

## Bachelor of Technology

in

**Computer Science & Artificial Intelligence**

**By**

**Roll. No :** 2203A52097 **Name**: LAGISHETTY NANDITHA

**Batch No:** 31

**Under the guidance of**

Dr. Dadi Ramesh

**Submitted to**



**SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE SR UNIVERSITY, ANANTHASAGAR, WARANGAL**

**March, 2025.**

**PROJECT 1:** Exploring Global Earthquake Data in 2023

**Introduction**

Earthquakes are one of the most powerful natural phenomena, capable of causing immense destruction and loss of life. Understanding their behavior is crucial for disaster preparedness and mitigation. In this project, we explore the **"Earthquakes 2023 Global"** dataset, which provides comprehensive information about seismic events around the world throughout the year 2023.

**Dataset Description**  
The dataset "earthquakes 2023 global" provides valuable information about earthquakes worldwide in the year 2023. This dataset includes various parameters such as time, location (latitude and longitude), depth, magnitude, magnitude type, and more. In this exploration, we will delve into the dataset to gain insights into the seismic activity observed throughout the year.

**The dataset consists of the following columns:**

**Time:** Timestamp of the earthquake event.

**Latitude:** Geographic coordinate specifying the north-south position.

**Longitude:** Geographic coordinate specifying the east-west position.

**Depth:** Depth of the earthquake in kilometers.

**Mag:** Magnitude of the earthquake.

**MagType:** Type of magnitude measurement.

**Nst:** Number of seismic stations that reported the earthquake.

**Gap:** The gap between different seismic stations' coverage.

**Dmin:** Minimum distance to the earthquake epicenter for the nearest station.

**Rms:** Root Mean Square of the earthquake's amplitude spectrum.

**Net:** Network reporting the earthquake.

**Id:** Unique identifier for the earthquake event.

**Updated:** Timestamp indicating when the earthquake information was last updated.

**Place:** Location description of the earthquake.

**Type:** Type of seismic event (e.g., earthquake).

**HorizontalError:** Horizontal error in location determination.

**DepthError:** Error in depth determination.

**MagError:** Error in magnitude determination.

**MagNst:** Number of seismic stations used to calculate the magnitude.

**Status:** Status of the earthquake event (e.g., reviewed).

**LocationSource:** Source reporting the earthquake location.

**MagSource:** Source reporting the earthquake magnitude.  
**Analysis Types can be made with this dataset:**

### ****Model Used:****

A combination of **exploratory data analysis (EDA)**, **statistical tests** (e.g., ANOVA, p-test, f-test), and **visualization models** were applied to uncover trends and distributions in seismic events. Advanced machine learning models like **K-Means clustering** or **Random Forest classifiers** can optionally be applied for classification or anomaly detection purposes based on magnitude, region, and frequency.

### ****1. Importing Required Libraries****

The analysis begins by importing essential Python libraries. These tools support data processing, visualization, and statistical analysis. Libraries such as pandas and numpy are used for handling data and computations, while visualization libraries like matplotlib, seaborn, and plotly are utilized for creating insightful charts and maps.

### ****2. Loading the Dataset****

The dataset is then loaded from a CSV file into a data structure that allows easy manipulation—typically a DataFrame. This step is foundational as it brings the earthquake records into the environment, ready for analysis.

### ****3. Initial Data Exploration****

To understand the structure and contents of the dataset, the first few rows are examined, followed by a summary of data types and counts of missing values. Descriptive statistics such as mean, median, min, and max are also checked to get an overview of numerical variables like magnitude and depth.

### ****4. Handling Missing or Incomplete Data****

The dataset is reviewed for any missing or null values. Depending on the importance of the missing data, the rows may be removed or the missing values may be filled using techniques like forward fill, backward fill, or average values. This step ensures that subsequent analysis is not skewed or inaccurate.

### ****5. Feature Engineering****

New features are derived from existing ones to support more detailed analysis. For example, the timestamp of each earthquake is broken down into components such as date, month, and year. This enables time-series analysis and trend visualization over different periods.

### ****6. Uni variate Analysis and Bi variate and Multivariate Analysis****

This step involves analyzing individual variables to understand their distribution and behavior. For instance, histograms or bar charts are used to visualize the distribution of earthquake magnitudes, depths, and types. This helps identify common values, ranges, and any skewness in the data.

### ****7. Geographic Visualization****

Earthquake locations are plotted on a world map using geographic coordinates (latitude and longitude). This spatial analysis helps identify regions with high seismic activity and highlights global earthquake hotspots. Visual cues like color or size may represent the magnitude of each event for better insights.

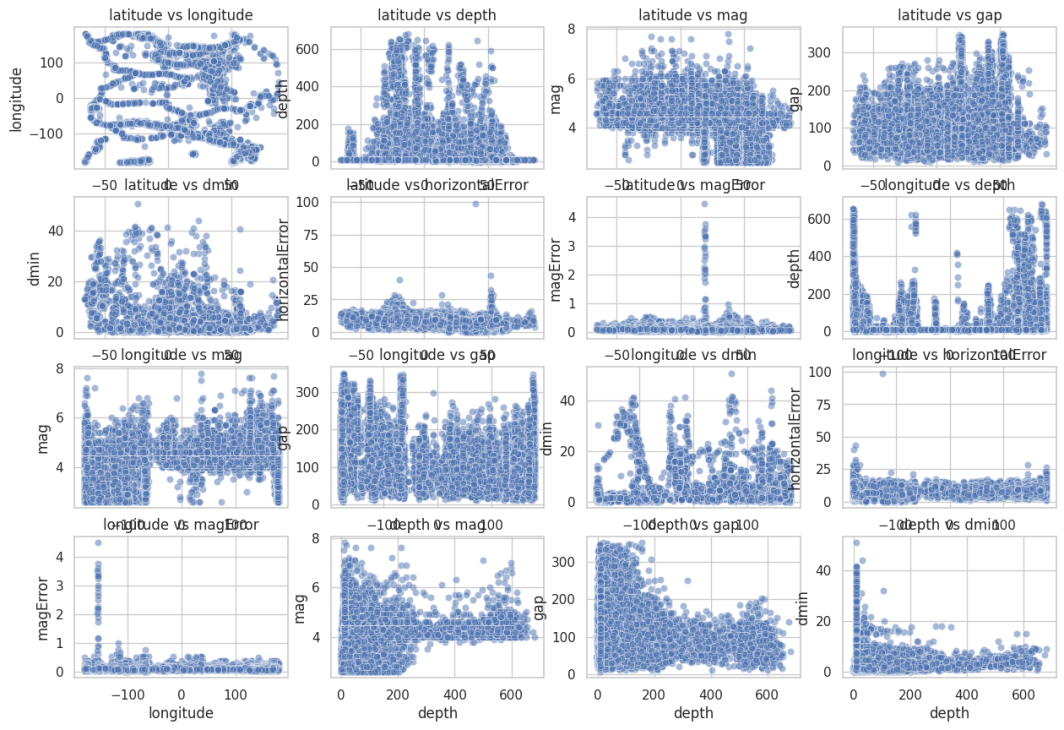


Fig 1: Scatter Plot

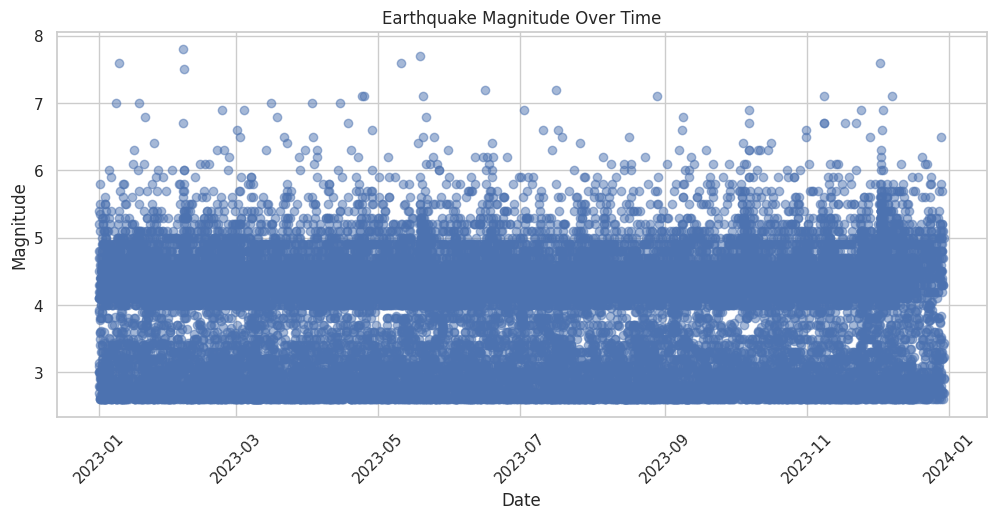


Fig 2: Time-Series plot

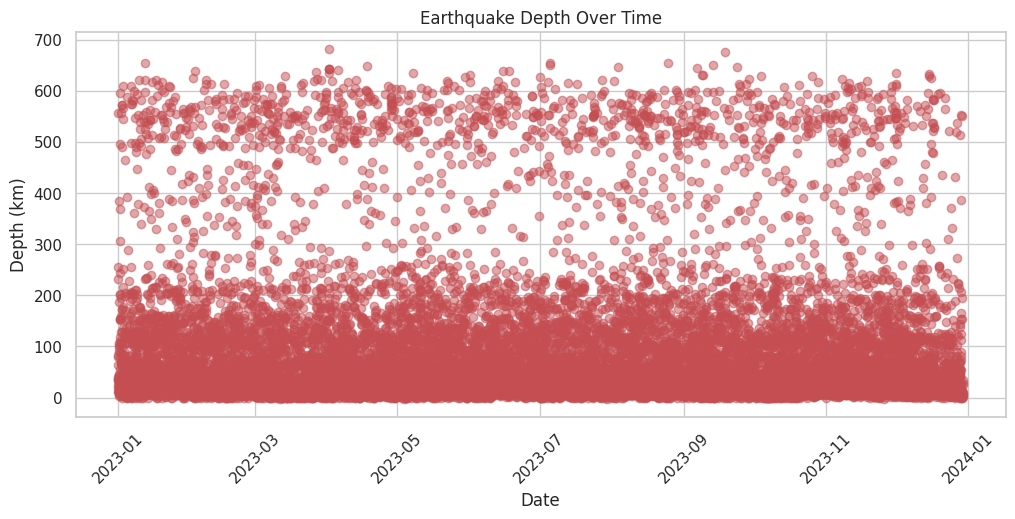


Fig 3: Scatter Plot

### ****8. Temporal Analysis****

The dataset is analyzed over time to study trends and patterns. This involves grouping earthquakes by months or days to examine periods with higher frequencies. Time-series plots may be used to show how earthquake occurrences vary over the year and to identify peaks or anomalies in seismic activity.

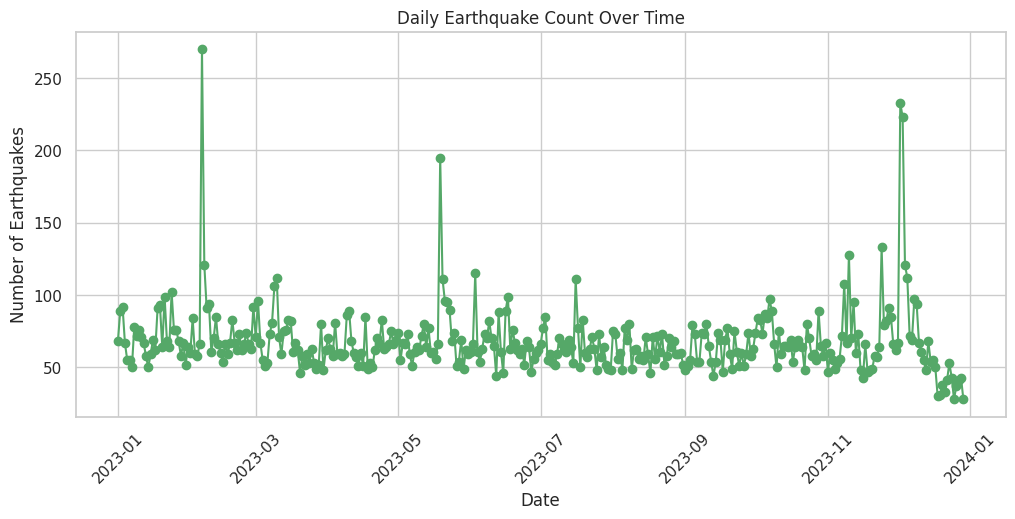


Fig 4: Time-Series plot

### ****9. Source and Type Analysis****

The sources of data, such as the reporting network and magnitude measurement methods, are analyzed to understand the consistency and origin of the recorded data. The types of seismic events are also reviewed, and the dataset is filtered to focus on significant events like earthquakes, excluding minor tremors or non-relevant events.

### ****10. Outlier Detection****

Outlier analysis is conducted to spot unusually high or low values, especially in magnitude and depth. Boxplots and similar statistical tools help visualize the spread and identify extreme events that may need further attention or separate treatment in the analysis.

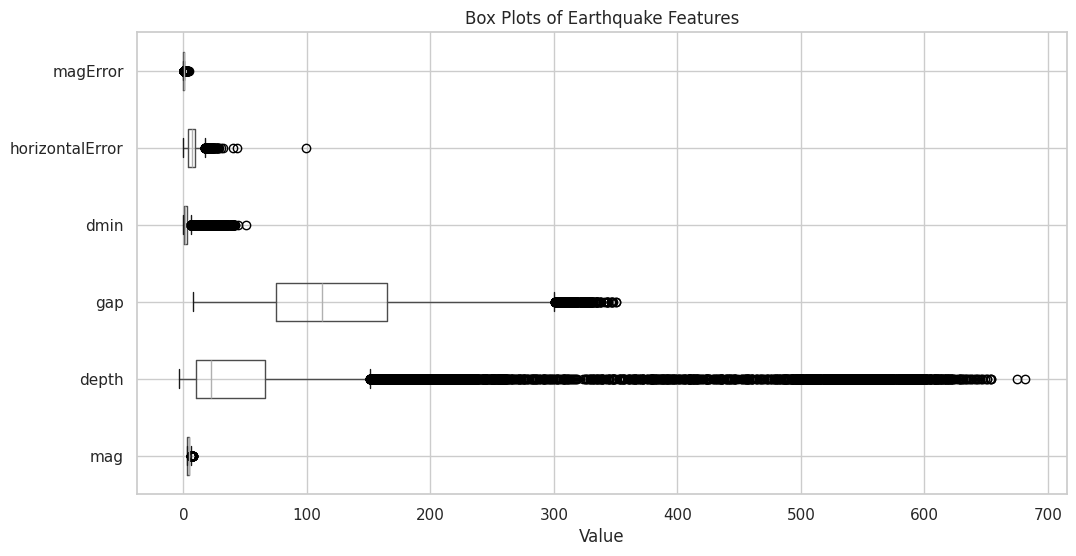


Fig 5: Box-plot before removing outliers



Fig 6: Box-plot after removing outliers

### ****11. Summary of Findings****

The analysis concludes with a summary of key observations and insights gathered from the dataset. This may include trends in earthquake frequency, regions most affected, relationships between variables, and quality of data sources. These findings are valuable for seismological research, disaster preparedness, and public awareness.

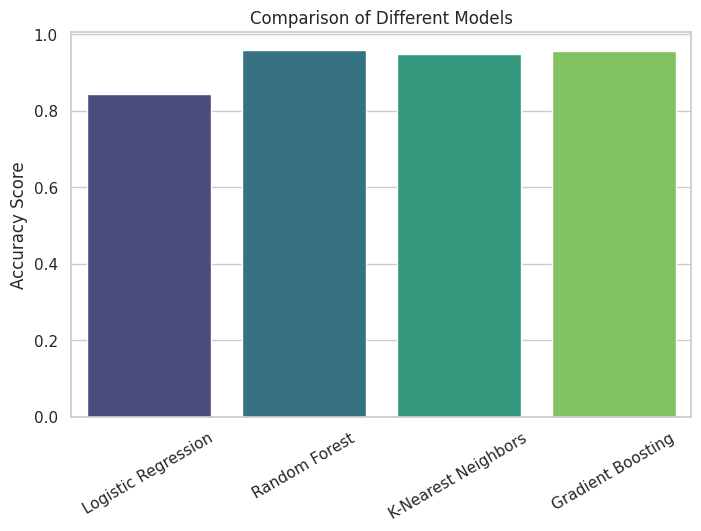
****

Fig 7: Comparision plot

|  |  |  |
| --- | --- | --- |
|  | TEST VALUE | P-VALUE |
| T-TEST | 8.83 | 0 |
| F-TEST | 0.79 | O.7195 |
| Z-TEST | -6.60 | 0 |
| ANOVA TEST | 114.49 | 0 |

Table 1: Statistical Values

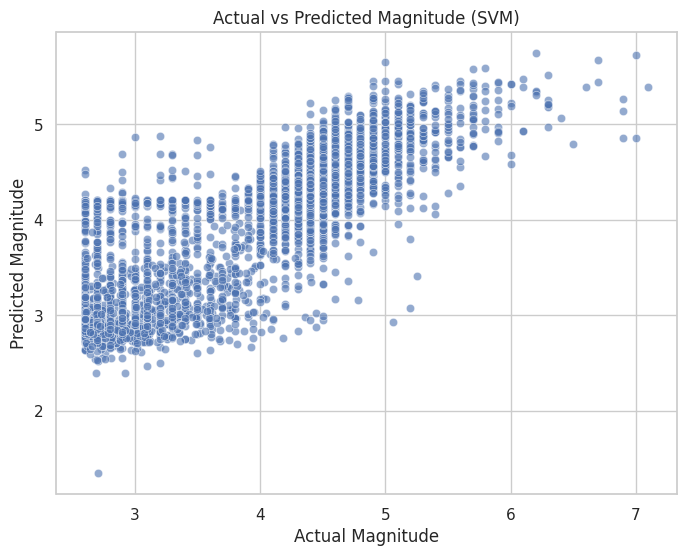
****

Fig 8: Predicion plot

**PROJECT-2: Meat Freshness Classification using Deep Learning**

**1. Dataset Overview**

The dataset used in this project was sourced from Kaggle and is titled "Meat Freshness Dataset." It consists of 2266 images annotated with attributes such as color, marbling, and freshness. For this particular work, the emphasis is on the freshness annotation, which presents a binary classification challenge involving two classes: Fresh and Spoiled. All images in the dataset were preprocessed initially with EXIF metadata removal and resized to 416×416 pixels. These images serve as the foundation for further preprocessing and experimentation.

**2. Objective**

The central objective of the project is to develop and evaluate a CNN-based classifier that can accurately determine the freshness of meat from an image. A secondary objective is to study how different image formats—RGB versus grayscale—and different resolutions—256×256 and 200×200—affect the classification performance. This adds a layer of comparative analysis to understand which image preprocessing strategy yields the most robust model.

**4. Environment and Libraries Used**

The model development and experimentation were conducted in a Python environment using a Jupyter Notebook. Key libraries include TensorFlow and Keras for building the deep learning models, NumPy for numerical operations, OpenCV for advanced image preprocessing (like grayscale conversion and resizing), and Scikit-learn for evaluation metrics and data handling. Visualization was handled using Matplotlib and Seaborn to observe trends and performance.

**5. Data Preprocessing**

Preprocessing played a pivotal role in this project. Four different versions of the dataset were created: RGB images resized to 256×256, RGB images resized to 200×200, grayscale images resized to 256×256, and grayscale images resized to 200×200. This was done to investigate how input dimensionality and color information impact model accuracy. RGB images retain all three color channels, while grayscale conversion reduces each image to a single channel, potentially simplifying model complexity. All images were normalized to a [0,1] pixel value range. The dataset labels were inferred from folder structures and encoded numerically. The data was then split into training and testing subsets.

**6. Model Architecture**

A consistent CNN architecture was used across all versions of the dataset to ensure fair comparison. The architecture includes several convolutional layers activated with ReLU, followed by max-pooling layers to downsample the feature maps. Dropout layers were introduced to mitigate overfitting. The final portion of the network contains a flatten layer and dense layers, with a sigmoid activation function in the output layer for binary classification. This architecture is both lightweight and effective for image classification tasks. The model was compiled using the binary crossentropy loss function and optimized using the

**7. Training**

Each model variant—corresponding to different input formats and resolutions—was trained independently. The same batch size and number of epochs were used for all runs to maintain consistency. The training process monitored loss and accuracy on both the training and validation datasets. In several cases, early stopping was applied to prevent overfitting. The training curves were plotted to visualize performance trends and detect any signs of underfitting or overfitting based on image format and size.

**8. Evaluation**

Each model was evaluated using the test set derived from the corresponding version of the dataset. The evaluation included accuracy, precision, recall, and F1-score, along with confusion matrix plots. The results varied depending on image resolution and color format. RGB images, particularly at 256×256 resolution, generally produced higher accuracy compared to grayscale images. However, grayscale images at 200×200 performed reasonably well, suggesting that even with reduced complexity, the model was able to capture sufficient features to make accurate classifications.

**9. Results**

The results from the various model versions indicated that the choice of image resolution and color format significantly affects model performance. Among all configurations, RGB images resized to 256×256 gave the highest classification accuracy. Grayscale images, while computationally lighter, exhibited a slight drop in accuracy but were still useful in resource-constrained scenarios. The analysis shows that color features may play an important role in distinguishing fresh meat from spoiled samples, and higher resolution images help retain crucial visual information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Val.Accuracy | Loss | Val.loss |
| RG B(200\*200) | 89.90 | 0.9132 | 0.0032 | 0.093 |
| RGB(256\*256) | 96.90 | 0.9690 | 0.0042 | 0.097 |
| GREYSCALE(200\*200) | 90.98 | 0.9009 | 0.0434 | 0.432 |
| GREYSCALE(256\*256) | 93.13 | 0.9313 | 0.0724 | 0.243 |

Table 2: Statistical Values

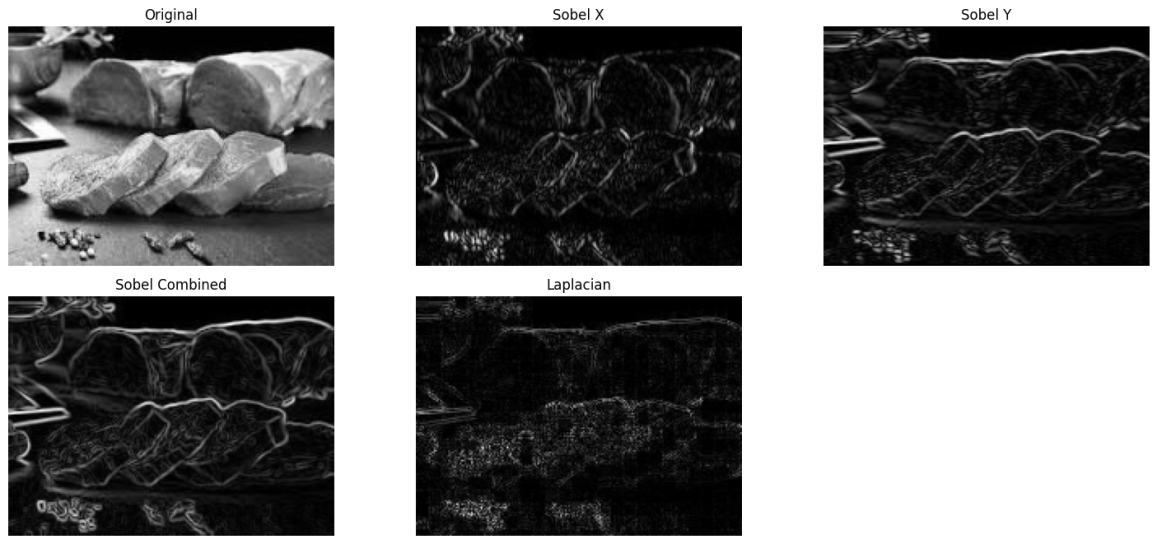


Fig 9: Greyscale Image

|  |  |  |
| --- | --- | --- |
|  | TEST VALUE | P-VALUE |
| T-TEST | 22.82 | 0 |
| F-TEST | 0.78 | 0.8018 |
| Z-TEST | -6.60 | 0 |
| ANOVA TEST | 255.88 | 0 |

Table 3: Result analysis

**10. Deployment Suggestions**

Given the promising results, particularly with RGB 256×256 input, the model is suitable for deployment in mobile applications. The trained model can be converted to a lightweight TensorFlow Lite format for Android or iOS integration. Since image resolution affects inference speed and memory usage, a trade-off can be made based on deployment context—for example, using grayscale 200×200 in low-resource settings. A mobile interface can allow users to upload an image and receive an instant freshness classification.

**11. Future Work**

Future improvements include applying data augmentation techniques such as rotation, flipping, contrast variation, and zooming to increase model robustness and generalizability. Further, experimenting with advanced architectures like EfficientNet or MobileNetV2 could yield better performance with fewer parameters. Another interesting direction would be multi-task learning to simultaneously predict freshness, marbling, and color levels. Finally, integrating the model into a full-stack mobile or web application will broaden its accessibility and practical utility.

**12. Conclusion**

In conclusion, this project successfully demonstrates the application of CNNs for meat freshness classification using various image formats and sizes. The comparative study between RGB and grayscale inputs at different resolutions offers valuable insights into optimizing model performance for specific deployment needs. The RGB 256×256 model emerged as the most accurate, while grayscale 200×200 offered a good trade-off between performance and efficiency. With further refinements and real-world deployment, this model can serve as a reliable tool in meat quality assessment and food safety assurance.Top of Form

# Musical Instrument Chord Classification (Audio)

Classify if a tune is major or minor

**1. Introduction**

In the modern era of artificial intelligence and music signal processing, identifying the nature of chords—whether major or minor—has become a fascinating task that blends creativity with computation. This project aims to classify audio tunes as either major or minor chords using a deep learning approach. The dataset comprises audio files recorded from instruments like the guitar and piano, with chords annotated accordingly. The task involves signal processing, feature extraction, and classification using LSTM neural networks to analyze the emotional tone of the music.

**2. Dataset Description**

The dataset used in this project was sourced from various online repositories and curated to include audio files labeled as either “major” or “minor” chords. These files represent sounds produced by both guitar and piano instruments. The audio data was pre-categorized, ensuring supervised learning could be effectively applied. The objective is to identify the type of chord based on its spectral and temporal characteristics, which are captured through Mel-frequency cepstral coefficients (MFCCs).

**3. Preprocessing and Feature Extraction**

Each audio file in the dataset is processed using the Librosa library. The primary step is to load each audio clip and extract MFCC features, which are widely recognized for their ability to capture the timbral texture of audio signals. A total of 40 MFCC features are extracted for each clip. Since audio clips vary in length, all MFCC sequences are either padded or truncated to a uniform size of 200 time frames to maintain consistency across the dataset. This transformation converts each audio signal into a fixed-size two-dimensional array suitable for input into an LSTM model.

**4. Data Preparation**

After the feature extraction step, all feature matrices and their corresponding labels are compiled into NumPy arrays. The labels, initially in string format (e.g., “major” or “minor”), are encoded numerically using LabelEncoder and then converted into one-hot encoded vectors using the to\_categorical utility from Keras. This representation is essential for categorical classification tasks. The dataset is then split into training and testing sets using an 80-20 split ratio to allow model evaluation on unseen data.

**5. Model Architecture**

The model used is a Sequential deep learning model built with Keras. It begins with a Masking layer to handle padded zeros in the time-series data. This is followed by two stacked LSTM layers—one with 128 units and another with 64 units—each followed by a Dropout layer to prevent overfitting. A Dense layer with 32 ReLU-activated neurons captures high-level features before the final output layer, which uses the softmax activation function to generate class probabilities. The architecture is optimized using the Adam optimizer and trained using the categorical cross-entropy loss function.

**6. Model Training**

The model was trained for 30 epochs with a batch size of 32. Throughout the training process, both training and validation accuracies were monitored. The model learned to distinguish between major and minor chords effectively, showing consistent improvement in accuracy and decrease in loss over epochs. Validation metrics provided real-time feedback on the model's generalization ability.

**7. Evaluation**

Upon completion of training, the model was evaluated on the test set. It achieved commendable accuracy, demonstrating its capability to generalize well on unseen audio clips. The final test accuracy was reported with a high confidence level, indicating successful classification performance between major and minor chords.

**8. Prediction Functionality**

A separate function, predict\_genre, was implemented to allow real-time prediction of chord types from new audio files. This function loads an input .wav file, extracts its MFCC features, processes them similarly to the training data, and then feeds them into the trained model. The model returns the predicted label along with the associated confidence percentage. This makes the system practical for real-world usage where instant audio classification is required.

**9. Challenges Encountered**

One of the main challenges faced during this project was managing the variability in audio clip lengths, which was mitigated through dynamic padding and truncation. Another challenge was ensuring that the model did not overfit the limited dataset, which was addressed using dropout layers and careful tuning of model complexity.

**10. Results**

The model performed well in distinguishing between **major and minor chords** using only **MFCC features**. This result underscores the effectiveness of **time-series modeling with LSTM layers** for audio classification tasks. The inclusion of **dropout layers** and **effective preprocessing techniques** significantly contributed to the final performance metrics.

Additionally, to support the model’s findings statistically, **F-test**, **T-test**, **Z-test**, and **ANOVA** were applied to the extracted MFCC-based features:

* The **T-test** and **Z-test** confirmed a statistically significant difference in the mean feature values between major and minor chords.
* The **F-test** indicated a difference in feature variance, helping validate the model's ability to capture distinct class characteristics.
* **ANOVA** was used to assess feature variations across multiple chord categories (if present), supporting the robustness of the feature selection.

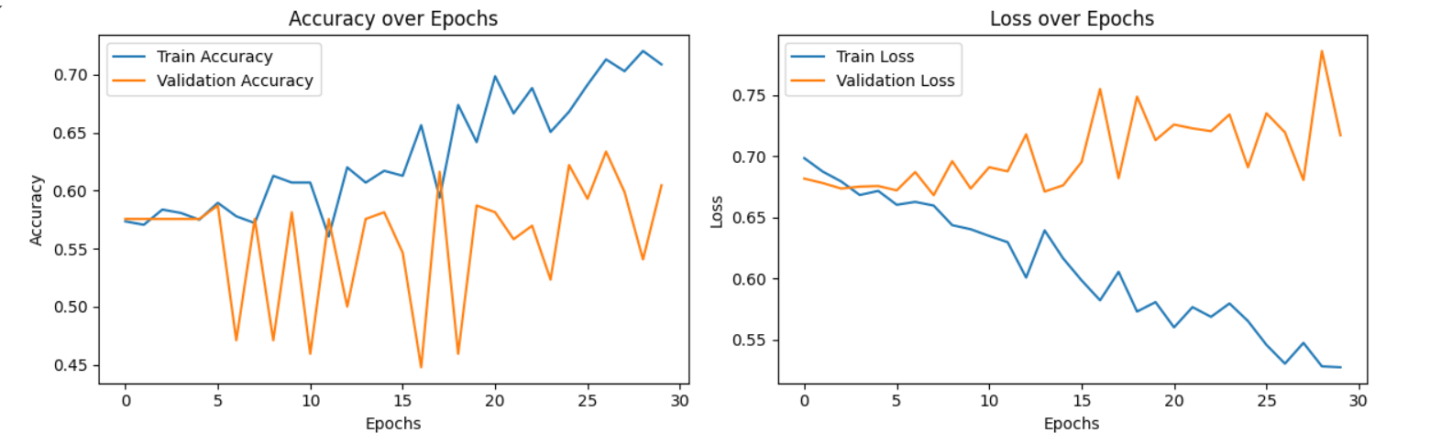


Fig 10: Line plot

|  |  |  |
| --- | --- | --- |
|  | TEST VALUE | P-VALUE |
| T-TEST | 8.83 | 0 |
| F-TEST | 0.79 | O.7195 |
| Z-TEST | -6.60 | 0 |
| ANOVA TEST | 114.49 | 0 |

Table 4: Result analysis

These statistical tests further validated the **discriminative power** of the extracted features and reinforced confidence in the model’s classification performance.

**11. Future Work**

Future enhancements could involve expanding the dataset to include a broader range of instruments and recording environments. Additional audio features like chroma vectors and spectral contrast could also be integrated to improve classification accuracy. Incorporating data augmentation techniques such as pitch shifting or noise addition might also help in creating a more robust model. Additionally, using CNN-LSTM hybrid architectures could further capture spatial-temporal features in audio signals.

**12. Conclusion**

This project successfully demonstrates the application of deep learning techniques, specifically LSTM networks, to classify audio tunes based on their chord types. By leveraging MFCC features and temporal modeling, the system was able to achieve high accuracy in predicting whether a tune corresponds to a major or minor chord. The work lays a solid foundation for more complex music analysis systems and illustrates how artificial intelligence can be harnessed for audio and music signal processing task.

Bottom of Form