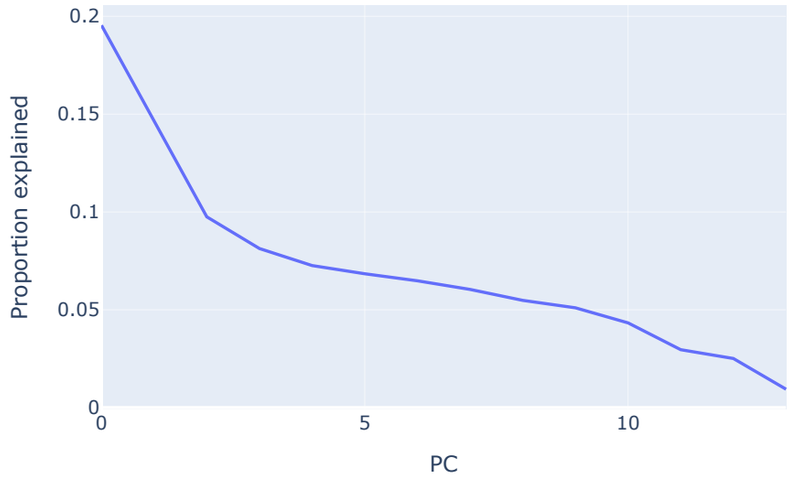
In [43]:
```
prop_var = np.square(pca_d) / sum(np.square(pca_d))
scree_data = pd.DataFrame(
{"PC": 1 + np.arange(0, prop_var.shape[0]),
"variability_explained": prop_var.round(4),
"cumulative_variability_explained": prop_var.cumsum().round(4)
})
scree_data.head(20)
```

Out[43]:

	PC	variability_explained	cumulative_variability_explained
0	1	0.1954	0.1954
1	2	0.1461	0.3416
2	3	0.0975	0.4391
3	4	0.0813	0.5204
4	5	0.0726	0.5930
5	6	0.0684	0.6614
6	7	0.0648	0.7263
7	8	0.0604	0.7866
8	9	0.0548	0.8415
9	10	0.0510	0.8925
10	11	0.0433	0.9358
11	12	0.0296	0.9655
12	13	0.0251	0.9906
13	14	0.0094	1.0000

In [44]:

```
px.line(x=np.arange(14),
y=scree_data.iloc[range(14), :].loc[:, 'variability_explained'],
labels={"x": "PC",
"y": "Proportion explained"},
width=600, height=400)
```



We attempted to perform Principal Component Analysis on the data to prepare for clustering based on time signature. The scree plot indicates that the first two principal components capture about 34.16% of the variability in the data, and after that, each principal component makes a small, consistent contribution. Unfortunately, keeping only the first two principal components would simply result in a data set that does not capture nearly enough information from the original data to be usable. Moreover, if we want to retain most of the information in the original data, let's say 90%, then we would need to keep the first ten principal components, which is not a very successful dimensionality reduction down from fourteen original features.

In [45]:
```
X_train_pca = np.dot(df_clustering.drop(columns='time_signature'), pca_V[np.arange(0, 9)].T)
X_train_pca = pd.DataFrame(X_train_pca, columns=['PC' + str(x) for x in np.arange(1, 10)])
X_train_pca.head()
```

Out[45]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	-1.737953	-1.237693	-0.573628	-1.434923	0.417737	-0.251574	-0.329893	-0.705392	-0.509406
1	-1.502498	-0.722715	-1.620502	0.727191	-0.105492	-0.296720	-1.382797	0.192409	-0.210325
2	-1.223136	-1.176942	-0.947351	0.804374	0.866419	-0.695013	-1.072830	-0.055739	0.416886
3	-1.750299	-0.736756	-0.589327	-1.595235	0.480411	-0.535357	0.041964	-0.623377	-0.235042
4	-1.099484	-1.289466	-0.236192	-0.447316	1.276546	0.314724	-0.978155	-1.587110	-0.371327

In [46]:
```
kmeans_new = KMeans(n_clusters=5)
y_kmeans = kmeans_new.fit_predict(X_train_pca)
```

```
/Users/peiyuanlee/miniforge3/envs/myenv/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:1412: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

In [47]:
```
for i in np.arange(0, len(kmeans_new.labels_)):
  if kmeans_new.labels_[i] > 1:
    kmeans_new.labels_[i] += 1
set(kmeans_new.labels_)
```

Out[47]:
```
{0, 1, 3, 4, 5}
```

In [48]:
```
print(silhouette_score(X_train_pca, kmeans_new.labels_))
```

```
0.14164406496997287
```

Keeping the first nine principal components and clustering with the new PCA transformed data set, we obtain only a slightly higher silhouette score of 0.1296, compared to a score of 0.1098 before performing PCA. Silhouette score is a measure of how tightly and distinctly the data is clustered, where 1 is tightly clustered and 0 is loosely (and indistinctively) clustered. It is one measure of effectiveness for a clustering algorithm. The calculated silhouette scores for both clustering schemes suggests that the effects of PCA are minimal for this data set, so we will not employ it in the main analysis of our classification of track genres.

VI. Neural Networks

In [ ]:
```
df_NN = df.copy()
df_NN.head()
```

Out[ ]:

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	track_genre
5000	2.352090	0.252651	-0.181294	0.063329	0.812652	0.215398	1.212446	-1.349109	-0.285892	-1.094908	-0.541600	-0.382670	0.301861	0.178699	0.241705	anime
5001	2.479707	0.127650	-0.181294	-0.576031	1.159506	-0.912980	1.222452	0.741230	-0.347826	-1.131754	-0.119504	0.341987	-0.256109	-1.062736	0.241705	anime
5002	3.117792	-0.366398	-0.181294	0.282539	1.187097	-1.195074	0.657329	0.741230	0.416500	-1.125528	-0.542039	-0.721608	-0.622775	-0.718561	0.241705	anime
5003	2.479707	0.108868	-0.181294	-0.137612	0.982138	0.497493	1.206533	-1.349109	0.151871	-0.982160	-0.542049	-0.678954	-0.463355	0.372413	0.241705	anime
5004	1.650196	0.208297	-0.181294	0.976702	0.524921	-1.477168	0.640500	-1.349109	-0.366125	-1.079190	-0.540688	-0.593647	0.214180	0.146523	0.241705	anime

From the feature importance derived from random forests, we select the top 10 features to reduce noise from irrelevant features. To optimize model performance, we designed three feedforward neural network architectures with 4, 6, and 10 layers, incorporating dropout and batch normalization layers to mitigate overfitting and stabilize training.

In [ ]:
```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
X = df_NN.drop(['track_genre', 'mode', 'explicit', 'liveness', 'key', 'time_signature'], axis = 1)
y = LabelEncoder().fit_transform(df_NN['track_genre'])

X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=2000, random_state=123, stratify=y)

X_train, X_valid, y_train, y_valid = train_test_split(
    X_temp, y_temp, test_size=1000, random_state=123, stratify=y_temp)

# Convert to categorical (onehot encoding matrices)
y_train = to_categorical(y_train, num_classes=20)
y_valid = to_categorical(y_valid, num_classes=20)
y_test = to_categorical(y_test, num_classes=20)
```

In [ ]:

```
import tensorflow as tf
from keras.layers import Dense, Activation, Dropout, BatchNormalization
from keras.models import Sequential, load_model, Model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.optimizers import Adam
from keras.regularizers import l2
from keras.losses import CategoricalCrossentropy
from keras.regularizers import L1L2, L2

class SequentialModel:
    def __init__(self):
        self.model = Sequential()

    def load_model(self, filepath):
        print('[Model] Loading model from file %s' % filepath)
        self.model = load_model(filepath)

    def build_model(self, config):
        """ Virtual Function """
        return

    def train(self, x, y, x_val, y_val, config):
        """ Virtual Function """
        return

    def evaluate(self, x_test, y_test, verbose=1):
        return self.model.evaluate(x_test,  y_test, verbose=verbose)

    def predict(self, x_test, verbose=1):
        return self.model.predict(x_test, verbose=verbose)

class Four_Layer_NN(SequentialModel):
    def __init__(self):
        super(Four_Layer_NN, self).__init__()

    def build_model(self, config):
        model = self.model

        input_shape = config["input_shape"]
        lr = config.get('lr', 0.001)
        decay = config.get("decay", 0.01)
        dropout = config.get('dropout', 0.2)

        model.add(Dense(64, kernel_regularizer=L2(l2=0.01), input_shape=input_shape, activation='elu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(32, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(16, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))
```

```
        model.add(Dense(20, kernel_regularizer=L2(l2=0.01), activation='softmax'))
        optimizer = Adam(learning_rate=lr, beta_1=0.9, beta_2=0.999,
                         amsgrad=False, epsilon=1e-8, decay=decay)
        model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
        print(model.summary())
        print("Model compiled.")

    def train(self, x, y, x_val, y_val, config):
        history = self.model.fit(x, y, epochs=config['epochs'], batch_size=config['batch_size'],
                                 validation_data=(x_val, y_val), shuffle=True,
                                )
        return history

class Six_Layer_NN(SequentialModel):
    def __init__(self):
        super(Six_Layer_NN, self).__init__()

    def build_model(self, config):
        model = self.model

        input_shape = config["input_shape"]
        lr = config.get('lr', 0.001)
        decay = config.get("decay", 0.01)
        dropout = config.get('dropout', 0.2)

        model.add(Dense(32, kernel_regularizer=L2(l2=0.01), input_shape=input_shape, activation='elu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(64, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(32, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(16, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(32, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(20, kernel_regularizer=L2(l2=0.01), activation='softmax'))
        optimizer = Adam(learning_rate=lr, beta_1=0.9, beta_2=0.999,
                         amsgrad=False, epsilon=1e-8, decay=decay)
        model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
        print(model.summary())
        print("Model compiled.")

    def train(self, x, y, x_val, y_val, config):
        history = self.model.fit(x, y, epochs=config['epochs'], batch_size=config['batch_size'],
                                 validation_data=(x_val, y_val), shuffle=True,
                                )
```

```
        return history

class Ten_Layer_NN(SequentialModel):
    def __init__(self):
        super(Ten_Layer_NN, self).__init__()

    def build_model(self, config):
        model = self.model

        input_shape = config["input_shape"]
        lr = config.get('lr', 0.001)
        decay = config.get("decay", 0.01)
        dropout = config.get('dropout', 0.2)

        model.add(Dense(32, kernel_regularizer=L2(l2=0.01), input_shape=input_shape, activation='elu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(64, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(128, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(64, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(32, activation='relu'))
        model.add(BatchNormalization(axis=1))

        model.add(Dense(16, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(8, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(16, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(32, activation='relu'))
        model.add(BatchNormalization(axis=1))
        model.add(Dropout(dropout))

        model.add(Dense(20, kernel_regularizer=L2(l2=0.01), activation='softmax'))
        optimizer = Adam(learning_rate=lr, beta_1=0.9, beta_2=0.999,
                         amsgrad=False, epsilon=1e-8, decay=decay)
        model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
        print(model.summary())
        print("Model compiled.")
```

```
    def train(self, x, y, x_val, y_val, config):
        history = self.model.fit(x, y, epochs=config['epochs'], batch_size=config['batch_size'],
                                 validation_data=(x_val, y_val), shuffle=True,
                                )
        return history
```

Four Layer Neural Network

In [ ]:
```
learning_rates = [0.001, 0.01, 0.1]
batch_sizes = [32, 64]
dropout = [0.2,0.3]
best_accuracy = 0
input_shape = X_train.shape[1:]
best_history = None
best_model = None

for lr in learning_rates:
    for batch in batch_sizes:
        for d in dropout:
            model = Four_Layer_NN()
            config = {
                        'input_shape': input_shape,
                        'epochs': 50,
                        'dropout': d,
                        'batch_size': batch,
                        'lr': lr
                    }
            model.build_model(config)
            history = model.train(X_train, y_train, X_valid, y_valid, config)
            loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
            print(f"Accuracy: {accuracy:.4f}")

            if accuracy > best_accuracy:
                best_model = model
                best_accuracy = accuracy
                best_params = {'learning_rate': lr, 'batch_size': batch, 'dropout': d}
                best_history = history

print("\nBest Hyperparameters:")
print(best_params)
print(f"Best Accuracy: {best_accuracy:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	704
batch_normalization (BatchNormalization)	(None, 64)	256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
batch_normalization_1 (BatchNormalization)	(None, 32)	128
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
batch_normalization_2 (BatchNormalization)	(None, 16)	64
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1780 - loss: 3.0373 - val_accuracy: 0.4450 - val_loss: 2.0548
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3697 - loss: 2.2233 - val_accuracy: 0.5010 - val_loss: 1.8159
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4088 - loss: 2.0432 - val_accuracy: 0.5120 - val_loss: 1.7264
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4218 - loss: 1.9605 - val_accuracy: 0.5350 - val_loss: 1.6675
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4351 - loss: 1.9100 - val_accuracy: 0.5220 - val_loss: 1.6223
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4328 - loss: 1.8601 - val_accuracy: 0.5340 - val_loss: 1.5859
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4434 - loss: 1.8339 - val_accuracy: 0.5450 - val_loss: 1.5718
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4552 - loss: 1.7982 - val_accuracy: 0.5500 - val_loss: 1.5483
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4488 - loss: 1.8050 - val_accuracy: 0.5480 - val_loss: 1.5321
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4444 - loss: 1.8070 - val_accuracy: 0.5560 - val_loss: 1.5098
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4543 - loss: 1.7758 - val_accuracy: 0.5430 - val_loss: 1.5140
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4609 - loss: 1.7486 - val_accuracy: 0.5510 - val_loss: 1.4944
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4737 - loss: 1.7286 - val_accuracy: 0.5460 - val_loss: 1.4955
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4642 - loss: 1.7360 - val_accuracy: 0.5560 - val_loss: 1.4781
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4625 - loss: 1.7296 - val_accuracy: 0.5500 - val_loss: 1.4666
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4641 - loss: 1.7205 - val_accuracy: 0.5650 - val_loss: 1.4635
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4673 - loss: 1.7279 - val_accuracy: 0.5570 - val_loss: 1.4488
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4713 - loss: 1.7102 - val_accuracy: 0.5690 - val_loss: 1.4618
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4693 - loss: 1.7182 - val_accuracy: 0.5670 - val_loss: 1.4502
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4713 - loss: 1.7033 - val_accuracy: 0.5710 - val_loss: 1.4403
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4721 - loss: 1.6916 - val_accuracy: 0.5690 - val_loss: 1.4354
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4700 - loss: 1.7030 - val_accuracy: 0.5650 - val_loss: 1.4417
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4709 - loss: 1.6914 - val_accuracy: 0.5530 - val_loss: 1.4322
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4632 - loss: 1.7032 - val_accuracy: 0.5530 - val_loss: 1.4329
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4709 - loss: 1.6910 - val_accuracy: 0.5720 - val_loss: 1.4180
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4679 - loss: 1.6940 - val_accuracy: 0.5620 - val_loss: 1.4217
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4681 - loss: 1.6965 - val_accuracy: 0.5690 - val_loss: 1.4236
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4787 - loss: 1.6667 - val_accuracy: 0.5620 - val_loss: 1.4146
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4741 - loss: 1.6910 - val_accuracy: 0.5610 - val_loss: 1.4112
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4745 - loss: 1.6753 - val_accuracy: 0.5860 - val_loss: 1.4032
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4762 - loss: 1.6745 - val_accuracy: 0.5660 - val_loss: 1.4000
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4665 - loss: 1.6815 - val_accuracy: 0.5720 - val_loss: 1.4067
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4861 - loss: 1.6515 - val_accuracy: 0.5830 - val_loss: 1.3895
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4775 - loss: 1.6615 - val_accuracy: 0.5840 - val_loss: 1.3907
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4777 - loss: 1.6593 - val_accuracy: 0.5710 - val_loss: 1.3956
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4721 - loss: 1.6697 - val_accuracy: 0.5650 - val_loss: 1.4013
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4782 - loss: 1.6542 - val_accuracy: 0.5680 - val_loss: 1.3793
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4832 - loss: 1.6593 - val_accuracy: 0.5790 - val_loss: 1.3882
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4794 - loss: 1.6579 - val_accuracy: 0.5730 - val_loss: 1.4000
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4767 - loss: 1.6574 - val_accuracy: 0.5640 - val_loss: 1.3908
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4720 - loss: 1.6791 - val_accuracy: 0.5710 - val_loss: 1.3946
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4739 - loss: 1.6583 - val_accuracy: 0.5750 - val_loss: 1.3806
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4700 - loss: 1.6505 - val_accuracy: 0.5770 - val_loss: 1.3763
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4755 - loss: 1.6563 - val_accuracy: 0.5720 - val_loss: 1.3688
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4845 - loss: 1.6431 - val_accuracy: 0.5920 - val_loss: 1.3659
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4771 - loss: 1.6480 - val_accuracy: 0.5730 - val_loss: 1.3839
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4814 - loss: 1.6449 - val_accuracy: 0.5780 - val_loss: 1.3670
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4806 - loss: 1.6472 - val_accuracy: 0.5790 - val_loss: 1.3696
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4854 - loss: 1.6322 - val_accuracy: 0.5670 - val_loss: 1.3656
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4912 - loss: 1.6291 - val_accuracy: 0.5670 - val_loss: 1.3803
Accuracy: 0.5660
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	704
batch_normalization_3 (BatchNormalization)	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 32)	2,080
batch_normalization_4 (BatchNormalization)	(None, 32)	128
dropout_4 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 16)	528
batch_normalization_5 (BatchNormalization)	(None, 16)	64
dropout_5 (Dropout)	(None, 16)	0
dense_7 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1392 - loss: 3.1796 - val_accuracy: 0.4310 - val_loss: 2.1901
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3063 - loss: 2.3979 - val_accuracy: 0.4840 - val_loss: 1.9408
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3497 - loss: 2.2005 - val_accuracy: 0.4820 - val_loss: 1.8115
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3556 - loss: 2.1103 - val_accuracy: 0.4990 - val_loss: 1.7418
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3774 - loss: 2.0385 - val_accuracy: 0.5160 - val_loss: 1.6971
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3916 - loss: 1.9840 - val_accuracy: 0.5100 - val_loss: 1.6656
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3994 - loss: 1.9680 - val_accuracy: 0.5110 - val_loss: 1.6354
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4046 - loss: 1.9433 - val_accuracy: 0.5210 - val_loss: 1.6190
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3979 - loss: 1.9445 - val_accuracy: 0.5210 - val_loss: 1.6047
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4068 - loss: 1.9360 - val_accuracy: 0.5170 - val_loss: 1.6031
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4111 - loss: 1.9258 - val_accuracy: 0.5230 - val_loss: 1.5791
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4104 - loss: 1.8858 - val_accuracy: 0.5340 - val_loss: 1.5547
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4131 - loss: 1.8824 - val_accuracy: 0.5220 - val_loss: 1.5655
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4118 - loss: 1.8685 - val_accuracy: 0.5360 - val_loss: 1.5515
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4148 - loss: 1.8869 - val_accuracy: 0.5270 - val_loss: 1.5449
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4209 - loss: 1.8546 - val_accuracy: 0.5370 - val_loss: 1.5289
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4204 - loss: 1.8532 - val_accuracy: 0.5340 - val_loss: 1.5271
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4287 - loss: 1.8326 - val_accuracy: 0.5440 - val_loss: 1.5177
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4308 - loss: 1.8417 - val_accuracy: 0.5320 - val_loss: 1.5285
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4141 - loss: 1.8585 - val_accuracy: 0.5410 - val_loss: 1.5100
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4270 - loss: 1.8394 - val_accuracy: 0.5410 - val_loss: 1.5176
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4198 - loss: 1.8453 - val_accuracy: 0.5300 - val_loss: 1.5069
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4243 - loss: 1.8241 - val_accuracy: 0.5400 - val_loss: 1.5079
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4269 - loss: 1.8177 - val_accuracy: 0.5340 - val_loss: 1.4957
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4272 - loss: 1.8323 - val_accuracy: 0.5410 - val_loss: 1.4952
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4327 - loss: 1.8084 - val_accuracy: 0.5380 - val_loss: 1.4945
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4294 - loss: 1.8217 - val_accuracy: 0.5430 - val_loss: 1.4808
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4293 - loss: 1.8053 - val_accuracy: 0.5520 - val_loss: 1.4861
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4366 - loss: 1.8005 - val_accuracy: 0.5220 - val_loss: 1.4989
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4336 - loss: 1.8065 - val_accuracy: 0.5410 - val_loss: 1.4928
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4364 - loss: 1.8160 - val_accuracy: 0.5440 - val_loss: 1.4881
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4291 - loss: 1.8174 - val_accuracy: 0.5440 - val_loss: 1.4717
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4334 - loss: 1.8115 - val_accuracy: 0.5360 - val_loss: 1.4846
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4325 - loss: 1.8159 - val_accuracy: 0.5420 - val_loss: 1.4848
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4383 - loss: 1.7803 - val_accuracy: 0.5520 - val_loss: 1.4666
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4255 - loss: 1.8182 - val_accuracy: 0.5460 - val_loss: 1.4727
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4327 - loss: 1.8160 - val_accuracy: 0.5280 - val_loss: 1.4721
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4398 - loss: 1.7864 - val_accuracy: 0.5500 - val_loss: 1.4651
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4354 - loss: 1.7962 - val_accuracy: 0.5330 - val_loss: 1.4789
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4382 - loss: 1.7784 - val_accuracy: 0.5280 - val_loss: 1.4582
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4384 - loss: 1.7802 - val_accuracy: 0.5460 - val_loss: 1.4558
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4436 - loss: 1.7731 - val_accuracy: 0.5370 - val_loss: 1.4698
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4441 - loss: 1.7780 - val_accuracy: 0.5480 - val_loss: 1.4682
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4352 - loss: 1.7885 - val_accuracy: 0.5490 - val_loss: 1.4648
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4373 - loss: 1.7981 - val_accuracy: 0.5480 - val_loss: 1.4553
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4437 - loss: 1.7781 - val_accuracy: 0.5420 - val_loss: 1.4562
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4326 - loss: 1.7921 - val_accuracy: 0.5420 - val_loss: 1.4556
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4492 - loss: 1.7625 - val_accuracy: 0.5520 - val_loss: 1.4532
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4280 - loss: 1.7979 - val_accuracy: 0.5430 - val_loss: 1.4569
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4421 - loss: 1.7922 - val_accuracy: 0.5440 - val_loss: 1.4399
Accuracy: 0.5370
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 64)	704
batch_normalization_6 (BatchNormalization)	(None, 64)	256
dropout_6 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 32)	2,080
batch_normalization_7 (BatchNormalization)	(None, 32)	128
dropout_7 (Dropout)	(None, 32)	0
dense_10 (Dense)	(None, 16)	528
batch_normalization_8 (BatchNormalization)	(None, 16)	64
dropout_8 (Dropout)	(None, 16)	0
dense_11 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1330 - loss: 3.2178 - val_accuracy: 0.4090 - val_loss: 2.3290
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3270 - loss: 2.3771 - val_accuracy: 0.4890 - val_loss: 1.9604
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3883 - loss: 2.1360 - val_accuracy: 0.4960 - val_loss: 1.8188
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4204 - loss: 2.0104 - val_accuracy: 0.5210 - val_loss: 1.7306
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4348 - loss: 1.9300 - val_accuracy: 0.5170 - val_loss: 1.6821
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4455 - loss: 1.8950 - val_accuracy: 0.5330 - val_loss: 1.6513
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4511 - loss: 1.8444 - val_accuracy: 0.5260 - val_loss: 1.6071
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4621 - loss: 1.8086 - val_accuracy: 0.5270 - val_loss: 1.5857
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4553 - loss: 1.8177 - val_accuracy: 0.5370 - val_loss: 1.5689
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4553 - loss: 1.7891 - val_accuracy: 0.5280 - val_loss: 1.5540
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4604 - loss: 1.7859 - val_accuracy: 0.5540 - val_loss: 1.5334
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4669 - loss: 1.7552 - val_accuracy: 0.5430 - val_loss: 1.5221
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4656 - loss: 1.7527 - val_accuracy: 0.5530 - val_loss: 1.5139
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4649 - loss: 1.7428 - val_accuracy: 0.5460 - val_loss: 1.5090
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4748 - loss: 1.7247 - val_accuracy: 0.5480 - val_loss: 1.4965
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4725 - loss: 1.7204 - val_accuracy: 0.5390 - val_loss: 1.4972
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4782 - loss: 1.7045 - val_accuracy: 0.5380 - val_loss: 1.4918
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4731 - loss: 1.7094 - val_accuracy: 0.5540 - val_loss: 1.4774
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4710 - loss: 1.7225 - val_accuracy: 0.5510 - val_loss: 1.4662
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4689 - loss: 1.7102 - val_accuracy: 0.5570 - val_loss: 1.4581
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4781 - loss: 1.6768 - val_accuracy: 0.5420 - val_loss: 1.4507
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4735 - loss: 1.6844 - val_accuracy: 0.5500 - val_loss: 1.4411
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4788 - loss: 1.6824 - val_accuracy: 0.5520 - val_loss: 1.4449
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4833 - loss: 1.6802 - val_accuracy: 0.5600 - val_loss: 1.4245
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4822 - loss: 1.6659 - val_accuracy: 0.5610 - val_loss: 1.4250
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4768 - loss: 1.6576 - val_accuracy: 0.5640 - val_loss: 1.4236
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4828 - loss: 1.6707 - val_accuracy: 0.5700 - val_loss: 1.4179
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4849 - loss: 1.6593 - val_accuracy: 0.5700 - val_loss: 1.4052
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4842 - loss: 1.6417 - val_accuracy: 0.5510 - val_loss: 1.4167
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4846 - loss: 1.6439 - val_accuracy: 0.5610 - val_loss: 1.4035
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4920 - loss: 1.6309 - val_accuracy: 0.5630 - val_loss: 1.4013
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4870 - loss: 1.6308 - val_accuracy: 0.5610 - val_loss: 1.4036
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4891 - loss: 1.6228 - val_accuracy: 0.5720 - val_loss: 1.3845
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4866 - loss: 1.6261 - val_accuracy: 0.5770 - val_loss: 1.3809
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4884 - loss: 1.6234 - val_accuracy: 0.5560 - val_loss: 1.3918
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4933 - loss: 1.6141 - val_accuracy: 0.5710 - val_loss: 1.3833
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4831 - loss: 1.6337 - val_accuracy: 0.5520 - val_loss: 1.3998
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4960 - loss: 1.6094 - val_accuracy: 0.5660 - val_loss: 1.3728
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4953 - loss: 1.6209 - val_accuracy: 0.5660 - val_loss: 1.3729
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4808 - loss: 1.6275 - val_accuracy: 0.5640 - val_loss: 1.3642
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4952 - loss: 1.6183 - val_accuracy: 0.5470 - val_loss: 1.3667
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4950 - loss: 1.6206 - val_accuracy: 0.5760 - val_loss: 1.3612
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4958 - loss: 1.6092 - val_accuracy: 0.5690 - val_loss: 1.3635
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5042 - loss: 1.5871 - val_accuracy: 0.5720 - val_loss: 1.3635
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4946 - loss: 1.6098 - val_accuracy: 0.5590 - val_loss: 1.3606
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4877 - loss: 1.6129 - val_accuracy: 0.5790 - val_loss: 1.3511
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4984 - loss: 1.6004 - val_accuracy: 0.5810 - val_loss: 1.3463
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4998 - loss: 1.5889 - val_accuracy: 0.5810 - val_loss: 1.3531
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4948 - loss: 1.6043 - val_accuracy: 0.5720 - val_loss: 1.3563
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4914 - loss: 1.6042 - val_accuracy: 0.5770 - val_loss: 1.3391
Accuracy: 0.5740
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 64)	704
batch_normalization_9 (BatchNormalization)	(None, 64)	256
dropout_9 (Dropout)	(None, 64)	0
dense_13 (Dense)	(None, 32)	2,080
batch_normalization_10 (BatchNormalization)	(None, 32)	128
dropout_10 (Dropout)	(None, 32)	0
dense_14 (Dense)	(None, 16)	528
batch_normalization_11 (BatchNormalization)	(None, 16)	64
dropout_11 (Dropout)	(None, 16)	0
dense_15 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1321 - loss: 3.2469 - val_accuracy: 0.3780 - val_loss: 2.4257
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2629 - loss: 2.5509 - val_accuracy: 0.4340 - val_loss: 2.0676
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3331 - loss: 2.2959 - val_accuracy: 0.4610 - val_loss: 1.9115
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3580 - loss: 2.1559 - val_accuracy: 0.4920 - val_loss: 1.8289
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3758 - loss: 2.0828 - val_accuracy: 0.4990 - val_loss: 1.7672
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3900 - loss: 2.0379 - val_accuracy: 0.5000 - val_loss: 1.7222
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3987 - loss: 2.0096 - val_accuracy: 0.5030 - val_loss: 1.6888
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4062 - loss: 1.9639 - val_accuracy: 0.5130 - val_loss: 1.6609
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4056 - loss: 1.9541 - val_accuracy: 0.5020 - val_loss: 1.6477
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4133 - loss: 1.9326 - val_accuracy: 0.5140 - val_loss: 1.6294
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4168 - loss: 1.9105 - val_accuracy: 0.5120 - val_loss: 1.6104
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4197 - loss: 1.8913 - val_accuracy: 0.5030 - val_loss: 1.6022
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4214 - loss: 1.8872 - val_accuracy: 0.5050 - val_loss: 1.5913
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4221 - loss: 1.8850 - val_accuracy: 0.5170 - val_loss: 1.5876
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4270 - loss: 1.8487 - val_accuracy: 0.5230 - val_loss: 1.5713
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4274 - loss: 1.8673 - val_accuracy: 0.5140 - val_loss: 1.5704
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4320 - loss: 1.8458 - val_accuracy: 0.5230 - val_loss: 1.5452
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4274 - loss: 1.8375 - val_accuracy: 0.5280 - val_loss: 1.5404
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4306 - loss: 1.8365 - val_accuracy: 0.5270 - val_loss: 1.5300
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4296 - loss: 1.8292 - val_accuracy: 0.5210 - val_loss: 1.5295
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4334 - loss: 1.8222 - val_accuracy: 0.5290 - val_loss: 1.5324
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4494 - loss: 1.8035 - val_accuracy: 0.5250 - val_loss: 1.5247
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4444 - loss: 1.7854 - val_accuracy: 0.5400 - val_loss: 1.5097
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4342 - loss: 1.8145 - val_accuracy: 0.5330 - val_loss: 1.5120
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4408 - loss: 1.7850 - val_accuracy: 0.5270 - val_loss: 1.5164
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4481 - loss: 1.7853 - val_accuracy: 0.5310 - val_loss: 1.5063
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4446 - loss: 1.8027 - val_accuracy: 0.5320 - val_loss: 1.5063
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4443 - loss: 1.7810 - val_accuracy: 0.5300 - val_loss: 1.4915
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4455 - loss: 1.7710 - val_accuracy: 0.5400 - val_loss: 1.4843
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4398 - loss: 1.7869 - val_accuracy: 0.5220 - val_loss: 1.4824
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4384 - loss: 1.7782 - val_accuracy: 0.5310 - val_loss: 1.4874
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4461 - loss: 1.7727 - val_accuracy: 0.5410 - val_loss: 1.4815
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4421 - loss: 1.7720 - val_accuracy: 0.5360 - val_loss: 1.4760
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4331 - loss: 1.7933 - val_accuracy: 0.5400 - val_loss: 1.4747
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4587 - loss: 1.7503 - val_accuracy: 0.5460 - val_loss: 1.4786
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4452 - loss: 1.7677 - val_accuracy: 0.5530 - val_loss: 1.4621
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4505 - loss: 1.7620 - val_accuracy: 0.5450 - val_loss: 1.4660
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4477 - loss: 1.7617 - val_accuracy: 0.5420 - val_loss: 1.4588
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4484 - loss: 1.7487 - val_accuracy: 0.5510 - val_loss: 1.4569
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4519 - loss: 1.7490 - val_accuracy: 0.5480 - val_loss: 1.4665
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4412 - loss: 1.7628 - val_accuracy: 0.5460 - val_loss: 1.4589
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4500 - loss: 1.7622 - val_accuracy: 0.5460 - val_loss: 1.4536
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4501 - loss: 1.7409 - val_accuracy: 0.5530 - val_loss: 1.4578
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4524 - loss: 1.7534 - val_accuracy: 0.5460 - val_loss: 1.4510
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4539 - loss: 1.7401 - val_accuracy: 0.5440 - val_loss: 1.4539
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4550 - loss: 1.7410 - val_accuracy: 0.5480 - val_loss: 1.4470
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4438 - loss: 1.7625 - val_accuracy: 0.5510 - val_loss: 1.4402
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4456 - loss: 1.7398 - val_accuracy: 0.5580 - val_loss: 1.4345
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4487 - loss: 1.7463 - val_accuracy: 0.5470 - val_loss: 1.4404
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4466 - loss: 1.7403 - val_accuracy: 0.5580 - val_loss: 1.4330
Accuracy: 0.5575
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_16 (Dense)	(None, 64)	704
batch_normalization_12 (BatchNormalization)	(None, 64)	256
dropout_12 (Dropout)	(None, 64)	0
dense_17 (Dense)	(None, 32)	2,080
batch_normalization_13 (BatchNormalization)	(None, 32)	128
dropout_13 (Dropout)	(None, 32)	0
dense_18 (Dense)	(None, 16)	528
batch_normalization_14 (BatchNormalization)	(None, 16)	64
dropout_14 (Dropout)	(None, 16)	0
dense_19 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.2960 - loss: 2.4303 - val_accuracy: 0.4610 - val_loss: 1.7796
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4002 - loss: 1.9871 - val_accuracy: 0.4890 - val_loss: 1.6952
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4160 - loss: 1.9398 - val_accuracy: 0.4970 - val_loss: 1.7294
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4138 - loss: 1.9244 - val_accuracy: 0.4890 - val_loss: 1.6648
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4153 - loss: 1.9200 - val_accuracy: 0.5170 - val_loss: 1.6206
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4168 - loss: 1.8948 - val_accuracy: 0.5250 - val_loss: 1.6377
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4193 - loss: 1.9140 - val_accuracy: 0.4990 - val_loss: 1.6517
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4228 - loss: 1.8875 - val_accuracy: 0.4990 - val_loss: 1.6367
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4222 - loss: 1.9084 - val_accuracy: 0.4980 - val_loss: 1.6685
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4200 - loss: 1.8915 - val_accuracy: 0.4970 - val_loss: 1.6654
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4210 - loss: 1.8821 - val_accuracy: 0.4890 - val_loss: 1.6480
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4244 - loss: 1.8825 - val_accuracy: 0.5310 - val_loss: 1.6123
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4274 - loss: 1.8678 - val_accuracy: 0.5130 - val_loss: 1.6472
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4251 - loss: 1.9027 - val_accuracy: 0.4770 - val_loss: 1.6691
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4207 - loss: 1.8835 - val_accuracy: 0.4890 - val_loss: 1.6514
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4262 - loss: 1.8810 - val_accuracy: 0.4860 - val_loss: 1.6451
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4302 - loss: 1.8831 - val_accuracy: 0.5320 - val_loss: 1.5786
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4337 - loss: 1.8603 - val_accuracy: 0.5270 - val_loss: 1.6132
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4285 - loss: 1.8727 - val_accuracy: 0.5250 - val_loss: 1.5805
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4428 - loss: 1.8558 - val_accuracy: 0.5020 - val_loss: 1.6353
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4344 - loss: 1.8764 - val_accuracy: 0.5230 - val_loss: 1.5871
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4376 - loss: 1.8621 - val_accuracy: 0.5270 - val_loss: 1.6087
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4290 - loss: 1.8836 - val_accuracy: 0.5080 - val_loss: 1.5986
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4299 - loss: 1.8741 - val_accuracy: 0.5270 - val_loss: 1.6051
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4405 - loss: 1.8539 - val_accuracy: 0.5210 - val_loss: 1.5784
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4256 - loss: 1.8802 - val_accuracy: 0.5380 - val_loss: 1.5802
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4430 - loss: 1.8545 - val_accuracy: 0.5270 - val_loss: 1.5673
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4346 - loss: 1.8681 - val_accuracy: 0.5170 - val_loss: 1.6192
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4319 - loss: 1.8622 - val_accuracy: 0.4930 - val_loss: 1.6542
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4365 - loss: 1.8586 - val_accuracy: 0.5060 - val_loss: 1.6045
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4307 - loss: 1.8665 - val_accuracy: 0.5150 - val_loss: 1.6234
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4357 - loss: 1.8600 - val_accuracy: 0.5300 - val_loss: 1.5617
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4389 - loss: 1.8493 - val_accuracy: 0.5120 - val_loss: 1.5924
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4331 - loss: 1.8691 - val_accuracy: 0.5350 - val_loss: 1.5728
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4388 - loss: 1.8474 - val_accuracy: 0.5120 - val_loss: 1.6230
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4386 - loss: 1.8559 - val_accuracy: 0.5170 - val_loss: 1.6093
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4330 - loss: 1.8492 - val_accuracy: 0.5200 - val_loss: 1.5977
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4390 - loss: 1.8492 - val_accuracy: 0.5120 - val_loss: 1.5953
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4358 - loss: 1.8641 - val_accuracy: 0.4970 - val_loss: 1.6315
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4363 - loss: 1.8597 - val_accuracy: 0.5370 - val_loss: 1.5751
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4408 - loss: 1.8399 - val_accuracy: 0.4970 - val_loss: 1.6133
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4321 - loss: 1.8583 - val_accuracy: 0.5270 - val_loss: 1.5741
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4317 - loss: 1.8711 - val_accuracy: 0.5340 - val_loss: 1.5793
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4368 - loss: 1.8700 - val_accuracy: 0.5160 - val_loss: 1.6095
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.4326 - loss: 1.8657 - val_accuracy: 0.5210 - val_loss: 1.6202
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4345 - loss: 1.8540 - val_accuracy: 0.5140 - val_loss: 1.6128
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4370 - loss: 1.8544 - val_accuracy: 0.5180 - val_loss: 1.6212
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4366 - loss: 1.8759 - val_accuracy: 0.5030 - val_loss: 1.6328
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4389 - loss: 1.8401 - val_accuracy: 0.5240 - val_loss: 1.6031
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4314 - loss: 1.8623 - val_accuracy: 0.5030 - val_loss: 1.6118
Accuracy: 0.5105
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_20 (Dense)	(None, 64)	704
batch_normalization_15 (BatchNormalization)	(None, 64)	256
dropout_15 (Dropout)	(None, 64)	0
dense_21 (Dense)	(None, 32)	2,080
batch_normalization_16 (BatchNormalization)	(None, 32)	128
dropout_16 (Dropout)	(None, 32)	0
dense_22 (Dense)	(None, 16)	528
batch_normalization_17 (BatchNormalization)	(None, 16)	64
dropout_17 (Dropout)	(None, 16)	0
dense_23 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2662 - loss: 2.5647 - val_accuracy: 0.4310 - val_loss: 1.8320
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3500 - loss: 2.1314 - val_accuracy: 0.4280 - val_loss: 1.8507
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3705 - loss: 2.0805 - val_accuracy: 0.4560 - val_loss: 1.7507
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3718 - loss: 2.0558 - val_accuracy: 0.4840 - val_loss: 1.7455
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.3754 - loss: 2.0628 - val_accuracy: 0.4960 - val_loss: 1.7070
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3756 - loss: 2.0323 - val_accuracy: 0.4730 - val_loss: 1.7336
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3851 - loss: 2.0199 - val_accuracy: 0.4820 - val_loss: 1.7362
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3775 - loss: 2.0426 - val_accuracy: 0.5160 - val_loss: 1.7054
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3878 - loss: 2.0331 - val_accuracy: 0.4860 - val_loss: 1.6992
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3892 - loss: 2.0174 - val_accuracy: 0.4990 - val_loss: 1.7081
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3909 - loss: 2.0093 - val_accuracy: 0.4920 - val_loss: 1.7218
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3863 - loss: 2.0259 - val_accuracy: 0.4660 - val_loss: 1.7086
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3962 - loss: 2.0203 - val_accuracy: 0.4890 - val_loss: 1.7173
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3824 - loss: 2.0175 - val_accuracy: 0.4890 - val_loss: 1.7178
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3947 - loss: 1.9986 - val_accuracy: 0.5050 - val_loss: 1.6976
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3838 - loss: 2.0144 - val_accuracy: 0.4740 - val_loss: 1.7393
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3890 - loss: 1.9974 - val_accuracy: 0.4780 - val_loss: 1.7004
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3952 - loss: 2.0132 - val_accuracy: 0.4940 - val_loss: 1.6920
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3898 - loss: 1.9981 - val_accuracy: 0.4860 - val_loss: 1.7040
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.3881 - loss: 2.0125 - val_accuracy: 0.4910 - val_loss: 1.7263
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3982 - loss: 2.0094 - val_accuracy: 0.5090 - val_loss: 1.6587
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4008 - loss: 1.9948 - val_accuracy: 0.4870 - val_loss: 1.6847
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3911 - loss: 2.0252 - val_accuracy: 0.5030 - val_loss: 1.7210
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3965 - loss: 1.9875 - val_accuracy: 0.4990 - val_loss: 1.6772
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3931 - loss: 1.9990 - val_accuracy: 0.4800 - val_loss: 1.6822
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3979 - loss: 1.9936 - val_accuracy: 0.4780 - val_loss: 1.7105
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3856 - loss: 2.0248 - val_accuracy: 0.4820 - val_loss: 1.7218
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3947 - loss: 2.0028 - val_accuracy: 0.4890 - val_loss: 1.6984
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3895 - loss: 2.0024 - val_accuracy: 0.5130 - val_loss: 1.6451
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3972 - loss: 2.0015 - val_accuracy: 0.4650 - val_loss: 1.7224
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3984 - loss: 2.0145 - val_accuracy: 0.5110 - val_loss: 1.7016
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3900 - loss: 2.0107 - val_accuracy: 0.4940 - val_loss: 1.7036
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3921 - loss: 1.9900 - val_accuracy: 0.4920 - val_loss: 1.6804
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3885 - loss: 1.9911 - val_accuracy: 0.4940 - val_loss: 1.6651
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3993 - loss: 1.9817 - val_accuracy: 0.5080 - val_loss: 1.6413
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3923 - loss: 1.9945 - val_accuracy: 0.5230 - val_loss: 1.6618
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3929 - loss: 1.9941 - val_accuracy: 0.5010 - val_loss: 1.6818
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4000 - loss: 1.9937 - val_accuracy: 0.4980 - val_loss: 1.7216
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3970 - loss: 1.9859 - val_accuracy: 0.4940 - val_loss: 1.6734
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3945 - loss: 1.9852 - val_accuracy: 0.5210 - val_loss: 1.7065
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3945 - loss: 2.0101 - val_accuracy: 0.5040 - val_loss: 1.6726
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4005 - loss: 1.9796 - val_accuracy: 0.4920 - val_loss: 1.7126
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.3907 - loss: 1.9951 - val_accuracy: 0.4840 - val_loss: 1.6818
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3883 - loss: 2.0173 - val_accuracy: 0.4650 - val_loss: 1.7501
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3930 - loss: 1.9896 - val_accuracy: 0.5070 - val_loss: 1.6844
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.3904 - loss: 2.0095 - val_accuracy: 0.4870 - val_loss: 1.6651
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3953 - loss: 1.9893 - val_accuracy: 0.5060 - val_loss: 1.6908
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3943 - loss: 2.0145 - val_accuracy: 0.5070 - val_loss: 1.6997
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.3966 - loss: 1.9958 - val_accuracy: 0.4950 - val_loss: 1.6919
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3973 - loss: 1.9944 - val_accuracy: 0.5080 - val_loss: 1.6839
Accuracy: 0.4895
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 64)	704
batch_normalization_18 (BatchNormalization)	(None, 64)	256
dropout_18 (Dropout)	(None, 64)	0
dense_25 (Dense)	(None, 32)	2,080
batch_normalization_19 (BatchNormalization)	(None, 32)	128
dropout_19 (Dropout)	(None, 32)	0
dense_26 (Dense)	(None, 16)	528
batch_normalization_20 (BatchNormalization)	(None, 16)	64
dropout_20 (Dropout)	(None, 16)	0
dense_27 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2959 - loss: 2.5100 - val_accuracy: 0.4170 - val_loss: 1.8895
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4202 - loss: 1.9207 - val_accuracy: 0.4850 - val_loss: 1.6959
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4188 - loss: 1.8881 - val_accuracy: 0.5030 - val_loss: 1.6485
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4357 - loss: 1.8373 - val_accuracy: 0.5000 - val_loss: 1.6582
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4315 - loss: 1.8362 - val_accuracy: 0.4820 - val_loss: 1.6730
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4349 - loss: 1.8330 - val_accuracy: 0.4970 - val_loss: 1.6033
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4369 - loss: 1.8288 - val_accuracy: 0.5090 - val_loss: 1.5762
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4392 - loss: 1.8148 - val_accuracy: 0.5390 - val_loss: 1.5420
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4479 - loss: 1.8119 - val_accuracy: 0.5180 - val_loss: 1.5817
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4530 - loss: 1.7843 - val_accuracy: 0.5270 - val_loss: 1.5364
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4589 - loss: 1.7790 - val_accuracy: 0.5360 - val_loss: 1.5645
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4587 - loss: 1.7734 - val_accuracy: 0.5220 - val_loss: 1.5590
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4521 - loss: 1.7784 - val_accuracy: 0.5190 - val_loss: 1.5762
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4584 - loss: 1.7632 - val_accuracy: 0.5490 - val_loss: 1.5429
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4463 - loss: 1.7941 - val_accuracy: 0.5330 - val_loss: 1.5595
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4554 - loss: 1.7747 - val_accuracy: 0.5320 - val_loss: 1.5884
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4496 - loss: 1.7783 - val_accuracy: 0.5290 - val_loss: 1.5280
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4540 - loss: 1.7723 - val_accuracy: 0.5450 - val_loss: 1.5112
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4555 - loss: 1.7875 - val_accuracy: 0.5370 - val_loss: 1.5277
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4553 - loss: 1.7741 - val_accuracy: 0.5440 - val_loss: 1.5463
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4618 - loss: 1.7533 - val_accuracy: 0.5240 - val_loss: 1.5324
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4576 - loss: 1.7827 - val_accuracy: 0.5450 - val_loss: 1.5042
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4671 - loss: 1.7556 - val_accuracy: 0.5170 - val_loss: 1.5673
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4563 - loss: 1.7657 - val_accuracy: 0.5290 - val_loss: 1.5146
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4481 - loss: 1.7855 - val_accuracy: 0.5120 - val_loss: 1.5600
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4623 - loss: 1.7676 - val_accuracy: 0.5480 - val_loss: 1.5034
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4663 - loss: 1.7533 - val_accuracy: 0.5460 - val_loss: 1.4818
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4565 - loss: 1.7714 - val_accuracy: 0.5290 - val_loss: 1.5472
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4675 - loss: 1.7543 - val_accuracy: 0.5420 - val_loss: 1.5002
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4681 - loss: 1.7590 - val_accuracy: 0.5110 - val_loss: 1.5577
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4502 - loss: 1.7818 - val_accuracy: 0.5350 - val_loss: 1.5388
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4432 - loss: 1.7951 - val_accuracy: 0.5200 - val_loss: 1.5108
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4537 - loss: 1.7731 - val_accuracy: 0.5220 - val_loss: 1.5441
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4663 - loss: 1.7541 - val_accuracy: 0.5100 - val_loss: 1.5364
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4560 - loss: 1.7652 - val_accuracy: 0.5530 - val_loss: 1.4796
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4624 - loss: 1.7579 - val_accuracy: 0.5070 - val_loss: 1.5158
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4646 - loss: 1.7538 - val_accuracy: 0.4910 - val_loss: 1.5910
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4640 - loss: 1.7649 - val_accuracy: 0.5590 - val_loss: 1.5076
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4623 - loss: 1.7576 - val_accuracy: 0.5380 - val_loss: 1.4935
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4739 - loss: 1.7382 - val_accuracy: 0.5300 - val_loss: 1.5044
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4650 - loss: 1.7481 - val_accuracy: 0.5450 - val_loss: 1.4849
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4579 - loss: 1.7694 - val_accuracy: 0.5400 - val_loss: 1.4992
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4674 - loss: 1.7492 - val_accuracy: 0.5140 - val_loss: 1.5247
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4666 - loss: 1.7430 - val_accuracy: 0.5150 - val_loss: 1.5679
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4601 - loss: 1.7481 - val_accuracy: 0.5390 - val_loss: 1.5184
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4726 - loss: 1.7247 - val_accuracy: 0.5090 - val_loss: 1.5677
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4615 - loss: 1.7633 - val_accuracy: 0.5460 - val_loss: 1.5015
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4694 - loss: 1.7291 - val_accuracy: 0.5020 - val_loss: 1.5227
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4612 - loss: 1.7581 - val_accuracy: 0.5370 - val_loss: 1.5156
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4687 - loss: 1.7430 - val_accuracy: 0.5120 - val_loss: 1.5367
Accuracy: 0.5295
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 64)	704
batch_normalization_21 (BatchNormalization)	(None, 64)	256
dropout_21 (Dropout)	(None, 64)	0
dense_29 (Dense)	(None, 32)	2,080
batch_normalization_22 (BatchNormalization)	(None, 32)	128
dropout_22 (Dropout)	(None, 32)	0
dense_30 (Dense)	(None, 16)	528
batch_normalization_23 (BatchNormalization)	(None, 16)	64
dropout_23 (Dropout)	(None, 16)	0
dense_31 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.2590 - loss: 2.5867 - val_accuracy: 0.4050 - val_loss: 2.0441
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3693 - loss: 2.0626 - val_accuracy: 0.4590 - val_loss: 1.7887
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3900 - loss: 1.9966 - val_accuracy: 0.4670 - val_loss: 1.7061
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3834 - loss: 1.9814 - val_accuracy: 0.4730 - val_loss: 1.7064
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3984 - loss: 1.9594 - val_accuracy: 0.4820 - val_loss: 1.6895
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3962 - loss: 1.9577 - val_accuracy: 0.4920 - val_loss: 1.7058
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4008 - loss: 1.9580 - val_accuracy: 0.4890 - val_loss: 1.6260
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3981 - loss: 1.9451 - val_accuracy: 0.4910 - val_loss: 1.6575
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4024 - loss: 1.9526 - val_accuracy: 0.4380 - val_loss: 1.7291
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4036 - loss: 1.9378 - val_accuracy: 0.4930 - val_loss: 1.6519
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4054 - loss: 1.9372 - val_accuracy: 0.5140 - val_loss: 1.6350
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4104 - loss: 1.9229 - val_accuracy: 0.5000 - val_loss: 1.6164
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4019 - loss: 1.9171 - val_accuracy: 0.4910 - val_loss: 1.6210
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4080 - loss: 1.9232 - val_accuracy: 0.5120 - val_loss: 1.6035
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4048 - loss: 1.9307 - val_accuracy: 0.4960 - val_loss: 1.6991
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4033 - loss: 1.9314 - val_accuracy: 0.4990 - val_loss: 1.6297
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4088 - loss: 1.9037 - val_accuracy: 0.5220 - val_loss: 1.5954
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4068 - loss: 1.9296 - val_accuracy: 0.5080 - val_loss: 1.6095
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4151 - loss: 1.8988 - val_accuracy: 0.5210 - val_loss: 1.5895
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4056 - loss: 1.9128 - val_accuracy: 0.5040 - val_loss: 1.6264
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4145 - loss: 1.9127 - val_accuracy: 0.5240 - val_loss: 1.6007
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4163 - loss: 1.9119 - val_accuracy: 0.4920 - val_loss: 1.6337
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4116 - loss: 1.9290 - val_accuracy: 0.5230 - val_loss: 1.6097
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4149 - loss: 1.9039 - val_accuracy: 0.4930 - val_loss: 1.6562
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4133 - loss: 1.9055 - val_accuracy: 0.5000 - val_loss: 1.6514
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4065 - loss: 1.9166 - val_accuracy: 0.5310 - val_loss: 1.6179
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4167 - loss: 1.9027 - val_accuracy: 0.5170 - val_loss: 1.6047
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4191 - loss: 1.8923 - val_accuracy: 0.4950 - val_loss: 1.6427
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4260 - loss: 1.8936 - val_accuracy: 0.5360 - val_loss: 1.5807
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4214 - loss: 1.8883 - val_accuracy: 0.4870 - val_loss: 1.6226
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4122 - loss: 1.9045 - val_accuracy: 0.4820 - val_loss: 1.6275
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4200 - loss: 1.8908 - val_accuracy: 0.5160 - val_loss: 1.6097
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4245 - loss: 1.8887 - val_accuracy: 0.5150 - val_loss: 1.6086
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4218 - loss: 1.8991 - val_accuracy: 0.5180 - val_loss: 1.6017
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4156 - loss: 1.9032 - val_accuracy: 0.4920 - val_loss: 1.6002
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4100 - loss: 1.9120 - val_accuracy: 0.5210 - val_loss: 1.6285
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4154 - loss: 1.9272 - val_accuracy: 0.5210 - val_loss: 1.6133
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4223 - loss: 1.8804 - val_accuracy: 0.5240 - val_loss: 1.5875
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4080 - loss: 1.9080 - val_accuracy: 0.5190 - val_loss: 1.6043
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4239 - loss: 1.8985 - val_accuracy: 0.5360 - val_loss: 1.5904
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4240 - loss: 1.8936 - val_accuracy: 0.5050 - val_loss: 1.6198
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4159 - loss: 1.9029 - val_accuracy: 0.4840 - val_loss: 1.5993
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4223 - loss: 1.8841 - val_accuracy: 0.5010 - val_loss: 1.5878
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4232 - loss: 1.8830 - val_accuracy: 0.4890 - val_loss: 1.6258
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4096 - loss: 1.9054 - val_accuracy: 0.5040 - val_loss: 1.5890
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4080 - loss: 1.9030 - val_accuracy: 0.4970 - val_loss: 1.6115
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4189 - loss: 1.8891 - val_accuracy: 0.5300 - val_loss: 1.5671
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4173 - loss: 1.8966 - val_accuracy: 0.5020 - val_loss: 1.6187
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4197 - loss: 1.8844 - val_accuracy: 0.5100 - val_loss: 1.5936
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4208 - loss: 1.8682 - val_accuracy: 0.5110 - val_loss: 1.6067
Accuracy: 0.5050
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
dense_32 (Dense)	(None, 64)	704
batch_normalization_24 (BatchNormalization)	(None, 64)	256
dropout_24 (Dropout)	(None, 64)	0
dense_33 (Dense)	(None, 32)	2,080
batch_normalization_25 (BatchNormalization)	(None, 32)	128
dropout_25 (Dropout)	(None, 32)	0
dense_34 (Dense)	(None, 16)	528
batch_normalization_26 (BatchNormalization)	(None, 16)	64
dropout_26 (Dropout)	(None, 16)	0
dense_35 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2039 - loss: 2.8946 - val_accuracy: 0.2780 - val_loss: 2.4542
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2376 - loss: 2.6955 - val_accuracy: 0.2610 - val_loss: 2.5933
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2367 - loss: 2.6850 - val_accuracy: 0.2730 - val_loss: 2.5700
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2466 - loss: 2.6887 - val_accuracy: 0.3140 - val_loss: 2.4853
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2541 - loss: 2.7061 - val_accuracy: 0.2380 - val_loss: 2.7066
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2570 - loss: 2.6809 - val_accuracy: 0.2870 - val_loss: 2.4605
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2635 - loss: 2.6425 - val_accuracy: 0.3010 - val_loss: 2.5495
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2601 - loss: 2.6665 - val_accuracy: 0.2680 - val_loss: 2.5816
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2660 - loss: 2.6405 - val_accuracy: 0.2790 - val_loss: 2.5599
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2720 - loss: 2.6438 - val_accuracy: 0.3310 - val_loss: 2.4081
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2639 - loss: 2.6573 - val_accuracy: 0.3290 - val_loss: 2.3469
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2666 - loss: 2.6371 - val_accuracy: 0.3070 - val_loss: 2.5017
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2621 - loss: 2.6546 - val_accuracy: 0.3270 - val_loss: 2.4571
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2715 - loss: 2.6172 - val_accuracy: 0.3410 - val_loss: 2.4183
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2831 - loss: 2.6189 - val_accuracy: 0.3260 - val_loss: 2.3576
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2784 - loss: 2.6186 - val_accuracy: 0.2490 - val_loss: 2.4451
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2730 - loss: 2.6097 - val_accuracy: 0.2790 - val_loss: 2.6806
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2734 - loss: 2.6409 - val_accuracy: 0.2650 - val_loss: 2.5940
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2729 - loss: 2.6537 - val_accuracy: 0.2710 - val_loss: 2.6608
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2658 - loss: 2.6364 - val_accuracy: 0.3540 - val_loss: 2.4057
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2808 - loss: 2.6096 - val_accuracy: 0.2840 - val_loss: 2.4585
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2763 - loss: 2.6293 - val_accuracy: 0.3110 - val_loss: 2.4058
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2689 - loss: 2.6483 - val_accuracy: 0.3380 - val_loss: 2.4972
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2746 - loss: 2.6307 - val_accuracy: 0.2700 - val_loss: 2.4531
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2667 - loss: 2.6401 - val_accuracy: 0.3020 - val_loss: 2.4772
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2719 - loss: 2.6404 - val_accuracy: 0.2620 - val_loss: 2.4850
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2751 - loss: 2.6128 - val_accuracy: 0.2850 - val_loss: 2.4983
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2796 - loss: 2.6179 - val_accuracy: 0.3300 - val_loss: 2.4257
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2702 - loss: 2.6309 - val_accuracy: 0.3860 - val_loss: 2.3137
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2794 - loss: 2.6202 - val_accuracy: 0.2240 - val_loss: 2.7486
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2867 - loss: 2.6378 - val_accuracy: 0.3320 - val_loss: 2.5236
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2854 - loss: 2.6037 - val_accuracy: 0.2800 - val_loss: 2.4323
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2687 - loss: 2.6364 - val_accuracy: 0.2610 - val_loss: 2.4245
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2721 - loss: 2.6090 - val_accuracy: 0.3200 - val_loss: 2.4318
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2786 - loss: 2.6239 - val_accuracy: 0.3380 - val_loss: 2.4154
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2850 - loss: 2.5868 - val_accuracy: 0.3860 - val_loss: 2.2973
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2760 - loss: 2.6239 - val_accuracy: 0.3970 - val_loss: 2.2532
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2692 - loss: 2.6429 - val_accuracy: 0.2520 - val_loss: 2.5514
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2736 - loss: 2.6444 - val_accuracy: 0.3460 - val_loss: 2.4039
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2730 - loss: 2.6402 - val_accuracy: 0.2990 - val_loss: 2.5381
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2849 - loss: 2.5955 - val_accuracy: 0.3140 - val_loss: 2.6130
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2730 - loss: 2.6241 - val_accuracy: 0.3020 - val_loss: 2.5193
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2736 - loss: 2.6160 - val_accuracy: 0.3520 - val_loss: 2.2877
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2743 - loss: 2.6031 - val_accuracy: 0.3240 - val_loss: 2.5387
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2710 - loss: 2.6466 - val_accuracy: 0.3540 - val_loss: 2.3695
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2649 - loss: 2.6508 - val_accuracy: 0.3480 - val_loss: 2.4364
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2749 - loss: 2.6355 - val_accuracy: 0.3600 - val_loss: 2.4095
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2833 - loss: 2.6267 - val_accuracy: 0.3500 - val_loss: 2.3756
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2860 - loss: 2.6080 - val_accuracy: 0.3500 - val_loss: 2.3556
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2812 - loss: 2.6059 - val_accuracy: 0.3030 - val_loss: 2.6480
Accuracy: 0.2965
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_36 (Dense)	(None, 64)	704
batch_normalization_27 (BatchNormalization)	(None, 64)	256
dropout_27 (Dropout)	(None, 64)	0
dense_37 (Dense)	(None, 32)	2,080
batch_normalization_28 (BatchNormalization)	(None, 32)	128
dropout_28 (Dropout)	(None, 32)	0
dense_38 (Dense)	(None, 16)	528
batch_normalization_29 (BatchNormalization)	(None, 16)	64
dropout_29 (Dropout)	(None, 16)	0
dense_39 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1874 - loss: 2.9843 - val_accuracy: 0.2540 - val_loss: 2.4878
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.1921 - loss: 2.8272 - val_accuracy: 0.2740 - val_loss: 2.5103
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2117 - loss: 2.8117 - val_accuracy: 0.2860 - val_loss: 2.4389
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2148 - loss: 2.7919 - val_accuracy: 0.2670 - val_loss: 2.6379
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2127 - loss: 2.7952 - val_accuracy: 0.1860 - val_loss: 2.7019
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2114 - loss: 2.8068 - val_accuracy: 0.2790 - val_loss: 2.5248
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2146 - loss: 2.8101 - val_accuracy: 0.2340 - val_loss: 2.6990
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2150 - loss: 2.7975 - val_accuracy: 0.2640 - val_loss: 2.7727
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2264 - loss: 2.7663 - val_accuracy: 0.2780 - val_loss: 2.5478
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2220 - loss: 2.7746 - val_accuracy: 0.2600 - val_loss: 2.6255
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2190 - loss: 2.7971 - val_accuracy: 0.2740 - val_loss: 2.5771
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2270 - loss: 2.7578 - val_accuracy: 0.2410 - val_loss: 2.6736
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2170 - loss: 2.7996 - val_accuracy: 0.2700 - val_loss: 2.6077
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2269 - loss: 2.7889 - val_accuracy: 0.2100 - val_loss: 2.6570
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2273 - loss: 2.7837 - val_accuracy: 0.2270 - val_loss: 2.7814
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.2224 - loss: 2.7825 - val_accuracy: 0.2730 - val_loss: 2.5719
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2332 - loss: 2.7713 - val_accuracy: 0.2370 - val_loss: 2.6690
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2337 - loss: 2.7759 - val_accuracy: 0.2980 - val_loss: 2.6052
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2304 - loss: 2.7737 - val_accuracy: 0.3100 - val_loss: 2.5453
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2343 - loss: 2.7609 - val_accuracy: 0.2950 - val_loss: 2.4627
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2361 - loss: 2.7457 - val_accuracy: 0.2890 - val_loss: 2.6032
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2380 - loss: 2.7695 - val_accuracy: 0.2570 - val_loss: 2.6267
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.2277 - loss: 2.7733 - val_accuracy: 0.3180 - val_loss: 2.5219
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2371 - loss: 2.7505 - val_accuracy: 0.2210 - val_loss: 2.9500
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2274 - loss: 2.7820 - val_accuracy: 0.3210 - val_loss: 2.4746
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2380 - loss: 2.7490 - val_accuracy: 0.2820 - val_loss: 2.7082
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2336 - loss: 2.7668 - val_accuracy: 0.2680 - val_loss: 2.5925
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2340 - loss: 2.7508 - val_accuracy: 0.2710 - val_loss: 2.7224
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2322 - loss: 2.7693 - val_accuracy: 0.3750 - val_loss: 2.4601
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2296 - loss: 2.7801 - val_accuracy: 0.2500 - val_loss: 2.5642
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2272 - loss: 2.7879 - val_accuracy: 0.2950 - val_loss: 2.5988
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2397 - loss: 2.7578 - val_accuracy: 0.3140 - val_loss: 2.5654
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2386 - loss: 2.7637 - val_accuracy: 0.2490 - val_loss: 2.6134
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2326 - loss: 2.7586 - val_accuracy: 0.3070 - val_loss: 2.5028
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2404 - loss: 2.7518 - val_accuracy: 0.2660 - val_loss: 2.5248
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2243 - loss: 2.7916 - val_accuracy: 0.3150 - val_loss: 2.4228
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.2457 - loss: 2.7281 - val_accuracy: 0.2730 - val_loss: 2.6005
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2273 - loss: 2.7750 - val_accuracy: 0.3110 - val_loss: 2.5181
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2333 - loss: 2.7619 - val_accuracy: 0.2780 - val_loss: 2.6720
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2429 - loss: 2.7584 - val_accuracy: 0.2650 - val_loss: 2.4843
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2329 - loss: 2.7799 - val_accuracy: 0.3230 - val_loss: 2.5149
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2435 - loss: 2.7581 - val_accuracy: 0.2730 - val_loss: 2.4996
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2352 - loss: 2.7745 - val_accuracy: 0.2220 - val_loss: 2.6920
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2404 - loss: 2.7603 - val_accuracy: 0.2830 - val_loss: 2.5264
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2398 - loss: 2.7412 - val_accuracy: 0.3040 - val_loss: 2.5909
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2428 - loss: 2.7566 - val_accuracy: 0.2510 - val_loss: 2.6826
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2343 - loss: 2.7553 - val_accuracy: 0.3430 - val_loss: 2.4751
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2305 - loss: 2.7818 - val_accuracy: 0.2700 - val_loss: 2.6099
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2439 - loss: 2.7522 - val_accuracy: 0.2780 - val_loss: 2.5800
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2436 - loss: 2.7613 - val_accuracy: 0.2710 - val_loss: 2.5340
Accuracy: 0.2660
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
dense_40 (Dense)	(None, 64)	704
batch_normalization_30 (BatchNormalization)	(None, 64)	256
dropout_30 (Dropout)	(None, 64)	0
dense_41 (Dense)	(None, 32)	2,080
batch_normalization_31 (BatchNormalization)	(None, 32)	128
dropout_31 (Dropout)	(None, 32)	0
dense_42 (Dense)	(None, 16)	528
batch_normalization_32 (BatchNormalization)	(None, 16)	64
dropout_32 (Dropout)	(None, 16)	0
dense_43 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

Non-trainable params: 224 (896.00 B)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.2477 - loss: 2.7351 - val_accuracy: 0.2500 - val_loss: 2.5362
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2845 - loss: 2.5206 - val_accuracy: 0.3480 - val_loss: 2.3775
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3032 - loss: 2.4626 - val_accuracy: 0.3440 - val_loss: 2.3317
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3102 - loss: 2.4479 - val_accuracy: 0.3090 - val_loss: 2.3887
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3001 - loss: 2.4552 - val_accuracy: 0.3870 - val_loss: 2.1987
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3006 - loss: 2.4749 - val_accuracy: 0.3000 - val_loss: 2.4069
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3119 - loss: 2.4373 - val_accuracy: 0.3280 - val_loss: 2.3295
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3180 - loss: 2.4270 - val_accuracy: 0.3590 - val_loss: 2.2177
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3157 - loss: 2.4290 - val_accuracy: 0.3440 - val_loss: 2.4158
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3157 - loss: 2.4187 - val_accuracy: 0.2960 - val_loss: 2.4045
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3179 - loss: 2.4183 - val_accuracy: 0.3330 - val_loss: 2.4213
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3116 - loss: 2.4493 - val_accuracy: 0.3810 - val_loss: 2.2185
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3199 - loss: 2.4085 - val_accuracy: 0.3720 - val_loss: 2.2758
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3292 - loss: 2.4165 - val_accuracy: 0.3410 - val_loss: 2.2242
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3180 - loss: 2.4345 - val_accuracy: 0.3670 - val_loss: 2.3247
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3262 - loss: 2.4317 - val_accuracy: 0.3730 - val_loss: 2.2035
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3258 - loss: 2.3953 - val_accuracy: 0.3650 - val_loss: 2.2873
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3344 - loss: 2.3816 - val_accuracy: 0.3880 - val_loss: 2.2130
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3264 - loss: 2.3819 - val_accuracy: 0.3770 - val_loss: 2.3236
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3239 - loss: 2.4261 - val_accuracy: 0.3300 - val_loss: 2.4563
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3181 - loss: 2.4217 - val_accuracy: 0.3460 - val_loss: 2.3602
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3263 - loss: 2.4198 - val_accuracy: 0.3380 - val_loss: 2.3225
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3306 - loss: 2.4035 - val_accuracy: 0.3500 - val_loss: 2.2391
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3297 - loss: 2.3916 - val_accuracy: 0.3460 - val_loss: 2.2275
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3264 - loss: 2.3736 - val_accuracy: 0.2880 - val_loss: 2.5266
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3248 - loss: 2.4328 - val_accuracy: 0.3620 - val_loss: 2.3643
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3267 - loss: 2.3957 - val_accuracy: 0.3610 - val_loss: 2.3261
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3303 - loss: 2.3796 - val_accuracy: 0.4240 - val_loss: 2.2016
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3295 - loss: 2.4039 - val_accuracy: 0.3400 - val_loss: 2.3241
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3310 - loss: 2.3879 - val_accuracy: 0.4150 - val_loss: 2.1654
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3378 - loss: 2.3685 - val_accuracy: 0.3960 - val_loss: 2.2619
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3311 - loss: 2.4027 - val_accuracy: 0.3630 - val_loss: 2.2037
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3195 - loss: 2.4160 - val_accuracy: 0.3420 - val_loss: 2.2759
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3341 - loss: 2.3804 - val_accuracy: 0.4200 - val_loss: 2.1623
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3407 - loss: 2.3877 - val_accuracy: 0.3580 - val_loss: 2.3530
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3391 - loss: 2.3772 - val_accuracy: 0.3890 - val_loss: 2.2293
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3352 - loss: 2.3950 - val_accuracy: 0.3960 - val_loss: 2.2445
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3297 - loss: 2.3993 - val_accuracy: 0.3990 - val_loss: 2.2017
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3407 - loss: 2.4008 - val_accuracy: 0.3870 - val_loss: 2.2296
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3426 - loss: 2.3815 - val_accuracy: 0.4180 - val_loss: 2.2214
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3449 - loss: 2.3565 - val_accuracy: 0.3190 - val_loss: 2.4161
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3348 - loss: 2.3977 - val_accuracy: 0.3090 - val_loss: 2.4745
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3377 - loss: 2.3998 - val_accuracy: 0.3960 - val_loss: 2.2321
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3289 - loss: 2.4219 - val_accuracy: 0.3310 - val_loss: 2.4435
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3416 - loss: 2.3995 - val_accuracy: 0.4230 - val_loss: 2.1697
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3466 - loss: 2.3745 - val_accuracy: 0.3400 - val_loss: 2.4382
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3359 - loss: 2.4237 - val_accuracy: 0.4010 - val_loss: 2.1884
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3368 - loss: 2.3951 - val_accuracy: 0.3470 - val_loss: 2.3362
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3348 - loss: 2.3752 - val_accuracy: 0.3730 - val_loss: 2.2790
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3286 - loss: 2.4287 - val_accuracy: 0.3100 - val_loss: 2.4017
Accuracy: 0.3025
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
dense_44 (Dense)	(None, 64)	704
batch_normalization_33 (BatchNormalization)	(None, 64)	256
dropout_33 (Dropout)	(None, 64)	0
dense_45 (Dense)	(None, 32)	2,080
batch_normalization_34 (BatchNormalization)	(None, 32)	128
dropout_34 (Dropout)	(None, 32)	0
dense_46 (Dense)	(None, 16)	528
batch_normalization_35 (BatchNormalization)	(None, 16)	64
dropout_35 (Dropout)	(None, 16)	0
dense_47 (Dense)	(None, 20)	340

Total params: 4,100 (16.02 KB)

Trainable params: 3,876 (15.14 KB)

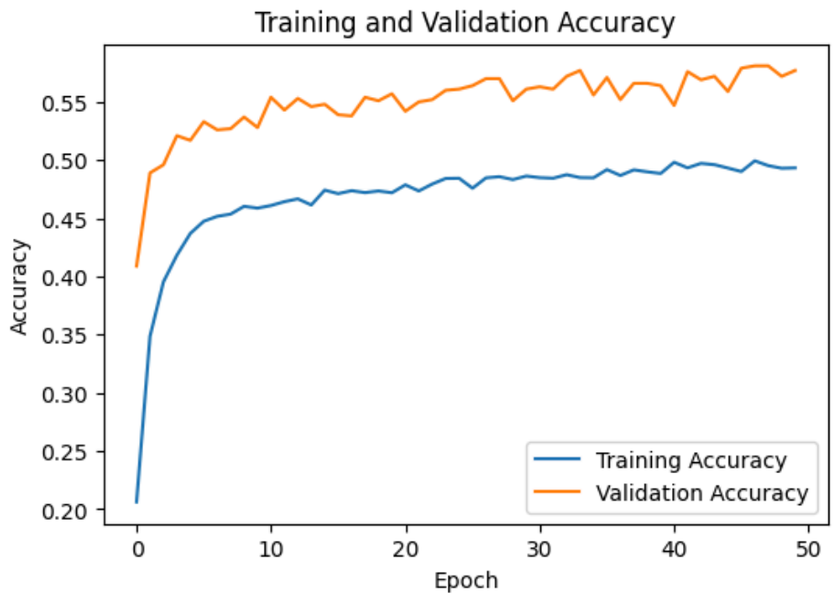
Non-trainable params: 224 (896.00 B)

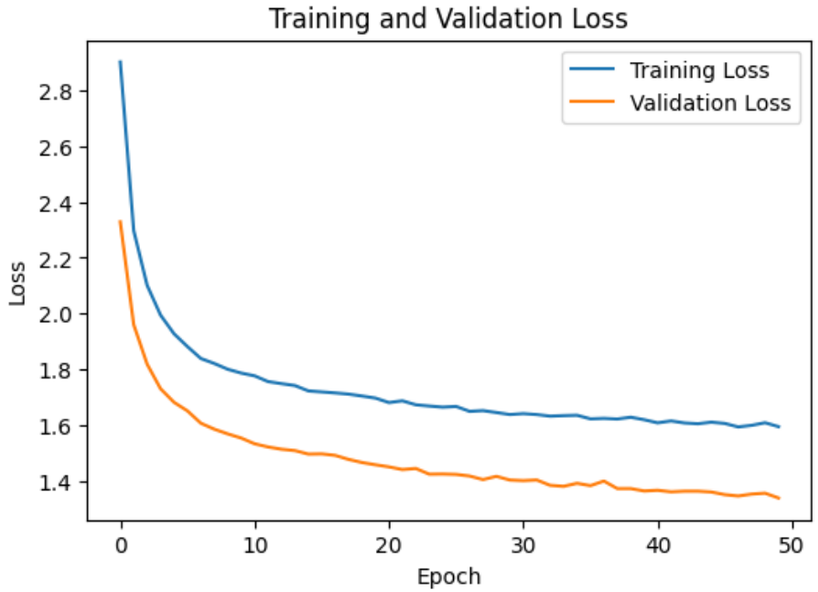
```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.1866 - loss: 2.8724 - val_accuracy: 0.2990 - val_loss: 2.5401
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2392 - loss: 2.6440 - val_accuracy: 0.3100 - val_loss: 2.4303
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2460 - loss: 2.6078 - val_accuracy: 0.2320 - val_loss: 2.5482
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2545 - loss: 2.5872 - val_accuracy: 0.2650 - val_loss: 2.6068
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2586 - loss: 2.5874 - val_accuracy: 0.2080 - val_loss: 2.4833
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2652 - loss: 2.5872 - val_accuracy: 0.2580 - val_loss: 2.5956
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2650 - loss: 2.5930 - val_accuracy: 0.2600 - val_loss: 2.6549
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2677 - loss: 2.5942 - val_accuracy: 0.2740 - val_loss: 2.4427
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2769 - loss: 2.5704 - val_accuracy: 0.2940 - val_loss: 2.5535
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2732 - loss: 2.5621 - val_accuracy: 0.3170 - val_loss: 2.3514
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2699 - loss: 2.5715 - val_accuracy: 0.3440 - val_loss: 2.3544
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2794 - loss: 2.5654 - val_accuracy: 0.3220 - val_loss: 2.3402
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2719 - loss: 2.5497 - val_accuracy: 0.3310 - val_loss: 2.4776
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2755 - loss: 2.5574 - val_accuracy: 0.2590 - val_loss: 2.5057
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2610 - loss: 2.6072 - val_accuracy: 0.3230 - val_loss: 2.3982
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2625 - loss: 2.5887 - val_accuracy: 0.3410 - val_loss: 2.2929
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2871 - loss: 2.5459 - val_accuracy: 0.2860 - val_loss: 2.4631
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2752 - loss: 2.5684 - val_accuracy: 0.3430 - val_loss: 2.2433
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2765 - loss: 2.5703 - val_accuracy: 0.3570 - val_loss: 2.3063
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2817 - loss: 2.5696 - val_accuracy: 0.3180 - val_loss: 2.3903
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2767 - loss: 2.5507 - val_accuracy: 0.3150 - val_loss: 2.3505
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2724 - loss: 2.5632 - val_accuracy: 0.3320 - val_loss: 2.3225
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2695 - loss: 2.5842 - val_accuracy: 0.3320 - val_loss: 2.3743
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2835 - loss: 2.5197 - val_accuracy: 0.3370 - val_loss: 2.2883
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2786 - loss: 2.5605 - val_accuracy: 0.2760 - val_loss: 2.4381
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2793 - loss: 2.5651 - val_accuracy: 0.3420 - val_loss: 2.3890
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2853 - loss: 2.5640 - val_accuracy: 0.3030 - val_loss: 2.4058
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2691 - loss: 2.5670 - val_accuracy: 0.3010 - val_loss: 2.4319
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2775 - loss: 2.5666 - val_accuracy: 0.3330 - val_loss: 2.4557
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2805 - loss: 2.5709 - val_accuracy: 0.2940 - val_loss: 2.3877
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2807 - loss: 2.5284 - val_accuracy: 0.2770 - val_loss: 2.3529
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2777 - loss: 2.5445 - val_accuracy: 0.3180 - val_loss: 2.3842
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2906 - loss: 2.5347 - val_accuracy: 0.3350 - val_loss: 2.3148
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2879 - loss: 2.5244 - val_accuracy: 0.2550 - val_loss: 2.6949
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2837 - loss: 2.5546 - val_accuracy: 0.3540 - val_loss: 2.4304
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2834 - loss: 2.5671 - val_accuracy: 0.3370 - val_loss: 2.4305
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2905 - loss: 2.5689 - val_accuracy: 0.3120 - val_loss: 2.5242
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2886 - loss: 2.5472 - val_accuracy: 0.3490 - val_loss: 2.3298
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2822 - loss: 2.5686 - val_accuracy: 0.2790 - val_loss: 2.4505
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2772 - loss: 2.5633 - val_accuracy: 0.3100 - val_loss: 2.3917
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2794 - loss: 2.5334 - val_accuracy: 0.3240 - val_loss: 2.3657
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2792 - loss: 2.5624 - val_accuracy: 0.3370 - val_loss: 2.4021
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3015 - loss: 2.5305 - val_accuracy: 0.3020 - val_loss: 2.4348
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2878 - loss: 2.5435 - val_accuracy: 0.3590 - val_loss: 2.3192
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2821 - loss: 2.5647 - val_accuracy: 0.3530 - val_loss: 2.3327
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2729 - loss: 2.5642 - val_accuracy: 0.3520 - val_loss: 2.3220
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2757 - loss: 2.5678 - val_accuracy: 0.3530 - val_loss: 2.4151
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2869 - loss: 2.5667 - val_accuracy: 0.2960 - val_loss: 2.4436
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2911 - loss: 2.5429 - val_accuracy: 0.3500 - val_loss: 2.3190
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2889 - loss: 2.5373 - val_accuracy: 0.3230 - val_loss: 2.3578
Accuracy: 0.3170

Best Hyperparameters:
{'learning_rate': 0.001, 'batch_size': 64, 'dropout': 0.2}
Best Accuracy: 0.5740
```

In [ ]:
```
plt.figure(figsize=(6, 4))
plt.plot(best_history.history['accuracy'], label='Training Accuracy')
plt.plot(best_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.figure(figsize=(6, 4))
plt.plot(best_history.history['loss'], label='Training Loss')
plt.plot(best_history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [ ]:
```
best_model.evaluate(X_test, y_test)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5773 - loss: 1.3965
```

Out[ ]:
```
[1.3894131183624268, 0.5740000009536743]
```

6 Layer Neural Network

In [ ]:
```
learning_rates = [0.001, 0.01, 0.1]
batch_sizes = [32, 64]
dropout = [0.1, 0.2, 0.3]
best_accuracy = 0
input_shape = X_train.shape[1:]
best_history = None
best_model = None

for lr in learning_rates:
    for batch in batch_sizes:
        for d in dropout:
            model = Six_Layer_NN()
            config = {
                        'input_shape': input_shape,
                        'epochs': 50,
                        'dropout': d,
                        'batch_size': batch,
                        'lr': lr
                    }
            model.build_model(config)
            history = model.train(X_train, y_train, X_valid, y_valid, config)
            loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
```

```
            print(f"Accuracy: {accuracy:.4f}")

            if accuracy > best_accuracy:
                best_model = model
                best_accuracy = accuracy
                best_params = {'learning_rate': lr, 'batch_size': batch, 'dropout': d}
                best_history = history

print("\nBest Hyperparameters:")
print(best_params)
print(f"Best Accuracy: {best_accuracy:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_12"

Layer (type)	Output Shape	Param #
dense_48 (Dense)	(None, 32)	352
batch_normalization_36 (BatchNormalization)	(None, 32)	128
dropout_36 (Dropout)	(None, 32)	0
dense_49 (Dense)	(None, 64)	2,112
batch_normalization_37 (BatchNormalization)	(None, 64)	256
dropout_37 (Dropout)	(None, 64)	0
dense_50 (Dense)	(None, 32)	2,080
batch_normalization_38 (BatchNormalization)	(None, 32)	128
dropout_38 (Dropout)	(None, 32)	0
dense_51 (Dense)	(None, 16)	528
batch_normalization_39 (BatchNormalization)	(None, 16)	64
dropout_39 (Dropout)	(None, 16)	0
dense_52 (Dense)	(None, 32)	544
batch_normalization_40 (BatchNormalization)	(None, 32)	128
dropout_40 (Dropout)	(None, 32)	0
dense_53 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.1728 - loss: 3.1767 - val_accuracy: 0.4760 - val_loss: 1.9919
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.3832 - loss: 2.2027 - val_accuracy: 0.5140 - val_loss: 1.7549
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4171 - loss: 2.0012 - val_accuracy: 0.5250 - val_loss: 1.6284
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4518 - loss: 1.8686 - val_accuracy: 0.5540 - val_loss: 1.5688
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4622 - loss: 1.8054 - val_accuracy: 0.5430 - val_loss: 1.5236
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4661 - loss: 1.7732 - val_accuracy: 0.5600 - val_loss: 1.4839
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4709 - loss: 1.7483 - val_accuracy: 0.5610 - val_loss: 1.4734
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4726 - loss: 1.7248 - val_accuracy: 0.5560 - val_loss: 1.4500
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4798 - loss: 1.6951 - val_accuracy: 0.5630 - val_loss: 1.4161
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4798 - loss: 1.6841 - val_accuracy: 0.5730 - val_loss: 1.4094
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4921 - loss: 1.6530 - val_accuracy: 0.5690 - val_loss: 1.3993
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4978 - loss: 1.6210 - val_accuracy: 0.5830 - val_loss: 1.3720
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4928 - loss: 1.6468 - val_accuracy: 0.5700 - val_loss: 1.3972
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4983 - loss: 1.6153 - val_accuracy: 0.5720 - val_loss: 1.3650
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4948 - loss: 1.6286 - val_accuracy: 0.5790 - val_loss: 1.3359
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5001 - loss: 1.6063 - val_accuracy: 0.5820 - val_loss: 1.3365
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4939 - loss: 1.6107 - val_accuracy: 0.5830 - val_loss: 1.3488
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5067 - loss: 1.5957 - val_accuracy: 0.5880 - val_loss: 1.3131
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5017 - loss: 1.5910 - val_accuracy: 0.5880 - val_loss: 1.3158
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5040 - loss: 1.5936 - val_accuracy: 0.5990 - val_loss: 1.2954
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5157 - loss: 1.5720 - val_accuracy: 0.5940 - val_loss: 1.3014
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5115 - loss: 1.5623 - val_accuracy: 0.5840 - val_loss: 1.3052
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5082 - loss: 1.5662 - val_accuracy: 0.5900 - val_loss: 1.3023
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5102 - loss: 1.5562 - val_accuracy: 0.5900 - val_loss: 1.2866
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5096 - loss: 1.5678 - val_accuracy: 0.5980 - val_loss: 1.2887
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5123 - loss: 1.5561 - val_accuracy: 0.5980 - val_loss: 1.2792
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5064 - loss: 1.5560 - val_accuracy: 0.6000 - val_loss: 1.2864
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5073 - loss: 1.5624 - val_accuracy: 0.5910 - val_loss: 1.3030
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5069 - loss: 1.5643 - val_accuracy: 0.5990 - val_loss: 1.2670
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5134 - loss: 1.5611 - val_accuracy: 0.5990 - val_loss: 1.2743
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5150 - loss: 1.5526 - val_accuracy: 0.6020 - val_loss: 1.2816
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5154 - loss: 1.5503 - val_accuracy: 0.5860 - val_loss: 1.2906
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5085 - loss: 1.5455 - val_accuracy: 0.6090 - val_loss: 1.2615
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5155 - loss: 1.5419 - val_accuracy: 0.5910 - val_loss: 1.2904
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5167 - loss: 1.5485 - val_accuracy: 0.6040 - val_loss: 1.2680
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5216 - loss: 1.5250 - val_accuracy: 0.6020 - val_loss: 1.2783
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5180 - loss: 1.5207 - val_accuracy: 0.5940 - val_loss: 1.2515
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5201 - loss: 1.5235 - val_accuracy: 0.5960 - val_loss: 1.2687
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5249 - loss: 1.5069 - val_accuracy: 0.6110 - val_loss: 1.2476
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5253 - loss: 1.5244 - val_accuracy: 0.5900 - val_loss: 1.2621
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5087 - loss: 1.5458 - val_accuracy: 0.6100 - val_loss: 1.2608
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5233 - loss: 1.5273 - val_accuracy: 0.5990 - val_loss: 1.2457
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.5286 - loss: 1.5287 - val_accuracy: 0.5960 - val_loss: 1.2589
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5107 - loss: 1.5364 - val_accuracy: 0.5980 - val_loss: 1.2446
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5183 - loss: 1.5199 - val_accuracy: 0.6080 - val_loss: 1.2530
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5261 - loss: 1.5084 - val_accuracy: 0.6100 - val_loss: 1.2457
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5267 - loss: 1.5024 - val_accuracy: 0.6190 - val_loss: 1.2321
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5219 - loss: 1.5141 - val_accuracy: 0.6090 - val_loss: 1.2561
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5305 - loss: 1.4878 - val_accuracy: 0.6110 - val_loss: 1.2461
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5190 - loss: 1.5207 - val_accuracy: 0.6170 - val_loss: 1.2292
Accuracy: 0.5920
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_13"

Layer (type)	Output Shape	Param #
dense_54 (Dense)	(None, 32)	352
batch_normalization_41 (BatchNormalization)	(None, 32)	128
dropout_41 (Dropout)	(None, 32)	0
dense_55 (Dense)	(None, 64)	2,112
batch_normalization_42 (BatchNormalization)	(None, 64)	256
dropout_42 (Dropout)	(None, 64)	0
dense_56 (Dense)	(None, 32)	2,080
batch_normalization_43 (BatchNormalization)	(None, 32)	128
dropout_43 (Dropout)	(None, 32)	0
dense_57 (Dense)	(None, 16)	528
batch_normalization_44 (BatchNormalization)	(None, 16)	64
dropout_44 (Dropout)	(None, 16)	0
dense_58 (Dense)	(None, 32)	544
batch_normalization_45 (BatchNormalization)	(None, 32)	128
dropout_45 (Dropout)	(None, 32)	0
dense_59 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.1133 - loss: 3.3656 - val_accuracy: 0.4180 - val_loss: 2.2183
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2737 - loss: 2.4675 - val_accuracy: 0.4400 - val_loss: 1.9275
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3263 - loss: 2.2238 - val_accuracy: 0.4820 - val_loss: 1.7790
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3628 - loss: 2.0941 - val_accuracy: 0.4840 - val_loss: 1.7022
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3866 - loss: 2.0151 - val_accuracy: 0.4990 - val_loss: 1.6378
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3972 - loss: 1.9724 - val_accuracy: 0.5090 - val_loss: 1.6038
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4001 - loss: 1.9422 - val_accuracy: 0.4950 - val_loss: 1.5861
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4004 - loss: 1.9239 - val_accuracy: 0.5060 - val_loss: 1.5472
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4082 - loss: 1.8993 - val_accuracy: 0.5320 - val_loss: 1.5423
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4230 - loss: 1.8779 - val_accuracy: 0.5240 - val_loss: 1.5125
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4263 - loss: 1.8508 - val_accuracy: 0.5130 - val_loss: 1.5297
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4290 - loss: 1.8428 - val_accuracy: 0.5400 - val_loss: 1.4900
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4386 - loss: 1.8261 - val_accuracy: 0.5270 - val_loss: 1.4934
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4252 - loss: 1.8415 - val_accuracy: 0.5340 - val_loss: 1.4796
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4378 - loss: 1.8085 - val_accuracy: 0.5430 - val_loss: 1.4599
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4469 - loss: 1.7906 - val_accuracy: 0.5370 - val_loss: 1.4661
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4480 - loss: 1.7877 - val_accuracy: 0.5410 - val_loss: 1.4600
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4531 - loss: 1.7842 - val_accuracy: 0.5290 - val_loss: 1.4540
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4514 - loss: 1.7643 - val_accuracy: 0.5320 - val_loss: 1.4511
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4520 - loss: 1.7546 - val_accuracy: 0.5510 - val_loss: 1.4347
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4548 - loss: 1.7522 - val_accuracy: 0.5460 - val_loss: 1.4288
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4542 - loss: 1.7804 - val_accuracy: 0.5460 - val_loss: 1.4239
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4480 - loss: 1.7783 - val_accuracy: 0.5490 - val_loss: 1.4285
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4571 - loss: 1.7594 - val_accuracy: 0.5550 - val_loss: 1.4273
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4603 - loss: 1.7458 - val_accuracy: 0.5490 - val_loss: 1.4228
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4568 - loss: 1.7480 - val_accuracy: 0.5330 - val_loss: 1.4128
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4577 - loss: 1.7219 - val_accuracy: 0.5530 - val_loss: 1.4118
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4533 - loss: 1.7572 - val_accuracy: 0.5580 - val_loss: 1.4034
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4591 - loss: 1.7421 - val_accuracy: 0.5670 - val_loss: 1.3910
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4574 - loss: 1.7404 - val_accuracy: 0.5710 - val_loss: 1.3875
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4620 - loss: 1.7287 - val_accuracy: 0.5570 - val_loss: 1.3924
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4550 - loss: 1.7481 - val_accuracy: 0.5600 - val_loss: 1.3844
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4598 - loss: 1.7244 - val_accuracy: 0.5560 - val_loss: 1.3756
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4650 - loss: 1.7214 - val_accuracy: 0.5720 - val_loss: 1.3900
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4619 - loss: 1.7336 - val_accuracy: 0.5590 - val_loss: 1.3828
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4635 - loss: 1.7127 - val_accuracy: 0.5660 - val_loss: 1.3736
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4672 - loss: 1.7201 - val_accuracy: 0.5620 - val_loss: 1.3778
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4703 - loss: 1.6934 - val_accuracy: 0.5680 - val_loss: 1.3694
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4574 - loss: 1.7265 - val_accuracy: 0.5680 - val_loss: 1.3633
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4692 - loss: 1.6910 - val_accuracy: 0.5710 - val_loss: 1.3610
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4664 - loss: 1.7051 - val_accuracy: 0.5640 - val_loss: 1.3939
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4712 - loss: 1.7118 - val_accuracy: 0.5650 - val_loss: 1.3553
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4604 - loss: 1.7151 - val_accuracy: 0.5740 - val_loss: 1.3590
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4668 - loss: 1.7068 - val_accuracy: 0.5760 - val_loss: 1.3554
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4727 - loss: 1.6923 - val_accuracy: 0.5760 - val_loss: 1.3612
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4725 - loss: 1.6913 - val_accuracy: 0.5750 - val_loss: 1.3541
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4610 - loss: 1.6965 - val_accuracy: 0.5790 - val_loss: 1.3370
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4690 - loss: 1.6901 - val_accuracy: 0.5730 - val_loss: 1.3391
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4649 - loss: 1.6992 - val_accuracy: 0.5620 - val_loss: 1.3614
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4717 - loss: 1.6920 - val_accuracy: 0.5670 - val_loss: 1.3556
Accuracy: 0.5700
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_14"

Layer (type)	Output Shape	Param #
dense_60 (Dense)	(None, 32)	352
batch_normalization_46 (BatchNormalization)	(None, 32)	128
dropout_46 (Dropout)	(None, 32)	0
dense_61 (Dense)	(None, 64)	2,112
batch_normalization_47 (BatchNormalization)	(None, 64)	256
dropout_47 (Dropout)	(None, 64)	0
dense_62 (Dense)	(None, 32)	2,080
batch_normalization_48 (BatchNormalization)	(None, 32)	128
dropout_48 (Dropout)	(None, 32)	0
dense_63 (Dense)	(None, 16)	528
batch_normalization_49 (BatchNormalization)	(None, 16)	64
dropout_49 (Dropout)	(None, 16)	0
dense_64 (Dense)	(None, 32)	544
batch_normalization_50 (BatchNormalization)	(None, 32)	128
dropout_50 (Dropout)	(None, 32)	0
dense_65 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.0905 - loss: 3.5267 - val_accuracy: 0.3750 - val_loss: 2.4199
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2122 - loss: 2.6798 - val_accuracy: 0.4120 - val_loss: 2.1326
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2660 - loss: 2.4104 - val_accuracy: 0.4310 - val_loss: 1.9700
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3000 - loss: 2.2771 - val_accuracy: 0.4540 - val_loss: 1.8616
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3130 - loss: 2.1935 - val_accuracy: 0.4720 - val_loss: 1.7844
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3324 - loss: 2.1359 - val_accuracy: 0.4790 - val_loss: 1.7513
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3362 - loss: 2.1085 - val_accuracy: 0.4830 - val_loss: 1.7175
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3445 - loss: 2.0779 - val_accuracy: 0.4750 - val_loss: 1.6989
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3535 - loss: 2.0606 - val_accuracy: 0.4690 - val_loss: 1.6808
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3542 - loss: 2.0462 - val_accuracy: 0.4800 - val_loss: 1.6597
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3572 - loss: 2.0280 - val_accuracy: 0.4890 - val_loss: 1.6267
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3696 - loss: 2.0015 - val_accuracy: 0.4880 - val_loss: 1.6339
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3720 - loss: 1.9900 - val_accuracy: 0.4840 - val_loss: 1.6205
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3786 - loss: 1.9806 - val_accuracy: 0.4850 - val_loss: 1.6077
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3724 - loss: 1.9797 - val_accuracy: 0.4800 - val_loss: 1.6123
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3762 - loss: 1.9869 - val_accuracy: 0.4830 - val_loss: 1.5905
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3822 - loss: 1.9642 - val_accuracy: 0.4970 - val_loss: 1.5964
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3835 - loss: 1.9519 - val_accuracy: 0.5040 - val_loss: 1.5847
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3790 - loss: 1.9564 - val_accuracy: 0.4880 - val_loss: 1.5959
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3847 - loss: 1.9356 - val_accuracy: 0.4870 - val_loss: 1.5785
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3861 - loss: 1.9282 - val_accuracy: 0.4950 - val_loss: 1.5748
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3881 - loss: 1.9257 - val_accuracy: 0.5020 - val_loss: 1.5707
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3957 - loss: 1.9367 - val_accuracy: 0.5050 - val_loss: 1.5563
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3965 - loss: 1.9160 - val_accuracy: 0.5000 - val_loss: 1.5606
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3982 - loss: 1.8990 - val_accuracy: 0.5110 - val_loss: 1.5554
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3946 - loss: 1.9259 - val_accuracy: 0.4970 - val_loss: 1.5474
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3938 - loss: 1.9232 - val_accuracy: 0.5120 - val_loss: 1.5549
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3943 - loss: 1.9179 - val_accuracy: 0.5270 - val_loss: 1.5394
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4058 - loss: 1.9104 - val_accuracy: 0.5190 - val_loss: 1.5432
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3979 - loss: 1.9106 - val_accuracy: 0.5050 - val_loss: 1.5475
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3953 - loss: 1.9214 - val_accuracy: 0.5190 - val_loss: 1.5238
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4036 - loss: 1.8932 - val_accuracy: 0.5060 - val_loss: 1.5217
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3995 - loss: 1.8997 - val_accuracy: 0.5270 - val_loss: 1.5295
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4097 - loss: 1.8928 - val_accuracy: 0.5260 - val_loss: 1.5147
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4115 - loss: 1.8829 - val_accuracy: 0.5170 - val_loss: 1.5158
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4023 - loss: 1.8896 - val_accuracy: 0.5340 - val_loss: 1.5295
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4054 - loss: 1.8669 - val_accuracy: 0.5120 - val_loss: 1.5122
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4052 - loss: 1.8985 - val_accuracy: 0.5150 - val_loss: 1.5095
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4026 - loss: 1.8832 - val_accuracy: 0.5050 - val_loss: 1.5220
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4021 - loss: 1.8806 - val_accuracy: 0.5220 - val_loss: 1.5176
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4032 - loss: 1.8687 - val_accuracy: 0.5330 - val_loss: 1.5070
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4121 - loss: 1.8787 - val_accuracy: 0.5310 - val_loss: 1.5143
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4118 - loss: 1.8720 - val_accuracy: 0.5140 - val_loss: 1.5114
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4159 - loss: 1.8825 - val_accuracy: 0.5230 - val_loss: 1.4943
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4093 - loss: 1.8807 - val_accuracy: 0.5210 - val_loss: 1.5199
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4135 - loss: 1.8721 - val_accuracy: 0.5200 - val_loss: 1.5102
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4120 - loss: 1.8814 - val_accuracy: 0.5230 - val_loss: 1.4820
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4061 - loss: 1.8726 - val_accuracy: 0.5110 - val_loss: 1.5068
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4073 - loss: 1.8895 - val_accuracy: 0.5140 - val_loss: 1.5039
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4055 - loss: 1.8884 - val_accuracy: 0.5100 - val_loss: 1.5018
Accuracy: 0.5335
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_15"

Layer (type)	Output Shape	Param #
dense_66 (Dense)	(None, 32)	352
batch_normalization_51 (BatchNormalization)	(None, 32)	128
dropout_51 (Dropout)	(None, 32)	0
dense_67 (Dense)	(None, 64)	2,112
batch_normalization_52 (BatchNormalization)	(None, 64)	256
dropout_52 (Dropout)	(None, 64)	0
dense_68 (Dense)	(None, 32)	2,080
batch_normalization_53 (BatchNormalization)	(None, 32)	128
dropout_53 (Dropout)	(None, 32)	0
dense_69 (Dense)	(None, 16)	528
batch_normalization_54 (BatchNormalization)	(None, 16)	64
dropout_54 (Dropout)	(None, 16)	0
dense_70 (Dense)	(None, 32)	544
batch_normalization_55 (BatchNormalization)	(None, 32)	128
dropout_55 (Dropout)	(None, 32)	0
dense_71 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.1526 - loss: 3.2465 - val_accuracy: 0.4210 - val_loss: 2.2212
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3418 - loss: 2.3274 - val_accuracy: 0.5030 - val_loss: 1.8427
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4129 - loss: 2.0592 - val_accuracy: 0.5220 - val_loss: 1.6936
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4421 - loss: 1.9251 - val_accuracy: 0.5310 - val_loss: 1.6186
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4697 - loss: 1.8221 - val_accuracy: 0.5520 - val_loss: 1.5512
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4760 - loss: 1.7814 - val_accuracy: 0.5620 - val_loss: 1.5109
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4797 - loss: 1.7375 - val_accuracy: 0.5620 - val_loss: 1.4833
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4928 - loss: 1.7032 - val_accuracy: 0.5730 - val_loss: 1.4616
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4937 - loss: 1.6667 - val_accuracy: 0.5680 - val_loss: 1.4327
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4880 - loss: 1.6706 - val_accuracy: 0.5660 - val_loss: 1.4150
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5053 - loss: 1.6289 - val_accuracy: 0.5690 - val_loss: 1.3976
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5075 - loss: 1.6145 - val_accuracy: 0.5720 - val_loss: 1.3887
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5058 - loss: 1.6078 - val_accuracy: 0.5880 - val_loss: 1.3756
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5138 - loss: 1.5922 - val_accuracy: 0.5560 - val_loss: 1.3739
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5105 - loss: 1.5930 - val_accuracy: 0.5860 - val_loss: 1.3387
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5096 - loss: 1.5796 - val_accuracy: 0.5750 - val_loss: 1.3423
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5204 - loss: 1.5708 - val_accuracy: 0.5820 - val_loss: 1.3390
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5140 - loss: 1.5658 - val_accuracy: 0.5720 - val_loss: 1.3363
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5152 - loss: 1.5509 - val_accuracy: 0.5900 - val_loss: 1.3311
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5179 - loss: 1.5448 - val_accuracy: 0.5880 - val_loss: 1.3120
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5256 - loss: 1.5297 - val_accuracy: 0.5890 - val_loss: 1.3089
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5030 - loss: 1.5681 - val_accuracy: 0.5850 - val_loss: 1.3088
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5262 - loss: 1.5245 - val_accuracy: 0.5850 - val_loss: 1.3067
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5179 - loss: 1.5333 - val_accuracy: 0.5940 - val_loss: 1.3065
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5346 - loss: 1.5026 - val_accuracy: 0.6020 - val_loss: 1.2902
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5319 - loss: 1.5058 - val_accuracy: 0.5930 - val_loss: 1.2858
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5261 - loss: 1.5168 - val_accuracy: 0.6100 - val_loss: 1.2744
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5383 - loss: 1.4835 - val_accuracy: 0.6050 - val_loss: 1.2702
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5297 - loss: 1.5024 - val_accuracy: 0.5950 - val_loss: 1.2743
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5284 - loss: 1.4919 - val_accuracy: 0.5990 - val_loss: 1.2883
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5258 - loss: 1.4979 - val_accuracy: 0.6070 - val_loss: 1.2710
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5231 - loss: 1.5000 - val_accuracy: 0.6030 - val_loss: 1.2664
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5372 - loss: 1.4927 - val_accuracy: 0.6000 - val_loss: 1.2739
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5364 - loss: 1.4784 - val_accuracy: 0.6060 - val_loss: 1.2585
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5406 - loss: 1.4627 - val_accuracy: 0.6030 - val_loss: 1.2561
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5364 - loss: 1.4813 - val_accuracy: 0.6110 - val_loss: 1.2510
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5339 - loss: 1.4883 - val_accuracy: 0.6140 - val_loss: 1.2457
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5347 - loss: 1.4799 - val_accuracy: 0.6090 - val_loss: 1.2360
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5393 - loss: 1.4630 - val_accuracy: 0.6040 - val_loss: 1.2499
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5376 - loss: 1.4648 - val_accuracy: 0.6130 - val_loss: 1.2423
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5381 - loss: 1.4660 - val_accuracy: 0.6120 - val_loss: 1.2497
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5327 - loss: 1.4832 - val_accuracy: 0.6040 - val_loss: 1.2431
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5329 - loss: 1.4675 - val_accuracy: 0.5960 - val_loss: 1.2356
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5374 - loss: 1.4689 - val_accuracy: 0.5980 - val_loss: 1.2455
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5332 - loss: 1.4700 - val_accuracy: 0.6050 - val_loss: 1.2322
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5461 - loss: 1.4451 - val_accuracy: 0.6050 - val_loss: 1.2404
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5404 - loss: 1.4614 - val_accuracy: 0.6150 - val_loss: 1.2096
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5543 - loss: 1.4359 - val_accuracy: 0.5980 - val_loss: 1.2318
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5421 - loss: 1.4487 - val_accuracy: 0.6050 - val_loss: 1.2304
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5466 - loss: 1.4390 - val_accuracy: 0.6090 - val_loss: 1.2198
Accuracy: 0.6040
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_16"

Layer (type)	Output Shape	Param #
dense_72 (Dense)	(None, 32)	352
batch_normalization_56 (BatchNormalization)	(None, 32)	128
dropout_56 (Dropout)	(None, 32)	0
dense_73 (Dense)	(None, 64)	2,112
batch_normalization_57 (BatchNormalization)	(None, 64)	256
dropout_57 (Dropout)	(None, 64)	0
dense_74 (Dense)	(None, 32)	2,080
batch_normalization_58 (BatchNormalization)	(None, 32)	128
dropout_58 (Dropout)	(None, 32)	0
dense_75 (Dense)	(None, 16)	528
batch_normalization_59 (BatchNormalization)	(None, 16)	64
dropout_59 (Dropout)	(None, 16)	0
dense_76 (Dense)	(None, 32)	544
batch_normalization_60 (BatchNormalization)	(None, 32)	128
dropout_60 (Dropout)	(None, 32)	0
dense_77 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 7s 7ms/step - accuracy: 0.1087 - loss: 3.4148 - val_accuracy: 0.3770 - val_loss: 2.3422
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2675 - loss: 2.5476 - val_accuracy: 0.4360 - val_loss: 2.0126
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3304 - loss: 2.2788 - val_accuracy: 0.4900 - val_loss: 1.8436
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3590 - loss: 2.1385 - val_accuracy: 0.5010 - val_loss: 1.7534
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3876 - loss: 2.0366 - val_accuracy: 0.5030 - val_loss: 1.6913
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3962 - loss: 1.9705 - val_accuracy: 0.5050 - val_loss: 1.6440
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4091 - loss: 1.9502 - val_accuracy: 0.5230 - val_loss: 1.6051
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4262 - loss: 1.8945 - val_accuracy: 0.5180 - val_loss: 1.5825
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4311 - loss: 1.8761 - val_accuracy: 0.5440 - val_loss: 1.5432
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4365 - loss: 1.8515 - val_accuracy: 0.5380 - val_loss: 1.5267
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4453 - loss: 1.8291 - val_accuracy: 0.5410 - val_loss: 1.5254
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4445 - loss: 1.8106 - val_accuracy: 0.5340 - val_loss: 1.5079
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4258 - loss: 1.8300 - val_accuracy: 0.5380 - val_loss: 1.4944
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4418 - loss: 1.8096 - val_accuracy: 0.5490 - val_loss: 1.4744
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4548 - loss: 1.7678 - val_accuracy: 0.5490 - val_loss: 1.4691
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4440 - loss: 1.7803 - val_accuracy: 0.5400 - val_loss: 1.4631
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4473 - loss: 1.7648 - val_accuracy: 0.5430 - val_loss: 1.4615
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4506 - loss: 1.7707 - val_accuracy: 0.5490 - val_loss: 1.4513
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4612 - loss: 1.7255 - val_accuracy: 0.5480 - val_loss: 1.4422
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4564 - loss: 1.7536 - val_accuracy: 0.5510 - val_loss: 1.4414
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4575 - loss: 1.7342 - val_accuracy: 0.5670 - val_loss: 1.4311
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4530 - loss: 1.7320 - val_accuracy: 0.5530 - val_loss: 1.4303
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4671 - loss: 1.7297 - val_accuracy: 0.5420 - val_loss: 1.4411
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4594 - loss: 1.7280 - val_accuracy: 0.5640 - val_loss: 1.4086
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4726 - loss: 1.7143 - val_accuracy: 0.5640 - val_loss: 1.4110
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4641 - loss: 1.7201 - val_accuracy: 0.5500 - val_loss: 1.4059
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4660 - loss: 1.7228 - val_accuracy: 0.5700 - val_loss: 1.4009
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4641 - loss: 1.6975 - val_accuracy: 0.5620 - val_loss: 1.3919
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4730 - loss: 1.6837 - val_accuracy: 0.5500 - val_loss: 1.3908
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4598 - loss: 1.7138 - val_accuracy: 0.5590 - val_loss: 1.3905
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4697 - loss: 1.7008 - val_accuracy: 0.5680 - val_loss: 1.3847
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4684 - loss: 1.7012 - val_accuracy: 0.5740 - val_loss: 1.3766
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4761 - loss: 1.6828 - val_accuracy: 0.5750 - val_loss: 1.3740
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4653 - loss: 1.6984 - val_accuracy: 0.5670 - val_loss: 1.3610
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4703 - loss: 1.6768 - val_accuracy: 0.5740 - val_loss: 1.3636
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4658 - loss: 1.7037 - val_accuracy: 0.5590 - val_loss: 1.3764
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4727 - loss: 1.6802 - val_accuracy: 0.5490 - val_loss: 1.3745
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4807 - loss: 1.6589 - val_accuracy: 0.5730 - val_loss: 1.3699
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4819 - loss: 1.6623 - val_accuracy: 0.5660 - val_loss: 1.3590
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4753 - loss: 1.6728 - val_accuracy: 0.5820 - val_loss: 1.3521
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4771 - loss: 1.6648 - val_accuracy: 0.5860 - val_loss: 1.3432
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4804 - loss: 1.6653 - val_accuracy: 0.5700 - val_loss: 1.3514
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4777 - loss: 1.6649 - val_accuracy: 0.5840 - val_loss: 1.3469
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4747 - loss: 1.6575 - val_accuracy: 0.5840 - val_loss: 1.3426
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4708 - loss: 1.6833 - val_accuracy: 0.5760 - val_loss: 1.3515
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4848 - loss: 1.6399 - val_accuracy: 0.5750 - val_loss: 1.3410
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4754 - loss: 1.6596 - val_accuracy: 0.5660 - val_loss: 1.3354
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4824 - loss: 1.6548 - val_accuracy: 0.5770 - val_loss: 1.3406
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4724 - loss: 1.6571 - val_accuracy: 0.5780 - val_loss: 1.3303
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4958 - loss: 1.6334 - val_accuracy: 0.5750 - val_loss: 1.3312
Accuracy: 0.5760
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_17"

Layer (type)	Output Shape	Param #
dense_78 (Dense)	(None, 32)	352
batch_normalization_61 (BatchNormalization)	(None, 32)	128
dropout_61 (Dropout)	(None, 32)	0
dense_79 (Dense)	(None, 64)	2,112
batch_normalization_62 (BatchNormalization)	(None, 64)	256
dropout_62 (Dropout)	(None, 64)	0
dense_80 (Dense)	(None, 32)	2,080
batch_normalization_63 (BatchNormalization)	(None, 32)	128
dropout_63 (Dropout)	(None, 32)	0
dense_81 (Dense)	(None, 16)	528
batch_normalization_64 (BatchNormalization)	(None, 16)	64
dropout_64 (Dropout)	(None, 16)	0
dense_82 (Dense)	(None, 32)	544
batch_normalization_65 (BatchNormalization)	(None, 32)	128
dropout_65 (Dropout)	(None, 32)	0
dense_83 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.0885 - loss: 3.5926 - val_accuracy: 0.3470 - val_loss: 2.5711
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1887 - loss: 2.7979 - val_accuracy: 0.4050 - val_loss: 2.2326
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2444 - loss: 2.5173 - val_accuracy: 0.4110 - val_loss: 2.0707
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2846 - loss: 2.3594 - val_accuracy: 0.4240 - val_loss: 1.9597
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3081 - loss: 2.2377 - val_accuracy: 0.4440 - val_loss: 1.8785
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3217 - loss: 2.1771 - val_accuracy: 0.4430 - val_loss: 1.8194
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3174 - loss: 2.1667 - val_accuracy: 0.4490 - val_loss: 1.7673
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3459 - loss: 2.0899 - val_accuracy: 0.4750 - val_loss: 1.7423
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3449 - loss: 2.0859 - val_accuracy: 0.4780 - val_loss: 1.7126
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3547 - loss: 2.0476 - val_accuracy: 0.4770 - val_loss: 1.6932
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3597 - loss: 2.0409 - val_accuracy: 0.4810 - val_loss: 1.6883
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3640 - loss: 2.0106 - val_accuracy: 0.4890 - val_loss: 1.6627
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3670 - loss: 2.0021 - val_accuracy: 0.4970 - val_loss: 1.6410
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3758 - loss: 1.9585 - val_accuracy: 0.5000 - val_loss: 1.6306
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3706 - loss: 1.9775 - val_accuracy: 0.4930 - val_loss: 1.6147
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3816 - loss: 1.9487 - val_accuracy: 0.5010 - val_loss: 1.6073
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3960 - loss: 1.9303 - val_accuracy: 0.4950 - val_loss: 1.5926
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3817 - loss: 1.9329 - val_accuracy: 0.5120 - val_loss: 1.5857
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3825 - loss: 1.9437 - val_accuracy: 0.5010 - val_loss: 1.5829
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3858 - loss: 1.9318 - val_accuracy: 0.5090 - val_loss: 1.5768
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3937 - loss: 1.9324 - val_accuracy: 0.5050 - val_loss: 1.5755
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3902 - loss: 1.9223 - val_accuracy: 0.4930 - val_loss: 1.5711
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3948 - loss: 1.9134 - val_accuracy: 0.5110 - val_loss: 1.5618
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3940 - loss: 1.8951 - val_accuracy: 0.5210 - val_loss: 1.5471
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4028 - loss: 1.8935 - val_accuracy: 0.5070 - val_loss: 1.5454
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4042 - loss: 1.8850 - val_accuracy: 0.5220 - val_loss: 1.5318
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4067 - loss: 1.8961 - val_accuracy: 0.5160 - val_loss: 1.5389
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4054 - loss: 1.8833 - val_accuracy: 0.5220 - val_loss: 1.5406
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4040 - loss: 1.8829 - val_accuracy: 0.5160 - val_loss: 1.5248
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4105 - loss: 1.8556 - val_accuracy: 0.5270 - val_loss: 1.5318
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4155 - loss: 1.8642 - val_accuracy: 0.5330 - val_loss: 1.5216
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4134 - loss: 1.8698 - val_accuracy: 0.5180 - val_loss: 1.5220
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4162 - loss: 1.8522 - val_accuracy: 0.5190 - val_loss: 1.5138
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4157 - loss: 1.8526 - val_accuracy: 0.5330 - val_loss: 1.5207
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4164 - loss: 1.8455 - val_accuracy: 0.5300 - val_loss: 1.5060
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4193 - loss: 1.8613 - val_accuracy: 0.5360 - val_loss: 1.5131
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4213 - loss: 1.8466 - val_accuracy: 0.5350 - val_loss: 1.4955
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4060 - loss: 1.8542 - val_accuracy: 0.5270 - val_loss: 1.4872
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4224 - loss: 1.8457 - val_accuracy: 0.5070 - val_loss: 1.4993
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4198 - loss: 1.8366 - val_accuracy: 0.5200 - val_loss: 1.4867
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4173 - loss: 1.8581 - val_accuracy: 0.5250 - val_loss: 1.4912
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4266 - loss: 1.8325 - val_accuracy: 0.5270 - val_loss: 1.4954
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4210 - loss: 1.8393 - val_accuracy: 0.5340 - val_loss: 1.4938
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4210 - loss: 1.8448 - val_accuracy: 0.5310 - val_loss: 1.4770
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4245 - loss: 1.8364 - val_accuracy: 0.5240 - val_loss: 1.4789
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4221 - loss: 1.8251 - val_accuracy: 0.5390 - val_loss: 1.4816
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4268 - loss: 1.8189 - val_accuracy: 0.5250 - val_loss: 1.4778
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4223 - loss: 1.8365 - val_accuracy: 0.5320 - val_loss: 1.4700
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4216 - loss: 1.8317 - val_accuracy: 0.5360 - val_loss: 1.4732
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4239 - loss: 1.8183 - val_accuracy: 0.5370 - val_loss: 1.4758
Accuracy: 0.5450
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_18"

Layer (type)	Output Shape	Param #
dense_84 (Dense)	(None, 32)	352
batch_normalization_66 (BatchNormalization)	(None, 32)	128
dropout_66 (Dropout)	(None, 32)	0
dense_85 (Dense)	(None, 64)	2,112
batch_normalization_67 (BatchNormalization)	(None, 64)	256
dropout_67 (Dropout)	(None, 64)	0
dense_86 (Dense)	(None, 32)	2,080
batch_normalization_68 (BatchNormalization)	(None, 32)	128
dropout_68 (Dropout)	(None, 32)	0
dense_87 (Dense)	(None, 16)	528
batch_normalization_69 (BatchNormalization)	(None, 16)	64
dropout_69 (Dropout)	(None, 16)	0
dense_88 (Dense)	(None, 32)	544
batch_normalization_70 (BatchNormalization)	(None, 32)	128
dropout_70 (Dropout)	(None, 32)	0
dense_89 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.3016 - loss: 2.4531 - val_accuracy: 0.4510 - val_loss: 1.7561
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4129 - loss: 1.9578 - val_accuracy: 0.5080 - val_loss: 1.6296
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4339 - loss: 1.8939 - val_accuracy: 0.4950 - val_loss: 1.6867
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4489 - loss: 1.8600 - val_accuracy: 0.5050 - val_loss: 1.6738
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4326 - loss: 1.8810 - val_accuracy: 0.4980 - val_loss: 1.6693
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4374 - loss: 1.8523 - val_accuracy: 0.5090 - val_loss: 1.6183
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4498 - loss: 1.8305 - val_accuracy: 0.5180 - val_loss: 1.6048
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4586 - loss: 1.8163 - val_accuracy: 0.5290 - val_loss: 1.6116
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4581 - loss: 1.8291 - val_accuracy: 0.5110 - val_loss: 1.6391
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4562 - loss: 1.8195 - val_accuracy: 0.5190 - val_loss: 1.5747
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4521 - loss: 1.8299 - val_accuracy: 0.5320 - val_loss: 1.5630
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4665 - loss: 1.7856 - val_accuracy: 0.5390 - val_loss: 1.5802
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4564 - loss: 1.8062 - val_accuracy: 0.5380 - val_loss: 1.5359
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4650 - loss: 1.8030 - val_accuracy: 0.5120 - val_loss: 1.5999
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4652 - loss: 1.8044 - val_accuracy: 0.5220 - val_loss: 1.5927
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4557 - loss: 1.8051 - val_accuracy: 0.5120 - val_loss: 1.5640
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4604 - loss: 1.8004 - val_accuracy: 0.5360 - val_loss: 1.5530
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4721 - loss: 1.7704 - val_accuracy: 0.5530 - val_loss: 1.5421
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4743 - loss: 1.7658 - val_accuracy: 0.5190 - val_loss: 1.5873
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4642 - loss: 1.7862 - val_accuracy: 0.5680 - val_loss: 1.5026
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4617 - loss: 1.8121 - val_accuracy: 0.5320 - val_loss: 1.5468
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4704 - loss: 1.7828 - val_accuracy: 0.5210 - val_loss: 1.5963
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4671 - loss: 1.7790 - val_accuracy: 0.5470 - val_loss: 1.5177
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4824 - loss: 1.7498 - val_accuracy: 0.5290 - val_loss: 1.5597
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4697 - loss: 1.7681 - val_accuracy: 0.5440 - val_loss: 1.5155
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4809 - loss: 1.7471 - val_accuracy: 0.5480 - val_loss: 1.5197
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4663 - loss: 1.7984 - val_accuracy: 0.5290 - val_loss: 1.5564
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4735 - loss: 1.7728 - val_accuracy: 0.5450 - val_loss: 1.5360
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4724 - loss: 1.7507 - val_accuracy: 0.5310 - val_loss: 1.5894
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4764 - loss: 1.7615 - val_accuracy: 0.5520 - val_loss: 1.5227
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4766 - loss: 1.7818 - val_accuracy: 0.5560 - val_loss: 1.4908
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4807 - loss: 1.7683 - val_accuracy: 0.5360 - val_loss: 1.5476
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4866 - loss: 1.7438 - val_accuracy: 0.5170 - val_loss: 1.5479
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4839 - loss: 1.7395 - val_accuracy: 0.5430 - val_loss: 1.5663
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4769 - loss: 1.7680 - val_accuracy: 0.5430 - val_loss: 1.5407
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4785 - loss: 1.7465 - val_accuracy: 0.5330 - val_loss: 1.5527
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4707 - loss: 1.7734 - val_accuracy: 0.5600 - val_loss: 1.4829
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4803 - loss: 1.7418 - val_accuracy: 0.5280 - val_loss: 1.5610
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4786 - loss: 1.7602 - val_accuracy: 0.5470 - val_loss: 1.5185
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4757 - loss: 1.7582 - val_accuracy: 0.5680 - val_loss: 1.4763
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4818 - loss: 1.7374 - val_accuracy: 0.5630 - val_loss: 1.5108
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4800 - loss: 1.7497 - val_accuracy: 0.5390 - val_loss: 1.5169
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4822 - loss: 1.7521 - val_accuracy: 0.5550 - val_loss: 1.4953
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4747 - loss: 1.7566 - val_accuracy: 0.5350 - val_loss: 1.4961
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4666 - loss: 1.7764 - val_accuracy: 0.5520 - val_loss: 1.5006
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4867 - loss: 1.7337 - val_accuracy: 0.5490 - val_loss: 1.5016
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4777 - loss: 1.7585 - val_accuracy: 0.5550 - val_loss: 1.4943
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4860 - loss: 1.7344 - val_accuracy: 0.5270 - val_loss: 1.5738
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4868 - loss: 1.7207 - val_accuracy: 0.5420 - val_loss: 1.4889
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4877 - loss: 1.7174 - val_accuracy: 0.5590 - val_loss: 1.4976
Accuracy: 0.5460
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_19"

Layer (type)	Output Shape	Param #
dense_90 (Dense)	(None, 32)	352
batch_normalization_71 (BatchNormalization)	(None, 32)	128
dropout_71 (Dropout)	(None, 32)	0
dense_91 (Dense)	(None, 64)	2,112
batch_normalization_72 (BatchNormalization)	(None, 64)	256
dropout_72 (Dropout)	(None, 64)	0
dense_92 (Dense)	(None, 32)	2,080
batch_normalization_73 (BatchNormalization)	(None, 32)	128
dropout_73 (Dropout)	(None, 32)	0
dense_93 (Dense)	(None, 16)	528
batch_normalization_74 (BatchNormalization)	(None, 16)	64
dropout_74 (Dropout)	(None, 16)	0
dense_94 (Dense)	(None, 32)	544
batch_normalization_75 (BatchNormalization)	(None, 32)	128
dropout_75 (Dropout)	(None, 32)	0
dense_95 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.2560 - loss: 2.5913 - val_accuracy: 0.4330 - val_loss: 1.8655
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3518 - loss: 2.1158 - val_accuracy: 0.4730 - val_loss: 1.7731
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3752 - loss: 2.0664 - val_accuracy: 0.4670 - val_loss: 1.7502
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3836 - loss: 2.0380 - val_accuracy: 0.5010 - val_loss: 1.7214
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3919 - loss: 2.0086 - val_accuracy: 0.4700 - val_loss: 1.7416
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3911 - loss: 2.0128 - val_accuracy: 0.4640 - val_loss: 1.7582
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3879 - loss: 2.0119 - val_accuracy: 0.4780 - val_loss: 1.7408
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3916 - loss: 1.9995 - val_accuracy: 0.4620 - val_loss: 1.7775
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3945 - loss: 2.0076 - val_accuracy: 0.5150 - val_loss: 1.6631
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4090 - loss: 1.9802 - val_accuracy: 0.4990 - val_loss: 1.6532
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4006 - loss: 1.9789 - val_accuracy: 0.4870 - val_loss: 1.7255
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4068 - loss: 1.9821 - val_accuracy: 0.5020 - val_loss: 1.6575
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4135 - loss: 1.9545 - val_accuracy: 0.5130 - val_loss: 1.6373
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4088 - loss: 1.9620 - val_accuracy: 0.4750 - val_loss: 1.7108
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4059 - loss: 1.9662 - val_accuracy: 0.5200 - val_loss: 1.6346
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4095 - loss: 1.9705 - val_accuracy: 0.4800 - val_loss: 1.6885
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4166 - loss: 1.9415 - val_accuracy: 0.4930 - val_loss: 1.6843
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4199 - loss: 1.9417 - val_accuracy: 0.4890 - val_loss: 1.6584
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4094 - loss: 1.9614 - val_accuracy: 0.5060 - val_loss: 1.6587
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4160 - loss: 1.9546 - val_accuracy: 0.4790 - val_loss: 1.7300
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4191 - loss: 1.9475 - val_accuracy: 0.5040 - val_loss: 1.6992
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4217 - loss: 1.9390 - val_accuracy: 0.5280 - val_loss: 1.6152
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4279 - loss: 1.9234 - val_accuracy: 0.5150 - val_loss: 1.6383
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4186 - loss: 1.9374 - val_accuracy: 0.4890 - val_loss: 1.7182
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4171 - loss: 1.9360 - val_accuracy: 0.5170 - val_loss: 1.6645
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4254 - loss: 1.9215 - val_accuracy: 0.5020 - val_loss: 1.6632
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4213 - loss: 1.9408 - val_accuracy: 0.4990 - val_loss: 1.6759
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4203 - loss: 1.9296 - val_accuracy: 0.5140 - val_loss: 1.6472
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4331 - loss: 1.9131 - val_accuracy: 0.5140 - val_loss: 1.6434
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4304 - loss: 1.9258 - val_accuracy: 0.4520 - val_loss: 1.7705
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4284 - loss: 1.9310 - val_accuracy: 0.4910 - val_loss: 1.6432
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4258 - loss: 1.9144 - val_accuracy: 0.4910 - val_loss: 1.6727
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4216 - loss: 1.9456 - val_accuracy: 0.5140 - val_loss: 1.6891
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4295 - loss: 1.9359 - val_accuracy: 0.5130 - val_loss: 1.6319
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4197 - loss: 1.9311 - val_accuracy: 0.4960 - val_loss: 1.6567
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4288 - loss: 1.9163 - val_accuracy: 0.5000 - val_loss: 1.6191
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4224 - loss: 1.9211 - val_accuracy: 0.4990 - val_loss: 1.6086
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4302 - loss: 1.9178 - val_accuracy: 0.5220 - val_loss: 1.6115
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4274 - loss: 1.9233 - val_accuracy: 0.5140 - val_loss: 1.5991
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4268 - loss: 1.9268 - val_accuracy: 0.4940 - val_loss: 1.6281
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4234 - loss: 1.9285 - val_accuracy: 0.5320 - val_loss: 1.6254
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4194 - loss: 1.9563 - val_accuracy: 0.5120 - val_loss: 1.6479
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4318 - loss: 1.9367 - val_accuracy: 0.5210 - val_loss: 1.6301
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4342 - loss: 1.9086 - val_accuracy: 0.5190 - val_loss: 1.6340
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4319 - loss: 1.9160 - val_accuracy: 0.5120 - val_loss: 1.6186
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4274 - loss: 1.9246 - val_accuracy: 0.5230 - val_loss: 1.6004
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.4256 - loss: 1.9227 - val_accuracy: 0.5340 - val_loss: 1.6046
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4253 - loss: 1.9236 - val_accuracy: 0.5140 - val_loss: 1.6274
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4276 - loss: 1.9289 - val_accuracy: 0.5100 - val_loss: 1.6090
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4320 - loss: 1.9063 - val_accuracy: 0.5060 - val_loss: 1.6379
Accuracy: 0.4935
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_20"

Layer (type)	Output Shape	Param #
dense_96 (Dense)	(None, 32)	352
batch_normalization_76 (BatchNormalization)	(None, 32)	128
dropout_76 (Dropout)	(None, 32)	0
dense_97 (Dense)	(None, 64)	2,112
batch_normalization_77 (BatchNormalization)	(None, 64)	256
dropout_77 (Dropout)	(None, 64)	0
dense_98 (Dense)	(None, 32)	2,080
batch_normalization_78 (BatchNormalization)	(None, 32)	128
dropout_78 (Dropout)	(None, 32)	0
dense_99 (Dense)	(None, 16)	528
batch_normalization_79 (BatchNormalization)	(None, 16)	64
dropout_79 (Dropout)	(None, 16)	0
dense_100 (Dense)	(None, 32)	544
batch_normalization_80 (BatchNormalization)	(None, 32)	128
dropout_80 (Dropout)	(None, 32)	0
dense_101 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.2062 - loss: 2.7636 - val_accuracy: 0.3640 - val_loss: 1.9408
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2874 - loss: 2.2622 - val_accuracy: 0.3870 - val_loss: 1.9629
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3001 - loss: 2.2265 - val_accuracy: 0.4020 - val_loss: 1.8722
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3103 - loss: 2.2073 - val_accuracy: 0.3940 - val_loss: 1.8504
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3130 - loss: 2.2117 - val_accuracy: 0.4280 - val_loss: 1.8256
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.3237 - loss: 2.1738 - val_accuracy: 0.4380 - val_loss: 1.8687
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3198 - loss: 2.1756 - val_accuracy: 0.4260 - val_loss: 1.8493
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3215 - loss: 2.1804 - val_accuracy: 0.4070 - val_loss: 1.8723
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3295 - loss: 2.1706 - val_accuracy: 0.4440 - val_loss: 1.8105
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3382 - loss: 2.1465 - val_accuracy: 0.4480 - val_loss: 1.7869
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3312 - loss: 2.1652 - val_accuracy: 0.4690 - val_loss: 1.7585
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3595 - loss: 2.1071 - val_accuracy: 0.4260 - val_loss: 1.8432
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3449 - loss: 2.1481 - val_accuracy: 0.4810 - val_loss: 1.7903
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3449 - loss: 2.1337 - val_accuracy: 0.4440 - val_loss: 1.8028
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3428 - loss: 2.1355 - val_accuracy: 0.4660 - val_loss: 1.7803
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3445 - loss: 2.1272 - val_accuracy: 0.4490 - val_loss: 1.7778
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3457 - loss: 2.1370 - val_accuracy: 0.4630 - val_loss: 1.7768
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3513 - loss: 2.1203 - val_accuracy: 0.4410 - val_loss: 1.8624
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3514 - loss: 2.1271 - val_accuracy: 0.4360 - val_loss: 1.7834
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3480 - loss: 2.1380 - val_accuracy: 0.4590 - val_loss: 1.7599
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3446 - loss: 2.1256 - val_accuracy: 0.4370 - val_loss: 1.7977
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3579 - loss: 2.1003 - val_accuracy: 0.4520 - val_loss: 1.7803
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3592 - loss: 2.1146 - val_accuracy: 0.4770 - val_loss: 1.7356
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3570 - loss: 2.1059 - val_accuracy: 0.4820 - val_loss: 1.7442
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3636 - loss: 2.1112 - val_accuracy: 0.4610 - val_loss: 1.7814
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3559 - loss: 2.1171 - val_accuracy: 0.4660 - val_loss: 1.7557
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3478 - loss: 2.1263 - val_accuracy: 0.4550 - val_loss: 1.7504
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3608 - loss: 2.1071 - val_accuracy: 0.4520 - val_loss: 1.7841
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3485 - loss: 2.1080 - val_accuracy: 0.4830 - val_loss: 1.7268
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.3493 - loss: 2.1112 - val_accuracy: 0.4420 - val_loss: 1.7674
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3656 - loss: 2.1046 - val_accuracy: 0.4710 - val_loss: 1.7636
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3606 - loss: 2.0962 - val_accuracy: 0.4680 - val_loss: 1.7659
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3605 - loss: 2.1120 - val_accuracy: 0.4820 - val_loss: 1.7253
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3604 - loss: 2.1067 - val_accuracy: 0.4900 - val_loss: 1.7554
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3588 - loss: 2.0893 - val_accuracy: 0.4870 - val_loss: 1.7450
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3563 - loss: 2.1023 - val_accuracy: 0.4670 - val_loss: 1.7990
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3639 - loss: 2.0981 - val_accuracy: 0.4470 - val_loss: 1.7917
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3616 - loss: 2.1021 - val_accuracy: 0.4690 - val_loss: 1.7881
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3652 - loss: 2.1046 - val_accuracy: 0.4810 - val_loss: 1.7384
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3634 - loss: 2.0833 - val_accuracy: 0.4340 - val_loss: 1.8657
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3579 - loss: 2.1085 - val_accuracy: 0.4760 - val_loss: 1.7468
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3647 - loss: 2.1023 - val_accuracy: 0.4390 - val_loss: 1.7921
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3632 - loss: 2.0933 - val_accuracy: 0.4920 - val_loss: 1.7339
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3649 - loss: 2.1019 - val_accuracy: 0.4580 - val_loss: 1.7405
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.3657 - loss: 2.0897 - val_accuracy: 0.4670 - val_loss: 1.7347
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3636 - loss: 2.0902 - val_accuracy: 0.4620 - val_loss: 1.7698
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3605 - loss: 2.0896 - val_accuracy: 0.4790 - val_loss: 1.7375
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3707 - loss: 2.1024 - val_accuracy: 0.4830 - val_loss: 1.7532
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3669 - loss: 2.0913 - val_accuracy: 0.4960 - val_loss: 1.7371
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3583 - loss: 2.1213 - val_accuracy: 0.4830 - val_loss: 1.7561
Accuracy: 0.5035
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_21"

Layer (type)	Output Shape	Param #
dense_102 (Dense)	(None, 32)	352
batch_normalization_81 (BatchNormalization)	(None, 32)	128
dropout_81 (Dropout)	(None, 32)	0
dense_103 (Dense)	(None, 64)	2,112
batch_normalization_82 (BatchNormalization)	(None, 64)	256
dropout_82 (Dropout)	(None, 64)	0
dense_104 (Dense)	(None, 32)	2,080
batch_normalization_83 (BatchNormalization)	(None, 32)	128
dropout_83 (Dropout)	(None, 32)	0
dense_105 (Dense)	(None, 16)	528
batch_normalization_84 (BatchNormalization)	(None, 16)	64
dropout_84 (Dropout)	(None, 16)	0
dense_106 (Dense)	(None, 32)	544
batch_normalization_85 (BatchNormalization)	(None, 32)	128
dropout_85 (Dropout)	(None, 32)	0
dense_107 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.3094 - loss: 2.4743 - val_accuracy: 0.5000 - val_loss: 1.7378
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4396 - loss: 1.8542 - val_accuracy: 0.5090 - val_loss: 1.6173
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4577 - loss: 1.7918 - val_accuracy: 0.4980 - val_loss: 1.6341
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4643 - loss: 1.7528 - val_accuracy: 0.5290 - val_loss: 1.5102
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4749 - loss: 1.7245 - val_accuracy: 0.5100 - val_loss: 1.5633
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4735 - loss: 1.7341 - val_accuracy: 0.5320 - val_loss: 1.4940
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4810 - loss: 1.7010 - val_accuracy: 0.5360 - val_loss: 1.5221
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4818 - loss: 1.7065 - val_accuracy: 0.5270 - val_loss: 1.5406
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4841 - loss: 1.6945 - val_accuracy: 0.5640 - val_loss: 1.4697
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4874 - loss: 1.6935 - val_accuracy: 0.5380 - val_loss: 1.4830
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4996 - loss: 1.6612 - val_accuracy: 0.5490 - val_loss: 1.4892
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5002 - loss: 1.6584 - val_accuracy: 0.5140 - val_loss: 1.5760
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4869 - loss: 1.6813 - val_accuracy: 0.5560 - val_loss: 1.4445
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4954 - loss: 1.6587 - val_accuracy: 0.5520 - val_loss: 1.4527
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5035 - loss: 1.6412 - val_accuracy: 0.5430 - val_loss: 1.5250
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4967 - loss: 1.6532 - val_accuracy: 0.5550 - val_loss: 1.4629
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4955 - loss: 1.6573 - val_accuracy: 0.5530 - val_loss: 1.4624
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5143 - loss: 1.6276 - val_accuracy: 0.5410 - val_loss: 1.4539
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5008 - loss: 1.6429 - val_accuracy: 0.5500 - val_loss: 1.4509
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5102 - loss: 1.6150 - val_accuracy: 0.5410 - val_loss: 1.5098
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5019 - loss: 1.6522 - val_accuracy: 0.5550 - val_loss: 1.5013
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5059 - loss: 1.6241 - val_accuracy: 0.5400 - val_loss: 1.4955
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5052 - loss: 1.6379 - val_accuracy: 0.5560 - val_loss: 1.4697
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5123 - loss: 1.6220 - val_accuracy: 0.5690 - val_loss: 1.4291
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5035 - loss: 1.6254 - val_accuracy: 0.5770 - val_loss: 1.4204
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5039 - loss: 1.6255 - val_accuracy: 0.5420 - val_loss: 1.4771
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5094 - loss: 1.6374 - val_accuracy: 0.5610 - val_loss: 1.4178
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5082 - loss: 1.6341 - val_accuracy: 0.5460 - val_loss: 1.4675
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5008 - loss: 1.6406 - val_accuracy: 0.5700 - val_loss: 1.4429
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5141 - loss: 1.6196 - val_accuracy: 0.5580 - val_loss: 1.4600
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5074 - loss: 1.6266 - val_accuracy: 0.5640 - val_loss: 1.4354
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5049 - loss: 1.6412 - val_accuracy: 0.5720 - val_loss: 1.4253
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5218 - loss: 1.6034 - val_accuracy: 0.5580 - val_loss: 1.4428
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5162 - loss: 1.6180 - val_accuracy: 0.5750 - val_loss: 1.3943
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5145 - loss: 1.6018 - val_accuracy: 0.5500 - val_loss: 1.4594
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5137 - loss: 1.6102 - val_accuracy: 0.5920 - val_loss: 1.4110
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5201 - loss: 1.5989 - val_accuracy: 0.5660 - val_loss: 1.3982
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5185 - loss: 1.6110 - val_accuracy: 0.5680 - val_loss: 1.4342
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5234 - loss: 1.5905 - val_accuracy: 0.5930 - val_loss: 1.3864
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5101 - loss: 1.6073 - val_accuracy: 0.5800 - val_loss: 1.3889
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5177 - loss: 1.5915 - val_accuracy: 0.5860 - val_loss: 1.4431
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5104 - loss: 1.6163 - val_accuracy: 0.5470 - val_loss: 1.5467
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5083 - loss: 1.6090 - val_accuracy: 0.5660 - val_loss: 1.4332
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5216 - loss: 1.6128 - val_accuracy: 0.5720 - val_loss: 1.4201
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5152 - loss: 1.6189 - val_accuracy: 0.5560 - val_loss: 1.4385
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5265 - loss: 1.5868 - val_accuracy: 0.5920 - val_loss: 1.3979
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5204 - loss: 1.5963 - val_accuracy: 0.5670 - val_loss: 1.4480
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5170 - loss: 1.6157 - val_accuracy: 0.5690 - val_loss: 1.4257
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5162 - loss: 1.6036 - val_accuracy: 0.5520 - val_loss: 1.4598
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5158 - loss: 1.6065 - val_accuracy: 0.5830 - val_loss: 1.3977
Accuracy: 0.5790
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_22"

Layer (type)	Output Shape	Param #
dense_108 (Dense)	(None, 32)	352
batch_normalization_86 (BatchNormalization)	(None, 32)	128
dropout_86 (Dropout)	(None, 32)	0
dense_109 (Dense)	(None, 64)	2,112
batch_normalization_87 (BatchNormalization)	(None, 64)	256
dropout_87 (Dropout)	(None, 64)	0
dense_110 (Dense)	(None, 32)	2,080
batch_normalization_88 (BatchNormalization)	(None, 32)	128
dropout_88 (Dropout)	(None, 32)	0
dense_111 (Dense)	(None, 16)	528
batch_normalization_89 (BatchNormalization)	(None, 16)	64
dropout_89 (Dropout)	(None, 16)	0
dense_112 (Dense)	(None, 32)	544
batch_normalization_90 (BatchNormalization)	(None, 32)	128
dropout_90 (Dropout)	(None, 32)	0
dense_113 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.2456 - loss: 2.6776 - val_accuracy: 0.4410 - val_loss: 1.8231
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3797 - loss: 2.0396 - val_accuracy: 0.4590 - val_loss: 1.7551
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3862 - loss: 1.9852 - val_accuracy: 0.4970 - val_loss: 1.6644
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4097 - loss: 1.9332 - val_accuracy: 0.4710 - val_loss: 1.7036
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4095 - loss: 1.9258 - val_accuracy: 0.4910 - val_loss: 1.6522
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4185 - loss: 1.9071 - val_accuracy: 0.5010 - val_loss: 1.6370
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4114 - loss: 1.9051 - val_accuracy: 0.4850 - val_loss: 1.6461
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4164 - loss: 1.8934 - val_accuracy: 0.5010 - val_loss: 1.6349
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4273 - loss: 1.8991 - val_accuracy: 0.4960 - val_loss: 1.6248
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4270 - loss: 1.8827 - val_accuracy: 0.5010 - val_loss: 1.6717
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4296 - loss: 1.8664 - val_accuracy: 0.5070 - val_loss: 1.5899
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4306 - loss: 1.8665 - val_accuracy: 0.4910 - val_loss: 1.6642
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4307 - loss: 1.8742 - val_accuracy: 0.5030 - val_loss: 1.5830
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4303 - loss: 1.8775 - val_accuracy: 0.5070 - val_loss: 1.6152
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4255 - loss: 1.8762 - val_accuracy: 0.5240 - val_loss: 1.5761
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4381 - loss: 1.8669 - val_accuracy: 0.4890 - val_loss: 1.6751
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4435 - loss: 1.8472 - val_accuracy: 0.5390 - val_loss: 1.5551
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4390 - loss: 1.8470 - val_accuracy: 0.5120 - val_loss: 1.5998
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4431 - loss: 1.8456 - val_accuracy: 0.5110 - val_loss: 1.6193
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4373 - loss: 1.8623 - val_accuracy: 0.5210 - val_loss: 1.5818
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4371 - loss: 1.8422 - val_accuracy: 0.4910 - val_loss: 1.6064
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4503 - loss: 1.8286 - val_accuracy: 0.5320 - val_loss: 1.5426
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4455 - loss: 1.8385 - val_accuracy: 0.5340 - val_loss: 1.5577
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4490 - loss: 1.8267 - val_accuracy: 0.5120 - val_loss: 1.5889
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4551 - loss: 1.8282 - val_accuracy: 0.5420 - val_loss: 1.5386
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4466 - loss: 1.8230 - val_accuracy: 0.5010 - val_loss: 1.6170
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4439 - loss: 1.8393 - val_accuracy: 0.5340 - val_loss: 1.5345
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4491 - loss: 1.8280 - val_accuracy: 0.5340 - val_loss: 1.5748
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4526 - loss: 1.8178 - val_accuracy: 0.5270 - val_loss: 1.5717
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4563 - loss: 1.8218 - val_accuracy: 0.5370 - val_loss: 1.5444
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4532 - loss: 1.8164 - val_accuracy: 0.5100 - val_loss: 1.6085
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4514 - loss: 1.8225 - val_accuracy: 0.5310 - val_loss: 1.5720
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4589 - loss: 1.7861 - val_accuracy: 0.5430 - val_loss: 1.5060
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4590 - loss: 1.7949 - val_accuracy: 0.5250 - val_loss: 1.5369
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4539 - loss: 1.8173 - val_accuracy: 0.5220 - val_loss: 1.5607
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4434 - loss: 1.8415 - val_accuracy: 0.5410 - val_loss: 1.5387
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4459 - loss: 1.8406 - val_accuracy: 0.5070 - val_loss: 1.6151
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4393 - loss: 1.8474 - val_accuracy: 0.5170 - val_loss: 1.5787
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4445 - loss: 1.8490 - val_accuracy: 0.5100 - val_loss: 1.6026
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4477 - loss: 1.8441 - val_accuracy: 0.5320 - val_loss: 1.5620
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4529 - loss: 1.8315 - val_accuracy: 0.5110 - val_loss: 1.5788
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4598 - loss: 1.8114 - val_accuracy: 0.5460 - val_loss: 1.5305
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4548 - loss: 1.8137 - val_accuracy: 0.5150 - val_loss: 1.5552
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4548 - loss: 1.8103 - val_accuracy: 0.5330 - val_loss: 1.5462
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4543 - loss: 1.8270 - val_accuracy: 0.5390 - val_loss: 1.5281
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4536 - loss: 1.8038 - val_accuracy: 0.5320 - val_loss: 1.5471
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4604 - loss: 1.8142 - val_accuracy: 0.5340 - val_loss: 1.5241
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4544 - loss: 1.8007 - val_accuracy: 0.5420 - val_loss: 1.5106
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4496 - loss: 1.8077 - val_accuracy: 0.5120 - val_loss: 1.6086
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4464 - loss: 1.8107 - val_accuracy: 0.5510 - val_loss: 1.5231
Accuracy: 0.5450
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_23"

Layer (type)	Output Shape	Param #
dense_114 (Dense)	(None, 32)	352
batch_normalization_91 (BatchNormalization)	(None, 32)	128
dropout_91 (Dropout)	(None, 32)	0
dense_115 (Dense)	(None, 64)	2,112
batch_normalization_92 (BatchNormalization)	(None, 64)	256
dropout_92 (Dropout)	(None, 64)	0
dense_116 (Dense)	(None, 32)	2,080
batch_normalization_93 (BatchNormalization)	(None, 32)	128
dropout_93 (Dropout)	(None, 32)	0
dense_117 (Dense)	(None, 16)	528
batch_normalization_94 (BatchNormalization)	(None, 16)	64
dropout_94 (Dropout)	(None, 16)	0
dense_118 (Dense)	(None, 32)	544
batch_normalization_95 (BatchNormalization)	(None, 32)	128
dropout_95 (Dropout)	(None, 32)	0
dense_119 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.1923 - loss: 2.8364 - val_accuracy: 0.3300 - val_loss: 2.0708
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3158 - loss: 2.1831 - val_accuracy: 0.4280 - val_loss: 1.8199
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3247 - loss: 2.1383 - val_accuracy: 0.4090 - val_loss: 1.8908
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3481 - loss: 2.0785 - val_accuracy: 0.4230 - val_loss: 1.8233
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3412 - loss: 2.1026 - val_accuracy: 0.4440 - val_loss: 1.7394
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3468 - loss: 2.0898 - val_accuracy: 0.4490 - val_loss: 1.7702
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3534 - loss: 2.0660 - val_accuracy: 0.4340 - val_loss: 1.7457
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3596 - loss: 2.0447 - val_accuracy: 0.4070 - val_loss: 1.8521
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3660 - loss: 2.0439 - val_accuracy: 0.4280 - val_loss: 1.7859
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3607 - loss: 2.0435 - val_accuracy: 0.4970 - val_loss: 1.7171
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3711 - loss: 2.0511 - val_accuracy: 0.4680 - val_loss: 1.7345
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3721 - loss: 2.0146 - val_accuracy: 0.4890 - val_loss: 1.7290
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3719 - loss: 2.0227 - val_accuracy: 0.4680 - val_loss: 1.7273
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3704 - loss: 2.0365 - val_accuracy: 0.4610 - val_loss: 1.7155
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3737 - loss: 2.0228 - val_accuracy: 0.4450 - val_loss: 1.6990
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3743 - loss: 2.0182 - val_accuracy: 0.4710 - val_loss: 1.7001
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3763 - loss: 2.0064 - val_accuracy: 0.4880 - val_loss: 1.6834
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3911 - loss: 1.9914 - val_accuracy: 0.4760 - val_loss: 1.7011
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3871 - loss: 2.0133 - val_accuracy: 0.4760 - val_loss: 1.7117
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3813 - loss: 2.0089 - val_accuracy: 0.5100 - val_loss: 1.6713
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3811 - loss: 2.0156 - val_accuracy: 0.4990 - val_loss: 1.6443
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3889 - loss: 1.9863 - val_accuracy: 0.4930 - val_loss: 1.6732
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3861 - loss: 2.0129 - val_accuracy: 0.4900 - val_loss: 1.6847
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3870 - loss: 2.0152 - val_accuracy: 0.5020 - val_loss: 1.6579
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3859 - loss: 1.9961 - val_accuracy: 0.5000 - val_loss: 1.6603
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3760 - loss: 2.0070 - val_accuracy: 0.4950 - val_loss: 1.6331
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3929 - loss: 1.9767 - val_accuracy: 0.4940 - val_loss: 1.6625
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3936 - loss: 2.0074 - val_accuracy: 0.5030 - val_loss: 1.6837
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3938 - loss: 2.0106 - val_accuracy: 0.4720 - val_loss: 1.7028
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3880 - loss: 2.0039 - val_accuracy: 0.4830 - val_loss: 1.6686
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3913 - loss: 1.9759 - val_accuracy: 0.4930 - val_loss: 1.6712
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3878 - loss: 1.9857 - val_accuracy: 0.4860 - val_loss: 1.6777
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3942 - loss: 1.9759 - val_accuracy: 0.5020 - val_loss: 1.6348
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3893 - loss: 1.9848 - val_accuracy: 0.4770 - val_loss: 1.6690
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3896 - loss: 1.9950 - val_accuracy: 0.4510 - val_loss: 1.7804
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3905 - loss: 1.9812 - val_accuracy: 0.4960 - val_loss: 1.6694
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3986 - loss: 1.9663 - val_accuracy: 0.4900 - val_loss: 1.7026
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3887 - loss: 1.9915 - val_accuracy: 0.5100 - val_loss: 1.6312
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3891 - loss: 1.9826 - val_accuracy: 0.4920 - val_loss: 1.6858
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3935 - loss: 1.9930 - val_accuracy: 0.4930 - val_loss: 1.6730
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3932 - loss: 1.9857 - val_accuracy: 0.5310 - val_loss: 1.6105
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3986 - loss: 1.9836 - val_accuracy: 0.4940 - val_loss: 1.6595
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3927 - loss: 1.9852 - val_accuracy: 0.5100 - val_loss: 1.6198
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4012 - loss: 1.9783 - val_accuracy: 0.4930 - val_loss: 1.6393
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3957 - loss: 1.9816 - val_accuracy: 0.5190 - val_loss: 1.6342
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3955 - loss: 1.9681 - val_accuracy: 0.4720 - val_loss: 1.7627
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3932 - loss: 1.9797 - val_accuracy: 0.5090 - val_loss: 1.6208
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4008 - loss: 1.9839 - val_accuracy: 0.4810 - val_loss: 1.6820
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4054 - loss: 1.9799 - val_accuracy: 0.4930 - val_loss: 1.6646
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3898 - loss: 1.9920 - val_accuracy: 0.4750 - val_loss: 1.6658
Accuracy: 0.4765
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_24"

Layer (type)	Output Shape	Param #
dense_120 (Dense)	(None, 32)	352
batch_normalization_96 (BatchNormalization)	(None, 32)	128
dropout_96 (Dropout)	(None, 32)	0
dense_121 (Dense)	(None, 64)	2,112
batch_normalization_97 (BatchNormalization)	(None, 64)	256
dropout_97 (Dropout)	(None, 64)	0
dense_122 (Dense)	(None, 32)	2,080
batch_normalization_98 (BatchNormalization)	(None, 32)	128
dropout_98 (Dropout)	(None, 32)	0
dense_123 (Dense)	(None, 16)	528
batch_normalization_99 (BatchNormalization)	(None, 16)	64
dropout_99 (Dropout)	(None, 16)	0
dense_124 (Dense)	(None, 32)	544
batch_normalization_100 (BatchNormalization)	(None, 32)	128
dropout_100 (Dropout)	(None, 32)	0
dense_125 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.2030 - loss: 2.9183 - val_accuracy: 0.2700 - val_loss: 2.6851
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2518 - loss: 2.6439 - val_accuracy: 0.2480 - val_loss: 3.0796
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2563 - loss: 2.6213 - val_accuracy: 0.3070 - val_loss: 2.4564
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2611 - loss: 2.6251 - val_accuracy: 0.3110 - val_loss: 2.5836
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2590 - loss: 2.6363 - val_accuracy: 0.3100 - val_loss: 2.3537
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2703 - loss: 2.5881 - val_accuracy: 0.3030 - val_loss: 2.4828
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2747 - loss: 2.5845 - val_accuracy: 0.2790 - val_loss: 2.6345
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2691 - loss: 2.5899 - val_accuracy: 0.3180 - val_loss: 2.3821
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2802 - loss: 2.5647 - val_accuracy: 0.2030 - val_loss: 2.7130
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2697 - loss: 2.6048 - val_accuracy: 0.3460 - val_loss: 2.3695
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2744 - loss: 2.5940 - val_accuracy: 0.3850 - val_loss: 2.2354
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2838 - loss: 2.5760 - val_accuracy: 0.3110 - val_loss: 2.4397
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2848 - loss: 2.5543 - val_accuracy: 0.3610 - val_loss: 2.3982
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2850 - loss: 2.5456 - val_accuracy: 0.3690 - val_loss: 2.3089
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2827 - loss: 2.5529 - val_accuracy: 0.3610 - val_loss: 2.2491
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2818 - loss: 2.5812 - val_accuracy: 0.3510 - val_loss: 2.3922
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2879 - loss: 2.5897 - val_accuracy: 0.2990 - val_loss: 2.4215
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2916 - loss: 2.5580 - val_accuracy: 0.3440 - val_loss: 2.3545
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2962 - loss: 2.5586 - val_accuracy: 0.3390 - val_loss: 2.3270
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2891 - loss: 2.5636 - val_accuracy: 0.3410 - val_loss: 2.4011
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2879 - loss: 2.6038 - val_accuracy: 0.3140 - val_loss: 2.4954
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2889 - loss: 2.5681 - val_accuracy: 0.3540 - val_loss: 2.3866
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2896 - loss: 2.5476 - val_accuracy: 0.2460 - val_loss: 2.7195
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3045 - loss: 2.5399 - val_accuracy: 0.3410 - val_loss: 2.3308
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3126 - loss: 2.5219 - val_accuracy: 0.3160 - val_loss: 2.3920
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2962 - loss: 2.5535 - val_accuracy: 0.3420 - val_loss: 2.4223
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3011 - loss: 2.5456 - val_accuracy: 0.3450 - val_loss: 2.4846
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3001 - loss: 2.5570 - val_accuracy: 0.3510 - val_loss: 2.3298
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3042 - loss: 2.5501 - val_accuracy: 0.3810 - val_loss: 2.2907
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3026 - loss: 2.5251 - val_accuracy: 0.3230 - val_loss: 2.4808
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.3096 - loss: 2.5277 - val_accuracy: 0.3280 - val_loss: 2.4834
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2991 - loss: 2.5820 - val_accuracy: 0.3790 - val_loss: 2.2836
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3160 - loss: 2.5080 - val_accuracy: 0.3760 - val_loss: 2.2557
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3088 - loss: 2.5429 - val_accuracy: 0.3560 - val_loss: 2.2900
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.3083 - loss: 2.5128 - val_accuracy: 0.3200 - val_loss: 2.4514
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3047 - loss: 2.5322 - val_accuracy: 0.3600 - val_loss: 2.3341
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3087 - loss: 2.5249 - val_accuracy: 0.3220 - val_loss: 2.3771
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3029 - loss: 2.5644 - val_accuracy: 0.4150 - val_loss: 2.1930
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3088 - loss: 2.5343 - val_accuracy: 0.3740 - val_loss: 2.3091
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.3126 - loss: 2.5013 - val_accuracy: 0.3430 - val_loss: 2.3204
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.3020 - loss: 2.5624 - val_accuracy: 0.3600 - val_loss: 2.4088
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2992 - loss: 2.5575 - val_accuracy: 0.3480 - val_loss: 2.3813
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3112 - loss: 2.5178 - val_accuracy: 0.3400 - val_loss: 2.2587
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3037 - loss: 2.4964 - val_accuracy: 0.2030 - val_loss: 4.9167
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2536 - loss: 3.0536 - val_accuracy: 0.3480 - val_loss: 2.3119
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2799 - loss: 2.5655 - val_accuracy: 0.3360 - val_loss: 2.3020
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2914 - loss: 2.5600 - val_accuracy: 0.3120 - val_loss: 2.3728
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2915 - loss: 2.5668 - val_accuracy: 0.3480 - val_loss: 2.3326
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2882 - loss: 2.5421 - val_accuracy: 0.3400 - val_loss: 2.3009
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3034 - loss: 2.5152 - val_accuracy: 0.3590 - val_loss: 2.3918
Accuracy: 0.3585
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_25"

Layer (type)	Output Shape	Param #
dense_126 (Dense)	(None, 32)	352
batch_normalization_101 (BatchNormalization)	(None, 32)	128
dropout_101 (Dropout)	(None, 32)	0
dense_127 (Dense)	(None, 64)	2,112
batch_normalization_102 (BatchNormalization)	(None, 64)	256
dropout_102 (Dropout)	(None, 64)	0
dense_128 (Dense)	(None, 32)	2,080
batch_normalization_103 (BatchNormalization)	(None, 32)	128
dropout_103 (Dropout)	(None, 32)	0
dense_129 (Dense)	(None, 16)	528
batch_normalization_104 (BatchNormalization)	(None, 16)	64
dropout_104 (Dropout)	(None, 16)	0
dense_130 (Dense)	(None, 32)	544
batch_normalization_105 (BatchNormalization)	(None, 32)	128
dropout_105 (Dropout)	(None, 32)	0
dense_131 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.1570 - loss: 3.0537 - val_accuracy: 0.2330 - val_loss: 3.0713
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1833 - loss: 2.8590 - val_accuracy: 0.2180 - val_loss: 2.6432
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.1960 - loss: 2.8106 - val_accuracy: 0.2550 - val_loss: 2.5600
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2064 - loss: 2.7980 - val_accuracy: 0.2380 - val_loss: 2.6965
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2119 - loss: 2.7844 - val_accuracy: 0.2610 - val_loss: 2.6448
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2156 - loss: 2.7946 - val_accuracy: 0.1800 - val_loss: 2.8366
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2113 - loss: 2.7849 - val_accuracy: 0.2600 - val_loss: 2.5250
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2223 - loss: 2.7857 - val_accuracy: 0.2760 - val_loss: 2.6236
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2143 - loss: 2.7751 - val_accuracy: 0.2500 - val_loss: 2.5852
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2153 - loss: 2.7846 - val_accuracy: 0.2410 - val_loss: 2.7535
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2164 - loss: 2.8093 - val_accuracy: 0.2800 - val_loss: 2.5779
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2240 - loss: 2.7748 - val_accuracy: 0.2450 - val_loss: 2.5965
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2249 - loss: 2.7762 - val_accuracy: 0.1660 - val_loss: 3.1267
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2168 - loss: 2.7983 - val_accuracy: 0.2950 - val_loss: 2.4817
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2282 - loss: 2.7737 - val_accuracy: 0.3080 - val_loss: 2.4264
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2229 - loss: 2.7824 - val_accuracy: 0.3020 - val_loss: 2.5755
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2354 - loss: 2.7638 - val_accuracy: 0.2910 - val_loss: 2.5163
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2297 - loss: 2.7655 - val_accuracy: 0.2980 - val_loss: 2.5310
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2275 - loss: 2.7679 - val_accuracy: 0.2760 - val_loss: 2.5928
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2439 - loss: 2.8091 - val_accuracy: 0.2680 - val_loss: 2.6327
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2259 - loss: 2.7721 - val_accuracy: 0.3300 - val_loss: 2.5752
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2286 - loss: 2.7638 - val_accuracy: 0.2610 - val_loss: 2.6692
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2370 - loss: 2.7441 - val_accuracy: 0.3000 - val_loss: 2.4661
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2358 - loss: 2.7630 - val_accuracy: 0.2460 - val_loss: 2.7314
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2322 - loss: 2.7954 - val_accuracy: 0.2090 - val_loss: 3.1587
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2424 - loss: 2.7704 - val_accuracy: 0.3180 - val_loss: 2.4919
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2466 - loss: 2.7352 - val_accuracy: 0.1870 - val_loss: 3.0536
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2445 - loss: 2.7693 - val_accuracy: 0.3300 - val_loss: 2.4954
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2321 - loss: 2.7620 - val_accuracy: 0.2870 - val_loss: 2.6344
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2507 - loss: 2.7263 - val_accuracy: 0.3070 - val_loss: 2.4443
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2378 - loss: 2.7606 - val_accuracy: 0.2800 - val_loss: 2.5969
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2285 - loss: 2.7744 - val_accuracy: 0.2420 - val_loss: 2.6201
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2321 - loss: 2.7621 - val_accuracy: 0.3220 - val_loss: 2.5079
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2496 - loss: 2.7433 - val_accuracy: 0.2390 - val_loss: 2.6413
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2479 - loss: 2.7607 - val_accuracy: 0.2850 - val_loss: 2.5022
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2462 - loss: 2.7460 - val_accuracy: 0.2930 - val_loss: 2.5975
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2456 - loss: 2.7661 - val_accuracy: 0.2880 - val_loss: 2.6529
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.2466 - loss: 2.7541 - val_accuracy: 0.3560 - val_loss: 2.3727
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2444 - loss: 2.7307 - val_accuracy: 0.3200 - val_loss: 2.3895
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2383 - loss: 2.7502 - val_accuracy: 0.3110 - val_loss: 2.3777
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2380 - loss: 2.7564 - val_accuracy: 0.3020 - val_loss: 2.5805
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2451 - loss: 2.7660 - val_accuracy: 0.2990 - val_loss: 2.6778
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2429 - loss: 2.7815 - val_accuracy: 0.2760 - val_loss: 2.5541
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2512 - loss: 2.7466 - val_accuracy: 0.3050 - val_loss: 2.7600
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2461 - loss: 2.7462 - val_accuracy: 0.3520 - val_loss: 2.5498
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2426 - loss: 2.7764 - val_accuracy: 0.3200 - val_loss: 2.4679
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.2576 - loss: 2.7168 - val_accuracy: 0.2690 - val_loss: 2.5873
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.2511 - loss: 2.7409 - val_accuracy: 0.2170 - val_loss: 2.6775
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2482 - loss: 2.7366 - val_accuracy: 0.3000 - val_loss: 2.4586
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2504 - loss: 2.7608 - val_accuracy: 0.2800 - val_loss: 2.4422
Accuracy: 0.2930
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_26"

Layer (type)	Output Shape	Param #
dense_132 (Dense)	(None, 32)	352
batch_normalization_106 (BatchNormalization)	(None, 32)	128
dropout_106 (Dropout)	(None, 32)	0
dense_133 (Dense)	(None, 64)	2,112
batch_normalization_107 (BatchNormalization)	(None, 64)	256
dropout_107 (Dropout)	(None, 64)	0
dense_134 (Dense)	(None, 32)	2,080
batch_normalization_108 (BatchNormalization)	(None, 32)	128
dropout_108 (Dropout)	(None, 32)	0
dense_135 (Dense)	(None, 16)	528
batch_normalization_109 (BatchNormalization)	(None, 16)	64
dropout_109 (Dropout)	(None, 16)	0
dense_136 (Dense)	(None, 32)	544
batch_normalization_110 (BatchNormalization)	(None, 32)	128
dropout_110 (Dropout)	(None, 32)	0
dense_137 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.1240 - loss: 3.1884 - val_accuracy: 0.1620 - val_loss: 2.9504
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1459 - loss: 2.9796 - val_accuracy: 0.1580 - val_loss: 2.9406
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1583 - loss: 2.9421 - val_accuracy: 0.1610 - val_loss: 2.9458
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1701 - loss: 2.9474 - val_accuracy: 0.2210 - val_loss: 2.7800
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1622 - loss: 2.9435 - val_accuracy: 0.1860 - val_loss: 2.6507
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1718 - loss: 2.9005 - val_accuracy: 0.2360 - val_loss: 2.8316
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1603 - loss: 2.9688 - val_accuracy: 0.2070 - val_loss: 2.6570
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1662 - loss: 2.9218 - val_accuracy: 0.2360 - val_loss: 2.8496
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1828 - loss: 2.9161 - val_accuracy: 0.2420 - val_loss: 2.6440
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1578 - loss: 2.9430 - val_accuracy: 0.2190 - val_loss: 2.7591
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1850 - loss: 2.9127 - val_accuracy: 0.1580 - val_loss: 3.1523
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1744 - loss: 2.9414 - val_accuracy: 0.2190 - val_loss: 2.7352
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1693 - loss: 2.9321 - val_accuracy: 0.2040 - val_loss: 2.7763
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1754 - loss: 2.8961 - val_accuracy: 0.2360 - val_loss: 2.8093
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1806 - loss: 2.9046 - val_accuracy: 0.1790 - val_loss: 2.8318
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1771 - loss: 2.9016 - val_accuracy: 0.1960 - val_loss: 2.6624
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1772 - loss: 2.9259 - val_accuracy: 0.1870 - val_loss: 2.6956
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1823 - loss: 2.8728 - val_accuracy: 0.2160 - val_loss: 2.7344
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1816 - loss: 2.8930 - val_accuracy: 0.2000 - val_loss: 2.7222
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1812 - loss: 2.8901 - val_accuracy: 0.1590 - val_loss: 2.8182
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1844 - loss: 2.8854 - val_accuracy: 0.1790 - val_loss: 2.7502
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1877 - loss: 2.9186 - val_accuracy: 0.2590 - val_loss: 2.6049
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1848 - loss: 2.8884 - val_accuracy: 0.1790 - val_loss: 2.7586
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1833 - loss: 2.8865 - val_accuracy: 0.2100 - val_loss: 2.7569
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1834 - loss: 2.8866 - val_accuracy: 0.2290 - val_loss: 2.7080
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1852 - loss: 2.8902 - val_accuracy: 0.2080 - val_loss: 2.7786
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1843 - loss: 2.8926 - val_accuracy: 0.2470 - val_loss: 2.7874
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1929 - loss: 2.8965 - val_accuracy: 0.2210 - val_loss: 2.6287
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1814 - loss: 2.9015 - val_accuracy: 0.2410 - val_loss: 2.6714
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.1876 - loss: 2.8678 - val_accuracy: 0.2500 - val_loss: 2.6608
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1834 - loss: 2.8928 - val_accuracy: 0.1730 - val_loss: 2.8121
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1851 - loss: 2.8938 - val_accuracy: 0.2500 - val_loss: 2.6643
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.1925 - loss: 2.8988 - val_accuracy: 0.1850 - val_loss: 2.7344
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1865 - loss: 2.8868 - val_accuracy: 0.2290 - val_loss: 2.7681
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1856 - loss: 2.9034 - val_accuracy: 0.2780 - val_loss: 2.6095
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.1946 - loss: 2.8714 - val_accuracy: 0.2340 - val_loss: 2.5943
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.1907 - loss: 2.8846 - val_accuracy: 0.2360 - val_loss: 2.5711
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1879 - loss: 2.8905 - val_accuracy: 0.1870 - val_loss: 2.6698
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1836 - loss: 2.9101 - val_accuracy: 0.2740 - val_loss: 2.5841
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1908 - loss: 2.8830 - val_accuracy: 0.2220 - val_loss: 2.6095
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1866 - loss: 2.8893 - val_accuracy: 0.2490 - val_loss: 2.6282
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.1925 - loss: 2.8883 - val_accuracy: 0.2030 - val_loss: 2.8606
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1802 - loss: 2.9154 - val_accuracy: 0.2290 - val_loss: 2.7916
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.1913 - loss: 2.8806 - val_accuracy: 0.2230 - val_loss: 2.7096
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1879 - loss: 2.8952 - val_accuracy: 0.2370 - val_loss: 2.7714
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1840 - loss: 2.8852 - val_accuracy: 0.2120 - val_loss: 2.8639
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1936 - loss: 2.8791 - val_accuracy: 0.1950 - val_loss: 2.6539
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1930 - loss: 2.8846 - val_accuracy: 0.2330 - val_loss: 2.7570
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1868 - loss: 2.8914 - val_accuracy: 0.2110 - val_loss: 2.7564
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.1915 - loss: 2.8826 - val_accuracy: 0.2630 - val_loss: 2.5889
Accuracy: 0.2530
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_27"

Layer (type)	Output Shape	Param #
dense_138 (Dense)	(None, 32)	352
batch_normalization_111 (BatchNormalization)	(None, 32)	128
dropout_111 (Dropout)	(None, 32)	0
dense_139 (Dense)	(None, 64)	2,112
batch_normalization_112 (BatchNormalization)	(None, 64)	256
dropout_112 (Dropout)	(None, 64)	0
dense_140 (Dense)	(None, 32)	2,080
batch_normalization_113 (BatchNormalization)	(None, 32)	128
dropout_113 (Dropout)	(None, 32)	0
dense_141 (Dense)	(None, 16)	528
batch_normalization_114 (BatchNormalization)	(None, 16)	64
dropout_114 (Dropout)	(None, 16)	0
dense_142 (Dense)	(None, 32)	544
batch_normalization_115 (BatchNormalization)	(None, 32)	128
dropout_115 (Dropout)	(None, 32)	0
dense_143 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.2302 - loss: 2.7664 - val_accuracy: 0.3510 - val_loss: 2.8479
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2957 - loss: 2.4599 - val_accuracy: 0.3210 - val_loss: 2.6770
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3075 - loss: 2.4468 - val_accuracy: 0.3420 - val_loss: 2.3964
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3220 - loss: 2.3915 - val_accuracy: 0.3340 - val_loss: 2.4171
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3336 - loss: 2.3574 - val_accuracy: 0.3490 - val_loss: 2.3555
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3344 - loss: 2.3599 - val_accuracy: 0.3890 - val_loss: 2.2288
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3390 - loss: 2.3394 - val_accuracy: 0.3470 - val_loss: 2.4673
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3378 - loss: 2.3744 - val_accuracy: 0.3750 - val_loss: 2.3649
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3425 - loss: 2.3607 - val_accuracy: 0.3470 - val_loss: 2.3215
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3422 - loss: 2.3542 - val_accuracy: 0.3630 - val_loss: 2.3433
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3498 - loss: 2.3094 - val_accuracy: 0.3470 - val_loss: 2.1934
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3438 - loss: 2.3675 - val_accuracy: 0.3980 - val_loss: 2.1586
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3533 - loss: 2.3222 - val_accuracy: 0.3380 - val_loss: 2.2812
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3571 - loss: 2.3072 - val_accuracy: 0.4030 - val_loss: 2.1962
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3538 - loss: 2.3131 - val_accuracy: 0.3850 - val_loss: 2.2756
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3622 - loss: 2.3002 - val_accuracy: 0.4110 - val_loss: 2.1164
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3618 - loss: 2.2929 - val_accuracy: 0.4160 - val_loss: 2.1845
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3620 - loss: 2.3163 - val_accuracy: 0.3770 - val_loss: 2.1955
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3690 - loss: 2.2982 - val_accuracy: 0.4380 - val_loss: 2.0555
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3620 - loss: 2.2954 - val_accuracy: 0.3880 - val_loss: 2.2467
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3604 - loss: 2.3128 - val_accuracy: 0.4460 - val_loss: 2.1156
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3752 - loss: 2.2702 - val_accuracy: 0.3390 - val_loss: 2.3660
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3652 - loss: 2.3096 - val_accuracy: 0.4080 - val_loss: 2.1688
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3675 - loss: 2.2926 - val_accuracy: 0.4050 - val_loss: 2.0701
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3737 - loss: 2.2506 - val_accuracy: 0.3560 - val_loss: 2.2885
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3679 - loss: 2.2991 - val_accuracy: 0.3680 - val_loss: 2.2519
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3667 - loss: 2.2928 - val_accuracy: 0.4370 - val_loss: 2.1143
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3596 - loss: 2.2965 - val_accuracy: 0.3890 - val_loss: 2.2290
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3742 - loss: 2.2998 - val_accuracy: 0.4170 - val_loss: 2.0911
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3709 - loss: 2.2903 - val_accuracy: 0.3600 - val_loss: 2.1725
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3676 - loss: 2.2878 - val_accuracy: 0.3080 - val_loss: 2.6122
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3798 - loss: 2.2670 - val_accuracy: 0.3540 - val_loss: 2.3831
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3697 - loss: 2.3140 - val_accuracy: 0.3680 - val_loss: 2.2788
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3664 - loss: 2.2858 - val_accuracy: 0.3980 - val_loss: 2.1663
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3698 - loss: 2.2636 - val_accuracy: 0.4680 - val_loss: 2.0312
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3761 - loss: 2.2705 - val_accuracy: 0.3980 - val_loss: 2.1627
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3787 - loss: 2.2559 - val_accuracy: 0.3920 - val_loss: 2.2693
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3792 - loss: 2.2676 - val_accuracy: 0.4150 - val_loss: 2.1272
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3703 - loss: 2.2904 - val_accuracy: 0.3770 - val_loss: 2.2312
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3757 - loss: 2.2707 - val_accuracy: 0.4220 - val_loss: 2.1716
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3841 - loss: 2.2496 - val_accuracy: 0.3540 - val_loss: 2.2061
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3745 - loss: 2.3014 - val_accuracy: 0.3640 - val_loss: 2.4481
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3859 - loss: 2.2607 - val_accuracy: 0.4260 - val_loss: 2.0817
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3680 - loss: 2.3017 - val_accuracy: 0.3700 - val_loss: 2.2178
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3859 - loss: 2.2731 - val_accuracy: 0.4260 - val_loss: 2.1190
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3807 - loss: 2.2782 - val_accuracy: 0.4360 - val_loss: 2.1410
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3833 - loss: 2.2482 - val_accuracy: 0.4110 - val_loss: 2.1867
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3754 - loss: 2.2779 - val_accuracy: 0.4270 - val_loss: 2.2209
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3862 - loss: 2.2469 - val_accuracy: 0.4170 - val_loss: 2.1086
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3882 - loss: 2.2385 - val_accuracy: 0.3960 - val_loss: 2.1624
Accuracy: 0.4090
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_28"

Layer (type)	Output Shape	Param #
dense_144 (Dense)	(None, 32)	352
batch_normalization_116 (BatchNormalization)	(None, 32)	128
dropout_116 (Dropout)	(None, 32)	0
dense_145 (Dense)	(None, 64)	2,112
batch_normalization_117 (BatchNormalization)	(None, 64)	256
dropout_117 (Dropout)	(None, 64)	0
dense_146 (Dense)	(None, 32)	2,080
batch_normalization_118 (BatchNormalization)	(None, 32)	128
dropout_118 (Dropout)	(None, 32)	0
dense_147 (Dense)	(None, 16)	528
batch_normalization_119 (BatchNormalization)	(None, 16)	64
dropout_119 (Dropout)	(None, 16)	0
dense_148 (Dense)	(None, 32)	544
batch_normalization_120 (BatchNormalization)	(None, 32)	128
dropout_120 (Dropout)	(None, 32)	0
dense_149 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

Non-trainable params: 352 (1.38 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.1819 - loss: 2.9646 - val_accuracy: 0.2160 - val_loss: 2.9670
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2401 - loss: 2.6157 - val_accuracy: 0.2140 - val_loss: 2.7058
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2434 - loss: 2.6267 - val_accuracy: 0.2500 - val_loss: 2.6213
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.2505 - loss: 2.6185 - val_accuracy: 0.3410 - val_loss: 2.4186
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2666 - loss: 2.5532 - val_accuracy: 0.3310 - val_loss: 2.4611
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2673 - loss: 2.5649 - val_accuracy: 0.3210 - val_loss: 2.3377
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2787 - loss: 2.5491 - val_accuracy: 0.3500 - val_loss: 2.3099
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2783 - loss: 2.5532 - val_accuracy: 0.3280 - val_loss: 2.3334
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2804 - loss: 2.5285 - val_accuracy: 0.3170 - val_loss: 2.4224
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2752 - loss: 2.5662 - val_accuracy: 0.3530 - val_loss: 2.3393
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2887 - loss: 2.5102 - val_accuracy: 0.3410 - val_loss: 2.3974
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2814 - loss: 2.5167 - val_accuracy: 0.3330 - val_loss: 2.4507
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2846 - loss: 2.5115 - val_accuracy: 0.3390 - val_loss: 2.3478
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.2908 - loss: 2.5168 - val_accuracy: 0.3730 - val_loss: 2.3398
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.2882 - loss: 2.5311 - val_accuracy: 0.2720 - val_loss: 2.7846
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2905 - loss: 2.5402 - val_accuracy: 0.3830 - val_loss: 2.2808
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2970 - loss: 2.5195 - val_accuracy: 0.2810 - val_loss: 2.4925
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2934 - loss: 2.5247 - val_accuracy: 0.3030 - val_loss: 2.3571
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2910 - loss: 2.5144 - val_accuracy: 0.3490 - val_loss: 2.4112
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2849 - loss: 2.5281 - val_accuracy: 0.3360 - val_loss: 2.3874
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2927 - loss: 2.5219 - val_accuracy: 0.3790 - val_loss: 2.1819
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3004 - loss: 2.4805 - val_accuracy: 0.3500 - val_loss: 2.3433
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2990 - loss: 2.5362 - val_accuracy: 0.2890 - val_loss: 2.5158
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3073 - loss: 2.4893 - val_accuracy: 0.3500 - val_loss: 2.2327
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3030 - loss: 2.4970 - val_accuracy: 0.3450 - val_loss: 2.3622
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.2934 - loss: 2.5363 - val_accuracy: 0.3060 - val_loss: 2.3956
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2948 - loss: 2.5178 - val_accuracy: 0.3500 - val_loss: 2.3851
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2935 - loss: 2.5164 - val_accuracy: 0.2860 - val_loss: 2.6188
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3022 - loss: 2.5096 - val_accuracy: 0.3740 - val_loss: 2.3328
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3008 - loss: 2.5245 - val_accuracy: 0.3340 - val_loss: 2.3099
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3065 - loss: 2.5290 - val_accuracy: 0.4100 - val_loss: 2.1570
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2999 - loss: 2.5020 - val_accuracy: 0.3810 - val_loss: 2.2566
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3004 - loss: 2.5205 - val_accuracy: 0.3400 - val_loss: 2.4048
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2971 - loss: 2.5367 - val_accuracy: 0.3330 - val_loss: 2.3539
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2900 - loss: 2.5407 - val_accuracy: 0.3140 - val_loss: 2.5005
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3037 - loss: 2.5141 - val_accuracy: 0.3360 - val_loss: 2.3127
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2963 - loss: 2.4896 - val_accuracy: 0.3820 - val_loss: 2.2715
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3001 - loss: 2.4908 - val_accuracy: 0.3230 - val_loss: 2.4074
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2966 - loss: 2.5013 - val_accuracy: 0.3540 - val_loss: 2.3364
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3018 - loss: 2.5061 - val_accuracy: 0.3300 - val_loss: 2.3913
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3038 - loss: 2.5062 - val_accuracy: 0.3670 - val_loss: 2.2311
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3111 - loss: 2.4904 - val_accuracy: 0.3620 - val_loss: 2.2827
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3056 - loss: 2.5141 - val_accuracy: 0.3890 - val_loss: 2.2917
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3078 - loss: 2.5087 - val_accuracy: 0.3830 - val_loss: 2.2025
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3075 - loss: 2.4927 - val_accuracy: 0.3730 - val_loss: 2.2591
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3161 - loss: 2.4660 - val_accuracy: 0.3450 - val_loss: 2.3396
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.3070 - loss: 2.5050 - val_accuracy: 0.3690 - val_loss: 2.3488
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3082 - loss: 2.5048 - val_accuracy: 0.4100 - val_loss: 2.2526
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3036 - loss: 2.5097 - val_accuracy: 0.3770 - val_loss: 2.3539
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3022 - loss: 2.5117 - val_accuracy: 0.3860 - val_loss: 2.3124
Accuracy: 0.3770
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_29"

Layer (type)	Output Shape	Param #
dense_150 (Dense)	(None, 32)	352
batch_normalization_121 (BatchNormalization)	(None, 32)	128
dropout_121 (Dropout)	(None, 32)	0
dense_151 (Dense)	(None, 64)	2,112
batch_normalization_122 (BatchNormalization)	(None, 64)	256
dropout_122 (Dropout)	(None, 64)	0
dense_152 (Dense)	(None, 32)	2,080
batch_normalization_123 (BatchNormalization)	(None, 32)	128
dropout_123 (Dropout)	(None, 32)	0
dense_153 (Dense)	(None, 16)	528
batch_normalization_124 (BatchNormalization)	(None, 16)	64
dropout_124 (Dropout)	(None, 16)	0
dense_154 (Dense)	(None, 32)	544
batch_normalization_125 (BatchNormalization)	(None, 32)	128
dropout_125 (Dropout)	(None, 32)	0
dense_155 (Dense)	(None, 20)	660

Total params: 6,980 (27.27 KB)

Trainable params: 6,628 (25.89 KB)

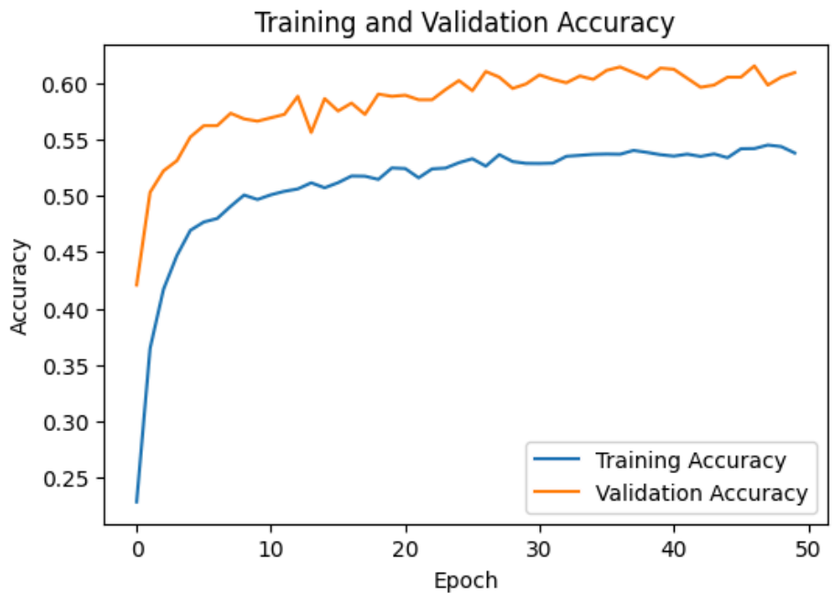
Non-trainable params: 352 (1.38 KB)

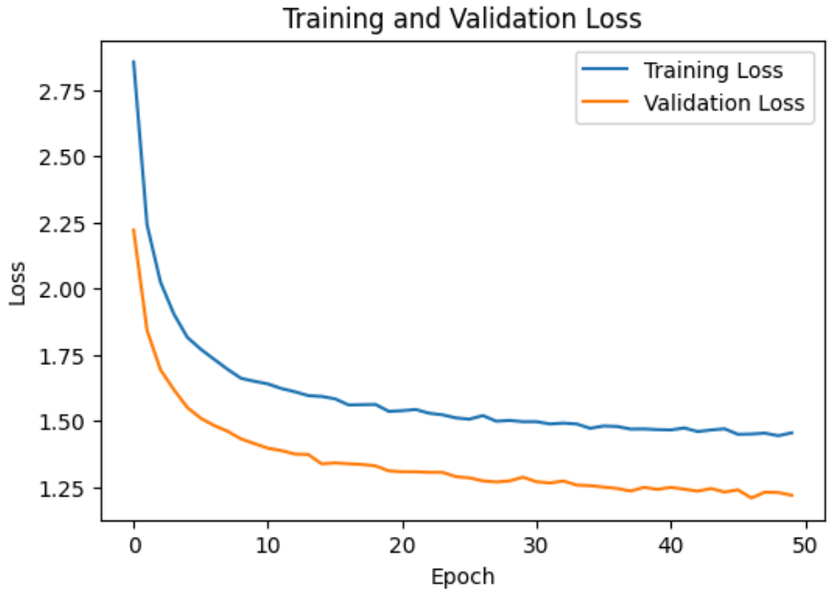
```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.1458 - loss: 3.0668 - val_accuracy: 0.1870 - val_loss: 2.7679
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.1904 - loss: 2.7690 - val_accuracy: 0.2000 - val_loss: 2.6431
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2032 - loss: 2.7721 - val_accuracy: 0.2100 - val_loss: 2.7980
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.1931 - loss: 2.7630 - val_accuracy: 0.2920 - val_loss: 2.6222
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2134 - loss: 2.7471 - val_accuracy: 0.2310 - val_loss: 2.5261
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2122 - loss: 2.7403 - val_accuracy: 0.2700 - val_loss: 2.4969
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2226 - loss: 2.6977 - val_accuracy: 0.3000 - val_loss: 2.3506
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2200 - loss: 2.7013 - val_accuracy: 0.2770 - val_loss: 2.4976
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2281 - loss: 2.7009 - val_accuracy: 0.2890 - val_loss: 2.5185
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2225 - loss: 2.7505 - val_accuracy: 0.2100 - val_loss: 2.5921
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2304 - loss: 2.6841 - val_accuracy: 0.2870 - val_loss: 2.5642
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2229 - loss: 2.7236 - val_accuracy: 0.2960 - val_loss: 2.4623
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2363 - loss: 2.6643 - val_accuracy: 0.2860 - val_loss: 2.4948
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.2362 - loss: 2.7072 - val_accuracy: 0.2260 - val_loss: 2.5876
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2236 - loss: 2.7129 - val_accuracy: 0.2770 - val_loss: 2.5634
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2303 - loss: 2.6645 - val_accuracy: 0.2690 - val_loss: 2.6255
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2294 - loss: 2.6946 - val_accuracy: 0.2320 - val_loss: 2.6450
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2335 - loss: 2.6817 - val_accuracy: 0.2910 - val_loss: 2.3833
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2281 - loss: 2.7225 - val_accuracy: 0.2900 - val_loss: 2.5074
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2310 - loss: 2.6914 - val_accuracy: 0.2940 - val_loss: 2.4766
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2407 - loss: 2.6819 - val_accuracy: 0.2440 - val_loss: 2.4456
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2368 - loss: 2.6694 - val_accuracy: 0.2100 - val_loss: 2.4603
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2388 - loss: 2.6893 - val_accuracy: 0.3310 - val_loss: 2.3677
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2300 - loss: 2.6833 - val_accuracy: 0.2600 - val_loss: 2.6282
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2317 - loss: 2.7051 - val_accuracy: 0.2800 - val_loss: 2.4753
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2296 - loss: 2.7001 - val_accuracy: 0.3220 - val_loss: 2.4320
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2330 - loss: 2.6756 - val_accuracy: 0.2590 - val_loss: 2.5363
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2399 - loss: 2.6944 - val_accuracy: 0.2650 - val_loss: 2.5416
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2306 - loss: 2.6834 - val_accuracy: 0.2450 - val_loss: 2.4926
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2392 - loss: 2.6781 - val_accuracy: 0.2710 - val_loss: 2.5196
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2438 - loss: 2.6707 - val_accuracy: 0.3120 - val_loss: 2.5309
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2466 - loss: 2.6869 - val_accuracy: 0.3150 - val_loss: 2.4244
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2394 - loss: 2.6556 - val_accuracy: 0.2910 - val_loss: 2.4420
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2421 - loss: 2.6819 - val_accuracy: 0.2910 - val_loss: 2.4573
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.2355 - loss: 2.6741 - val_accuracy: 0.3510 - val_loss: 2.3192
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2384 - loss: 2.6706 - val_accuracy: 0.2700 - val_loss: 2.5695
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2409 - loss: 2.6931 - val_accuracy: 0.3340 - val_loss: 2.4139
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2527 - loss: 2.6535 - val_accuracy: 0.3150 - val_loss: 2.4267
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2417 - loss: 2.6815 - val_accuracy: 0.2880 - val_loss: 2.5729
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2549 - loss: 2.6503 - val_accuracy: 0.2380 - val_loss: 2.5912
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2392 - loss: 2.6919 - val_accuracy: 0.2620 - val_loss: 2.5023
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2345 - loss: 2.6772 - val_accuracy: 0.2630 - val_loss: 2.4882
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2548 - loss: 2.6647 - val_accuracy: 0.2810 - val_loss: 2.4213
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2481 - loss: 2.6721 - val_accuracy: 0.2750 - val_loss: 2.4581
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2418 - loss: 2.6819 - val_accuracy: 0.2980 - val_loss: 2.4543
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.2462 - loss: 2.6797 - val_accuracy: 0.2750 - val_loss: 2.4565
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2411 - loss: 2.6812 - val_accuracy: 0.3060 - val_loss: 2.4075
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2469 - loss: 2.6560 - val_accuracy: 0.2900 - val_loss: 2.4483
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2384 - loss: 2.6823 - val_accuracy: 0.3060 - val_loss: 2.4029
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.2526 - loss: 2.6440 - val_accuracy: 0.3000 - val_loss: 2.4628
Accuracy: 0.2825

Best Hyperparameters:
{'learning_rate': 0.001, 'batch_size': 64, 'dropout': 0.1}
Best Accuracy: 0.6040
```

In [ ]:
```
plt.figure(figsize=(6, 4))
plt.plot(best_history.history['accuracy'], label='Training Accuracy')
plt.plot(best_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.figure(figsize=(6, 4))
plt.plot(best_history.history['loss'], label='Training Loss')
plt.plot(best_history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [ ]:
```
best_model.evaluate(X_test, y_test)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6100 - loss: 1.2808
```

Out[ ]:
```
[1.27437162399292, 0.6039999723434448]
```

10 Layer Neural Network

In [ ]:
```
learning_rates = [0.001, 0.01,0.0001]
batch_sizes = [32, 64]
dropout = [0.1,0.2]
best_accuracy = 0
input_shape = X_train.shape[1:]
best_history = None
best_model = None

for lr in learning_rates:
    for batch in batch_sizes:
        for d in dropout:
            model = Ten_Layer_NN()
            config = {
                        'input_shape': input_shape,
                        'epochs': 50,
                        'dropout': d,
                        'batch_size': batch,
                        'lr': lr
                    }
            model.build_model(config)
            history = model.train(X_train, y_train, X_valid, y_valid, config)
            loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
```

```
            print(f"Accuracy: {accuracy:.4f}")

            if accuracy > best_accuracy:
                best_model = model
                best_accuracy = accuracy
                best_params = {'learning_rate': lr, 'batch_size': batch, 'dropout': d}
                best_history = history

print("\nBest Hyperparameters:")
print(best_params)
print(f"Best Accuracy: {best_accuracy:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_30"

Layer (type)	Output Shape	Param #
dense_156 (Dense)	(None, 32)	352
batch_normalization_126 (BatchNormalization)	(None, 32)	128
dropout_126 (Dropout)	(None, 32)	0
dense_157 (Dense)	(None, 64)	2,112
batch_normalization_127 (BatchNormalization)	(None, 64)	256
dropout_127 (Dropout)	(None, 64)	0
dense_158 (Dense)	(None, 128)	8,320
batch_normalization_128 (BatchNormalization)	(None, 128)	512
dropout_128 (Dropout)	(None, 128)	0
dense_159 (Dense)	(None, 64)	8,256
batch_normalization_129 (BatchNormalization)	(None, 64)	256
dropout_129 (Dropout)	(None, 64)	0
dense_160 (Dense)	(None, 32)	2,080
batch_normalization_130 (BatchNormalization)	(None, 32)	128
dense_161 (Dense)	(None, 16)	528
batch_normalization_131 (BatchNormalization)	(None, 16)	64
dropout_130 (Dropout)	(None, 16)	0
dense_162 (Dense)	(None, 8)	136
batch_normalization_132 (BatchNormalization)	(None, 8)	32
dropout_131 (Dropout)	(None, 8)	0
dense_163 (Dense)	(None, 16)	144
batch_normalization_133 (BatchNormalization)	(None, 16)	64
dropout_132 (Dropout)	(None, 16)	0

dense_164 (Dense)	(None, 32)	544
batch_normalization_134 (BatchNormalization)	(None, 32)	128
dropout_133 (Dropout)	(None, 32)	0
dense_165 (Dense)	(None, 20)	660

Total params: 24,700 (96.48 KB)

Trainable params: 23,916 (93.42 KB)

Non-trainable params: 784 (3.06 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.1039 - loss: 3.4464 - val_accuracy: 0.2840 - val_loss: 2.4897
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2492 - loss: 2.6182 - val_accuracy: 0.4170 - val_loss: 2.0357
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3261 - loss: 2.3207 - val_accuracy: 0.4690 - val_loss: 1.8317
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3502 - loss: 2.1546 - val_accuracy: 0.5010 - val_loss: 1.7345
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3849 - loss: 2.0567 - val_accuracy: 0.5110 - val_loss: 1.6658
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3992 - loss: 2.0023 - val_accuracy: 0.5320 - val_loss: 1.6228
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4141 - loss: 1.9537 - val_accuracy: 0.5250 - val_loss: 1.5884
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4233 - loss: 1.9103 - val_accuracy: 0.5440 - val_loss: 1.5505
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4312 - loss: 1.9019 - val_accuracy: 0.5330 - val_loss: 1.5383
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4222 - loss: 1.8930 - val_accuracy: 0.5320 - val_loss: 1.5470
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4410 - loss: 1.8471 - val_accuracy: 0.5570 - val_loss: 1.4872
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4461 - loss: 1.8355 - val_accuracy: 0.5490 - val_loss: 1.4795
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4371 - loss: 1.8276 - val_accuracy: 0.5490 - val_loss: 1.4698
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4383 - loss: 1.8025 - val_accuracy: 0.5380 - val_loss: 1.4755
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4528 - loss: 1.8145 - val_accuracy: 0.5710 - val_loss: 1.4260
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4659 - loss: 1.7594 - val_accuracy: 0.5720 - val_loss: 1.4509
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4570 - loss: 1.7733 - val_accuracy: 0.5650 - val_loss: 1.4346
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4584 - loss: 1.7681 - val_accuracy: 0.5780 - val_loss: 1.4008
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4688 - loss: 1.7373 - val_accuracy: 0.5720 - val_loss: 1.4109
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4712 - loss: 1.7185 - val_accuracy: 0.5880 - val_loss: 1.3932
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.4671 - loss: 1.7493 - val_accuracy: 0.5900 - val_loss: 1.3801
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4753 - loss: 1.7252 - val_accuracy: 0.5820 - val_loss: 1.3882
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4697 - loss: 1.7282 - val_accuracy: 0.5900 - val_loss: 1.3481
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4661 - loss: 1.7234 - val_accuracy: 0.5690 - val_loss: 1.3726
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4702 - loss: 1.7319 - val_accuracy: 0.5930 - val_loss: 1.3497
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.4732 - loss: 1.7120 - val_accuracy: 0.5900 - val_loss: 1.3500
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4810 - loss: 1.6979 - val_accuracy: 0.5940 - val_loss: 1.3350
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4798 - loss: 1.6902 - val_accuracy: 0.6090 - val_loss: 1.3314
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4837 - loss: 1.6870 - val_accuracy: 0.5900 - val_loss: 1.3563
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4778 - loss: 1.6811 - val_accuracy: 0.5940 - val_loss: 1.3228
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4803 - loss: 1.6812 - val_accuracy: 0.6010 - val_loss: 1.3133
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4863 - loss: 1.6688 - val_accuracy: 0.5820 - val_loss: 1.3444
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4838 - loss: 1.6928 - val_accuracy: 0.5920 - val_loss: 1.3244
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.4907 - loss: 1.6616 - val_accuracy: 0.5930 - val_loss: 1.3040
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4980 - loss: 1.6510 - val_accuracy: 0.5940 - val_loss: 1.3041
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4800 - loss: 1.6734 - val_accuracy: 0.6020 - val_loss: 1.3117
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4881 - loss: 1.6577 - val_accuracy: 0.5950 - val_loss: 1.3177
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4961 - loss: 1.6319 - val_accuracy: 0.5710 - val_loss: 1.3279
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4893 - loss: 1.6733 - val_accuracy: 0.6010 - val_loss: 1.2955
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.4844 - loss: 1.6577 - val_accuracy: 0.5920 - val_loss: 1.3072
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.4828 - loss: 1.6576 - val_accuracy: 0.5820 - val_loss: 1.3053
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4982 - loss: 1.6417 - val_accuracy: 0.5970 - val_loss: 1.3006
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4992 - loss: 1.6378 - val_accuracy: 0.5960 - val_loss: 1.2906
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4960 - loss: 1.6298 - val_accuracy: 0.5890 - val_loss: 1.3109
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.4932 - loss: 1.6437 - val_accuracy: 0.6020 - val_loss: 1.2907
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4982 - loss: 1.6168 - val_accuracy: 0.5920 - val_loss: 1.2973
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4953 - loss: 1.6358 - val_accuracy: 0.5950 - val_loss: 1.2814
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5017 - loss: 1.6193 - val_accuracy: 0.5890 - val_loss: 1.2962
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4981 - loss: 1.6259 - val_accuracy: 0.5960 - val_loss: 1.2847
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4965 - loss: 1.6360 - val_accuracy: 0.5820 - val_loss: 1.3218
Accuracy: 0.5735
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_31"

Layer (type)	Output Shape	Param #
dense_166 (Dense)	(None, 32)	352
batch_normalization_135 (BatchNormalization)	(None, 32)	128
dropout_134 (Dropout)	(None, 32)	0
dense_167 (Dense)	(None, 64)	2,112
batch_normalization_136 (BatchNormalization)	(None, 64)	256
dropout_135 (Dropout)	(None, 64)	0
dense_168 (Dense)	(None, 128)	8,320
batch_normalization_137 (BatchNormalization)	(None, 128)	512
dropout_136 (Dropout)	(None, 128)	0
dense_169 (Dense)	(None, 64)	8,256
batch_normalization_138 (BatchNormalization)	(None, 64)	256
dropout_137 (Dropout)	(None, 64)	0
dense_170 (Dense)	(None, 32)	2,080
batch_normalization_139 (BatchNormalization)	(None, 32)	128
dense_171 (Dense)	(None, 16)	528
batch_normalization_140 (BatchNormalization)	(None, 16)	64
dropout_138 (Dropout)	(None, 16)	0
dense_172 (Dense)	(None, 8)	136
batch_normalization_141 (BatchNormalization)	(None, 8)	32
dropout_139 (Dropout)	(None, 8)	0
dense_173 (Dense)	(None, 16)	144
batch_normalization_142 (BatchNormalization)	(None, 16)	64
dropout_140 (Dropout)	(None, 16)	0

dense_174 (Dense)	(None, 32)	544
batch_normalization_143 (BatchNormalization)	(None, 32)	128
dropout_141 (Dropout)	(None, 32)	0
dense_175 (Dense)	(None, 20)	660

Total params: 24,700 (96.48 KB)

Trainable params: 23,916 (93.42 KB)

Non-trainable params: 784 (3.06 KB)

```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.0652 - loss: 3.6193 - val_accuracy: 0.2160 - val_loss: 2.6899
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.1364 - loss: 2.8716 - val_accuracy: 0.2640 - val_loss: 2.4207
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.1969 - loss: 2.6125 - val_accuracy: 0.3030 - val_loss: 2.2435
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.2173 - loss: 2.4785 - val_accuracy: 0.3220 - val_loss: 2.1304
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.2494 - loss: 2.3779 - val_accuracy: 0.3550 - val_loss: 2.0285
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2502 - loss: 2.3063 - val_accuracy: 0.3630 - val_loss: 1.9734
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2790 - loss: 2.2437 - val_accuracy: 0.4030 - val_loss: 1.8944
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.2960 - loss: 2.2006 - val_accuracy: 0.4170 - val_loss: 1.8358
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.2993 - loss: 2.1694 - val_accuracy: 0.4350 - val_loss: 1.8128
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.3007 - loss: 2.1665 - val_accuracy: 0.4390 - val_loss: 1.7946
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3220 - loss: 2.1210 - val_accuracy: 0.4250 - val_loss: 1.7651
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3231 - loss: 2.1034 - val_accuracy: 0.4710 - val_loss: 1.7243
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3349 - loss: 2.0818 - val_accuracy: 0.4320 - val_loss: 1.7344
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3309 - loss: 2.0938 - val_accuracy: 0.4590 - val_loss: 1.6916
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3310 - loss: 2.0755 - val_accuracy: 0.4690 - val_loss: 1.6853
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3380 - loss: 2.0558 - val_accuracy: 0.4670 - val_loss: 1.6923
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3571 - loss: 2.0320 - val_accuracy: 0.4640 - val_loss: 1.6624
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3522 - loss: 2.0347 - val_accuracy: 0.4770 - val_loss: 1.6580
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3583 - loss: 2.0143 - val_accuracy: 0.4940 - val_loss: 1.6258
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.3540 - loss: 2.0216 - val_accuracy: 0.4810 - val_loss: 1.6305
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3659 - loss: 2.0059 - val_accuracy: 0.5010 - val_loss: 1.6323
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3700 - loss: 1.9992 - val_accuracy: 0.5090 - val_loss: 1.6017
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.3678 - loss: 1.9924 - val_accuracy: 0.4880 - val_loss: 1.5810
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3737 - loss: 1.9745 - val_accuracy: 0.5060 - val_loss: 1.6082
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3771 - loss: 1.9779 - val_accuracy: 0.5130 - val_loss: 1.5856
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3727 - loss: 1.9790 - val_accuracy: 0.5150 - val_loss: 1.5969
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3794 - loss: 1.9632 - val_accuracy: 0.5200 - val_loss: 1.5693
```

```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.3746 - loss: 1.9693 - val_accuracy: 0.5020 - val_loss: 1.5882
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3766 - loss: 1.9604 - val_accuracy: 0.5110 - val_loss: 1.5553
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3856 - loss: 1.9331 - val_accuracy: 0.5290 - val_loss: 1.5361
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3863 - loss: 1.9434 - val_accuracy: 0.5260 - val_loss: 1.5274
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3951 - loss: 1.9275 - val_accuracy: 0.5140 - val_loss: 1.5341
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3883 - loss: 1.9496 - val_accuracy: 0.5400 - val_loss: 1.5324
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3878 - loss: 1.9355 - val_accuracy: 0.5410 - val_loss: 1.5319
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.3900 - loss: 1.9399 - val_accuracy: 0.5180 - val_loss: 1.5051
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.3984 - loss: 1.9106 - val_accuracy: 0.5530 - val_loss: 1.4851
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3987 - loss: 1.9188 - val_accuracy: 0.5280 - val_loss: 1.5098
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3969 - loss: 1.9387 - val_accuracy: 0.5410 - val_loss: 1.5139
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4027 - loss: 1.9159 - val_accuracy: 0.5360 - val_loss: 1.5084
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.4081 - loss: 1.9149 - val_accuracy: 0.5380 - val_loss: 1.5132
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4005 - loss: 1.9038 - val_accuracy: 0.5460 - val_loss: 1.4955
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4034 - loss: 1.9232 - val_accuracy: 0.5500 - val_loss: 1.4913
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4035 - loss: 1.9145 - val_accuracy: 0.5280 - val_loss: 1.5002
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.4031 - loss: 1.9040 - val_accuracy: 0.5330 - val_loss: 1.5085
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4053 - loss: 1.8867 - val_accuracy: 0.5450 - val_loss: 1.4821
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4135 - loss: 1.8904 - val_accuracy: 0.5500 - val_loss: 1.4905
Epoch 47/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.4107 - loss: 1.8913 - val_accuracy: 0.5410 - val_loss: 1.4816
Epoch 48/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4118 - loss: 1.9111 - val_accuracy: 0.5550 - val_loss: 1.4687
Epoch 49/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4155 - loss: 1.8979 - val_accuracy: 0.5530 - val_loss: 1.4780
Epoch 50/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.4107 - loss: 1.8992 - val_accuracy: 0.5520 - val_loss: 1.4823
Accuracy: 0.5465
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_32"

Layer (type)	Output Shape	Param #
dense_176 (Dense)	(None, 32)	352
batch_normalization_144 (BatchNormalization)	(None, 32)	128
dropout_142 (Dropout)	(None, 32)	0
dense_177 (Dense)	(None, 64)	2,112
batch_normalization_145 (BatchNormalization)	(None, 64)	256
dropout_143 (Dropout)	(None, 64)	0
dense_178 (Dense)	(None, 128)	8,320
batch_normalization_146 (BatchNormalization)	(None, 128)	512
dropout_144 (Dropout)	(None, 128)	0
dense_179 (Dense)	(None, 64)	8,256
batch_normalization_147 (BatchNormalization)	(None, 64)	256
dropout_145 (Dropout)	(None, 64)	0
dense_180 (Dense)	(None, 32)	2,080
batch_normalization_148 (BatchNormalization)	(None, 32)	128
dense_181 (Dense)	(None, 16)	528
batch_normalization_149 (BatchNormalization)	(None, 16)	64
dropout_146 (Dropout)	(None, 16)	0
dense_182 (Dense)	(None, 8)	136
batch_normalization_150 (BatchNormalization)	(None, 8)	32
dropout_147 (Dropout)	(None, 8)	0
dense_183 (Dense)	(None, 16)	144
batch_normalization_151 (BatchNormalization)	(None, 16)	64
dropout_148 (Dropout)	(None, 16)	0

dense_184 (Dense)	(None, 32)	544
batch_normalization_152 (BatchNormalization)	(None, 32)	128
dropout_149 (Dropout)	(None, 32)	0
dense_185 (Dense)	(None, 20)	660

Total params: 24,700 (96.48 KB)

Trainable params: 23,916 (93.42 KB)

Non-trainable params: 784 (3.06 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.0837 - loss: 3.4780 - val_accuracy: 0.2820 - val_loss: 2.5967
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.2366 - loss: 2.6812 - val_accuracy: 0.3750 - val_loss: 2.1362
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3069 - loss: 2.3918 - val_accuracy: 0.4480 - val_loss: 1.9425
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3446 - loss: 2.2142 - val_accuracy: 0.4790 - val_loss: 1.8183
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3786 - loss: 2.1020 - val_accuracy: 0.5040 - val_loss: 1.7367
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3948 - loss: 2.0290 - val_accuracy: 0.5150 - val_loss: 1.6753
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4021 - loss: 1.9782 - val_accuracy: 0.5290 - val_loss: 1.6197
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.4242 - loss: 1.9354 - val_accuracy: 0.5350 - val_loss: 1.5907
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4277 - loss: 1.8975 - val_accuracy: 0.5400 - val_loss: 1.5364
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4424 - loss: 1.8506 - val_accuracy: 0.5440 - val_loss: 1.5290
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4409 - loss: 1.8281 - val_accuracy: 0.5480 - val_loss: 1.4947
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4509 - loss: 1.8189 - val_accuracy: 0.5590 - val_loss: 1.4766
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4532 - loss: 1.7794 - val_accuracy: 0.5680 - val_loss: 1.4700
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4589 - loss: 1.7861 - val_accuracy: 0.5700 - val_loss: 1.4582
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.4760 - loss: 1.7476 - val_accuracy: 0.5810 - val_loss: 1.4343
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4716 - loss: 1.7396 - val_accuracy: 0.5740 - val_loss: 1.4221
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4709 - loss: 1.7504 - val_accuracy: 0.5480 - val_loss: 1.4455
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4780 - loss: 1.7185 - val_accuracy: 0.5710 - val_loss: 1.4039
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4828 - loss: 1.7050 - val_accuracy: 0.5920 - val_loss: 1.3790
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4863 - loss: 1.6852 - val_accuracy: 0.5840 - val_loss: 1.3842
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4851 - loss: 1.6964 - val_accuracy: 0.5780 - val_loss: 1.3581
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.4976 - loss: 1.6663 - val_accuracy: 0.5930 - val_loss: 1.3692
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4986 - loss: 1.6645 - val_accuracy: 0.5810 - val_loss: 1.3694
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5007 - loss: 1.6570 - val_accuracy: 0.5730 - val_loss: 1.3621
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4966 - loss: 1.6625 - val_accuracy: 0.5810 - val_loss: 1.3647
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4948 - loss: 1.6372 - val_accuracy: 0.5820 - val_loss: 1.3597
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5065 - loss: 1.6247 - val_accuracy: 0.5980 - val_loss: 1.3290
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.5076 - loss: 1.6291 - val_accuracy: 0.5900 - val_loss: 1.3368
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5069 - loss: 1.6085 - val_accuracy: 0.5980 - val_loss: 1.3389
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5071 - loss: 1.6173 - val_accuracy: 0.5900 - val_loss: 1.3230
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5076 - loss: 1.5983 - val_accuracy: 0.5980 - val_loss: 1.3383
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5127 - loss: 1.5987 - val_accuracy: 0.6010 - val_loss: 1.3029
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.5004 - loss: 1.6281 - val_accuracy: 0.5970 - val_loss: 1.2927
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.5162 - loss: 1.5745 - val_accuracy: 0.6050 - val_loss: 1.2997
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5025 - loss: 1.6087 - val_accuracy: 0.5920 - val_loss: 1.2884
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5091 - loss: 1.6107 - val_accuracy: 0.5960 - val_loss: 1.3061
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5222 - loss: 1.5857 - val_accuracy: 0.5970 - val_loss: 1.2941
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5158 - loss: 1.5889 - val_accuracy: 0.6030 - val_loss: 1.2998
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5225 - loss: 1.5752 - val_accuracy: 0.5880 - val_loss: 1.2917
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5068 - loss: 1.5897 - val_accuracy: 0.6050 - val_loss: 1.2888
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.5168 - loss: 1.5768 - val_accuracy: 0.6010 - val_loss: 1.2974
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.5155 - loss: 1.5662 - val_accuracy: 0.6020 - val_loss: 1.2850
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5209 - loss: 1.5639 - val_accuracy: 0.6000 - val_loss: 1.2840
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5241 - loss: 1.5601 - val_accuracy: 0.5990 - val_loss: 1.2793
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5221 - loss: 1.5610 - val_accuracy: 0.6040 - val_loss: 1.2690
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5213 - loss: 1.5622 - val_accuracy: 0.6090 - val_loss: 1.2828
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5139 - loss: 1.5793 - val_accuracy: 0.6040 - val_loss: 1.2760
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.5195 - loss: 1.5680 - val_accuracy: 0.6030 - val_loss: 1.2823
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.5204 - loss: 1.5667 - val_accuracy: 0.5930 - val_loss: 1.2598
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5279 - loss: 1.5438 - val_accuracy: 0.6170 - val_loss: 1.2571
Accuracy: 0.5915
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_33"

Layer (type)	Output Shape	Param #
dense_186 (Dense)	(None, 32)	352
batch_normalization_153 (BatchNormalization)	(None, 32)	128
dropout_150 (Dropout)	(None, 32)	0
dense_187 (Dense)	(None, 64)	2,112
batch_normalization_154 (BatchNormalization)	(None, 64)	256
dropout_151 (Dropout)	(None, 64)	0
dense_188 (Dense)	(None, 128)	8,320
batch_normalization_155 (BatchNormalization)	(None, 128)	512
dropout_152 (Dropout)	(None, 128)	0
dense_189 (Dense)	(None, 64)	8,256
batch_normalization_156 (BatchNormalization)	(None, 64)	256
dropout_153 (Dropout)	(None, 64)	0
dense_190 (Dense)	(None, 32)	2,080
batch_normalization_157 (BatchNormalization)	(None, 32)	128
dense_191 (Dense)	(None, 16)	528
batch_normalization_158 (BatchNormalization)	(None, 16)	64
dropout_154 (Dropout)	(None, 16)	0
dense_192 (Dense)	(None, 8)	136
batch_normalization_159 (BatchNormalization)	(None, 8)	32
dropout_155 (Dropout)	(None, 8)	0
dense_193 (Dense)	(None, 16)	144
batch_normalization_160 (BatchNormalization)	(None, 16)	64
dropout_156 (Dropout)	(None, 16)	0

dense_194 (Dense)	(None, 32)	544
batch_normalization_161 (BatchNormalization)	(None, 32)	128
dropout_157 (Dropout)	(None, 32)	0
dense_195 (Dense)	(None, 20)	660

Total params: 24,700 (96.48 KB)

Trainable params: 23,916 (93.42 KB)

Non-trainable params: 784 (3.06 KB)

```
None
Model compiled.
Epoch 1/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.0654 - loss: 3.6842 - val_accuracy: 0.2330 - val_loss: 2.8401
Epoch 2/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.1357 - loss: 2.9722 - val_accuracy: 0.3030 - val_loss: 2.3648
Epoch 3/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.1888 - loss: 2.6483 - val_accuracy: 0.3310 - val_loss: 2.2186
Epoch 4/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2266 - loss: 2.5008 - val_accuracy: 0.3730 - val_loss: 2.1142
Epoch 5/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2522 - loss: 2.4020 - val_accuracy: 0.3770 - val_loss: 2.0370
Epoch 6/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.2746 - loss: 2.3296 - val_accuracy: 0.3830 - val_loss: 1.9656
Epoch 7/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2839 - loss: 2.2631 - val_accuracy: 0.4040 - val_loss: 1.9207
Epoch 8/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3028 - loss: 2.2033 - val_accuracy: 0.4140 - val_loss: 1.8665
Epoch 9/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.3086 - loss: 2.1675 - val_accuracy: 0.4270 - val_loss: 1.8373
Epoch 10/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3164 - loss: 2.1328 - val_accuracy: 0.4370 - val_loss: 1.8022
Epoch 11/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3187 - loss: 2.1192 - val_accuracy: 0.4510 - val_loss: 1.7789
Epoch 12/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3253 - loss: 2.0856 - val_accuracy: 0.4520 - val_loss: 1.7632
Epoch 13/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3289 - loss: 2.0785 - val_accuracy: 0.4540 - val_loss: 1.7368
Epoch 14/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3365 - loss: 2.0616 - val_accuracy: 0.4540 - val_loss: 1.7317
Epoch 15/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.3504 - loss: 2.0281 - val_accuracy: 0.4620 - val_loss: 1.6992
Epoch 16/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3430 - loss: 2.0281 - val_accuracy: 0.4880 - val_loss: 1.6798
Epoch 17/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3479 - loss: 2.0228 - val_accuracy: 0.4810 - val_loss: 1.6636
Epoch 18/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3496 - loss: 2.0091 - val_accuracy: 0.4740 - val_loss: 1.6453
Epoch 19/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3603 - loss: 1.9954 - val_accuracy: 0.4990 - val_loss: 1.6369
Epoch 20/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3682 - loss: 1.9816 - val_accuracy: 0.4780 - val_loss: 1.6354
Epoch 21/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.3665 - loss: 1.9814 - val_accuracy: 0.4960 - val_loss: 1.6107
Epoch 22/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.3690 - loss: 1.9655 - val_accuracy: 0.5010 - val_loss: 1.6013
Epoch 23/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3632 - loss: 1.9751 - val_accuracy: 0.4900 - val_loss: 1.5917
Epoch 24/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.3895 - loss: 1.9210 - val_accuracy: 0.5020 - val_loss: 1.5903
Epoch 25/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3844 - loss: 1.9467 - val_accuracy: 0.5110 - val_loss: 1.5833
Epoch 26/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3987 - loss: 1.9220 - val_accuracy: 0.5130 - val_loss: 1.5562
Epoch 27/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.3812 - loss: 1.9307 - val_accuracy: 0.5150 - val_loss: 1.5572
```

```
Epoch 28/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3899 - loss: 1.9235 - val_accuracy: 0.5310 - val_loss: 1.5506
Epoch 29/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.3945 - loss: 1.9107 - val_accuracy: 0.5110 - val_loss: 1.5606
Epoch 30/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3886 - loss: 1.9188 - val_accuracy: 0.5220 - val_loss: 1.5406
Epoch 31/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4045 - loss: 1.8780 - val_accuracy: 0.5150 - val_loss: 1.5293
Epoch 32/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4048 - loss: 1.8879 - val_accuracy: 0.5280 - val_loss: 1.5239
Epoch 33/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4044 - loss: 1.8891 - val_accuracy: 0.5430 - val_loss: 1.5025
Epoch 34/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4051 - loss: 1.8680 - val_accuracy: 0.5370 - val_loss: 1.5024
Epoch 35/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4019 - loss: 1.8912 - val_accuracy: 0.5250 - val_loss: 1.4957
Epoch 36/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.4080 - loss: 1.8723 - val_accuracy: 0.5350 - val_loss: 1.4835
Epoch 37/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4075 - loss: 1.8783 - val_accuracy: 0.5310 - val_loss: 1.4963
Epoch 38/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4143 - loss: 1.8525 - val_accuracy: 0.5360 - val_loss: 1.4764
Epoch 39/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4142 - loss: 1.8622 - val_accuracy: 0.5430 - val_loss: 1.4654
Epoch 40/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4163 - loss: 1.8589 - val_accuracy: 0.5320 - val_loss: 1.4784
Epoch 41/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.4164 - loss: 1.8481 - val_accuracy: 0.5390 - val_loss: 1.4772
Epoch 42/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.4132 - loss: 1.8464 - val_accuracy: 0.5570 - val_loss: 1.4517
Epoch 43/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4213 - loss: 1.8432 - val_accuracy: 0.5570 - val_loss: 1.4550
Epoch 44/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4242 - loss: 1.8436 - val_accuracy: 0.5470 - val_loss: 1.4648
Epoch 45/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4252 - loss: 1.8384 - val_accuracy: 0.5540 - val_loss: 1.4562
Epoch 46/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4364 - loss: 1.8115 - val_accuracy: 0.5530 - val_loss: 1.4542
Epoch 47/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4189 - loss: 1.8417 - val_accuracy: 0.5630 - val_loss: 1.4220
Epoch 48/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.4226 - loss: 1.8132 - val_accuracy: 0.5460 - val_loss: 1.4482
Epoch 49/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.4224 - loss: 1.8388 - val_accuracy: 0.5570 - val_loss: 1.4315
Epoch 50/50
265/265 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4316 - loss: 1.8190 - val_accuracy: 0.5720 - val_loss: 1.4126
Accuracy: 0.5745
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_optimizer.py:33: UserWarning:

Argument `decay` is no longer supported and will be ignored.
```

Model: "sequential_34"

Layer (type)	Output Shape	Param #
dense_196 (Dense)	(None, 32)	352
batch_normalization_162 (BatchNormalization)	(None, 32)	128
dropout_158 (Dropout)	(None, 32)	0
dense_197 (Dense)	(None, 64)	2,112
batch_normalization_163 (BatchNormalization)	(None, 64)	256
dropout_159 (Dropout)	(None, 64)	0
dense_198 (Dense)	(None, 128)	8,320
batch_normalization_164 (BatchNormalization)	(None, 128)	512
dropout_160 (Dropout)	(None, 128)	0
dense_199 (Dense)	(None, 64)	8,256
batch_normalization_165 (BatchNormalization)	(None, 64)	256
dropout_161 (Dropout)	(None, 64)	0
dense_200 (Dense)	(None, 32)	2,080
batch_normalization_166 (BatchNormalization)	(None, 32)	128
dense_201 (Dense)	(None, 16)	528
batch_normalization_167 (BatchNormalization)	(None, 16)	64
dropout_162 (Dropout)	(None, 16)	0
dense_202 (Dense)	(None, 8)	136
batch_normalization_168 (BatchNormalization)	(None, 8)	32
dropout_163 (Dropout)	(None, 8)	0
dense_203 (Dense)	(None, 16)	144
batch_normalization_169 (BatchNormalization)	(None, 16)	64
dropout_164 (Dropout)	(None, 16)	0

dense_204 (Dense)	(None, 32)	544
batch_normalization_170 (BatchNormalization)	(None, 32)	128
dropout_165 (Dropout)	(None, 32)	0
dense_205 (Dense)	(None, 20)	660

Total params: 24,700 (96.48 KB)

Trainable params: 23,916 (93.42 KB)

Non-trainable params: 784 (3.06 KB)

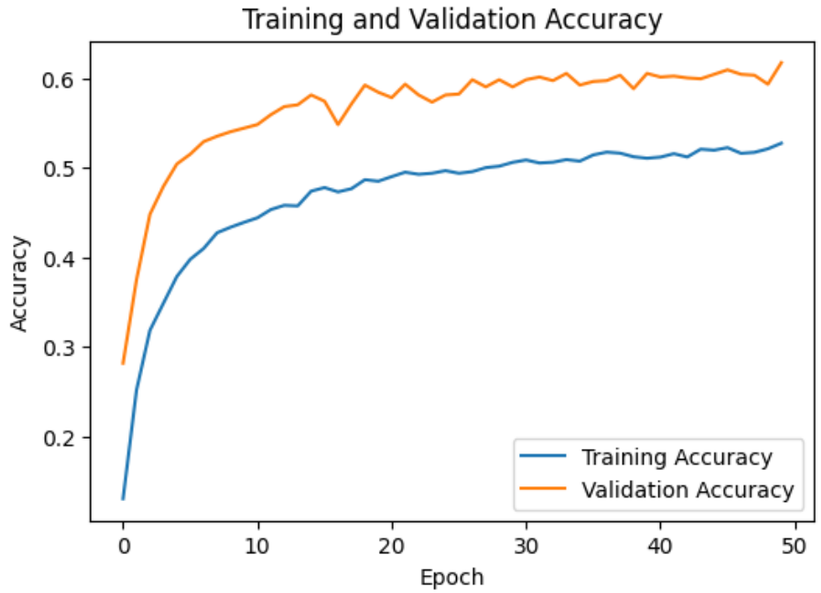
```
None
Model compiled.
Epoch 1/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 11s 8ms/step - accuracy: 0.2013 - loss: 2.7465 - val_accuracy: 0.3530 - val_loss: 2.0509
Epoch 2/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.3146 - loss: 2.1717 - val_accuracy: 0.4330 - val_loss: 1.8810
Epoch 3/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3406 - loss: 2.1291 - val_accuracy: 0.4170 - val_loss: 1.9342
Epoch 4/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3554 - loss: 2.0884 - val_accuracy: 0.4490 - val_loss: 1.8282
Epoch 5/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.3755 - loss: 2.0469 - val_accuracy: 0.4710 - val_loss: 1.7539
Epoch 6/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3789 - loss: 2.0443 - val_accuracy: 0.4730 - val_loss: 1.8405
Epoch 7/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.3925 - loss: 2.0220 - val_accuracy: 0.4470 - val_loss: 1.7660
Epoch 8/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.3989 - loss: 1.9941 - val_accuracy: 0.4860 - val_loss: 1.6912
Epoch 9/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4085 - loss: 2.0020 - val_accuracy: 0.4770 - val_loss: 1.7349
Epoch 10/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4041 - loss: 1.9768 - val_accuracy: 0.4760 - val_loss: 1.7736
Epoch 11/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.3955 - loss: 1.9996 - val_accuracy: 0.4770 - val_loss: 1.7291
Epoch 12/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4072 - loss: 1.9773 - val_accuracy: 0.4810 - val_loss: 1.6779
Epoch 13/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4136 - loss: 1.9704 - val_accuracy: 0.4570 - val_loss: 1.8247
Epoch 14/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4198 - loss: 1.9434 - val_accuracy: 0.4970 - val_loss: 1.6551
Epoch 15/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.4256 - loss: 1.9246 - val_accuracy: 0.4960 - val_loss: 1.6963
Epoch 16/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4265 - loss: 1.9419 - val_accuracy: 0.4790 - val_loss: 1.7155
Epoch 17/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4255 - loss: 1.9272 - val_accuracy: 0.5010 - val_loss: 1.6808
Epoch 18/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.4266 - loss: 1.9325 - val_accuracy: 0.4760 - val_loss: 1.7542
Epoch 19/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4249 - loss: 1.9322 - val_accuracy: 0.5340 - val_loss: 1.6195
Epoch 20/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4258 - loss: 1.9200 - val_accuracy: 0.5170 - val_loss: 1.6224
Epoch 21/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.4329 - loss: 1.9096 - val_accuracy: 0.5370 - val_loss: 1.6397
Epoch 22/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4221 - loss: 1.9333 - val_accuracy: 0.5090 - val_loss: 1.6798
Epoch 23/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.4416 - loss: 1.8949 - val_accuracy: 0.5190 - val_loss: 1.6842
Epoch 24/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4385 - loss: 1.9019 - val_accuracy: 0.5180 - val_loss: 1.6456
Epoch 25/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4392 - loss: 1.8919 - val_accuracy: 0.4990 - val_loss: 1.6642
Epoch 26/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4393 - loss: 1.8759 - val_accuracy: 0.5120 - val_loss: 1.6275
Epoch 27/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.4398 - loss: 1.9010 - val_accuracy: 0.5570 - val_loss: 1.5594
```

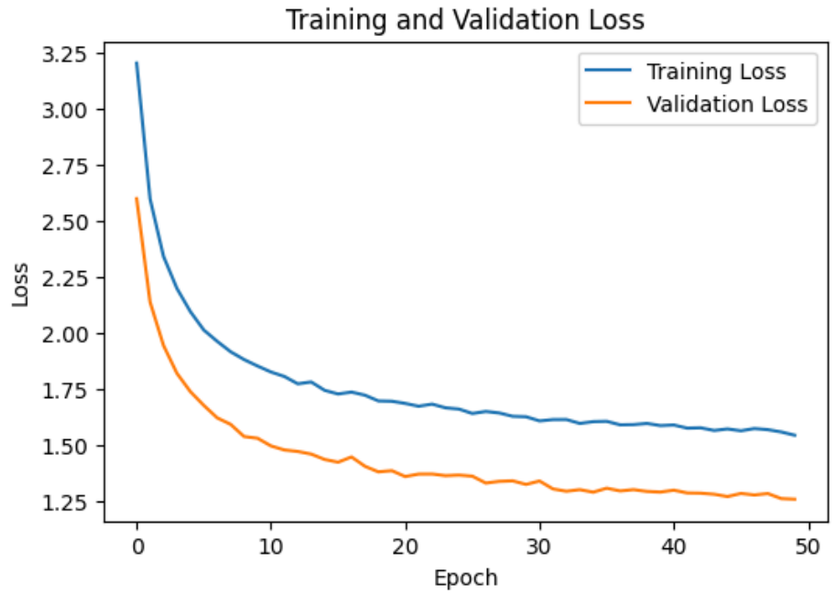
```
Epoch 28/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4543 - loss: 1.8880 - val_accuracy: 0.5360 - val_loss: 1.6269
Epoch 29/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4463 - loss: 1.8967 - val_accuracy: 0.4780 - val_loss: 1.7298
Epoch 30/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.4465 - loss: 1.8751 - val_accuracy: 0.5040 - val_loss: 1.6558
Epoch 31/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4419 - loss: 1.8875 - val_accuracy: 0.5060 - val_loss: 1.6449
Epoch 32/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4489 - loss: 1.8682 - val_accuracy: 0.5390 - val_loss: 1.5921
Epoch 33/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.4487 - loss: 1.8873 - val_accuracy: 0.5590 - val_loss: 1.5362
Epoch 34/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4517 - loss: 1.8568 - val_accuracy: 0.5060 - val_loss: 1.6511
Epoch 35/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.4386 - loss: 1.8941 - val_accuracy: 0.5300 - val_loss: 1.6404
Epoch 36/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.4472 - loss: 1.8844 - val_accuracy: 0.5310 - val_loss: 1.5783
Epoch 37/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4531 - loss: 1.8589 - val_accuracy: 0.5570 - val_loss: 1.5429
Epoch 38/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4639 - loss: 1.8632 - val_accuracy: 0.5190 - val_loss: 1.6559
Epoch 39/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.4491 - loss: 1.8719 - val_accuracy: 0.5510 - val_loss: 1.5815
Epoch 40/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.4463 - loss: 1.8931 - val_accuracy: 0.5110 - val_loss: 1.6564
Epoch 41/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.4532 - loss: 1.8581 - val_accuracy: 0.5410 - val_loss: 1.5864
Epoch 42/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.4482 - loss: 1.8718 - val_accuracy: 0.5370 - val_loss: 1.6099
Epoch 43/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.4605 - loss: 1.8644 - val_accuracy: 0.5270 - val_loss: 1.6306
Epoch 44/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.4669 - loss: 1.8444 - val_accuracy: 0.5390 - val_loss: 1.5459
Epoch 45/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4611 - loss: 1.8574 - val_accuracy: 0.5340 - val_loss: 1.5567
Epoch 46/50
530/530 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.4531 - loss: 1.8693 - val_accuracy: 0.5520 - val_loss: 1.5446
Epoch 47/50
```

In [ ]:

```
plt.figure(figsize=(6, 4))
plt.plot(best_history.history['accuracy'], label='Training Accuracy')
plt.plot(best_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.figure(figsize=(6, 4))
plt.plot(best_history.history['loss'], label='Training Loss')
plt.plot(best_history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()
```





In [ ]:
```
best_model.evaluate(X_test, y_test)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5909 - loss: 1.3261
```

Out[ ]: [1.3242542743682861, 0.5914999842643738]