Final Python Project

Image Sentiment Analysis for Happy, Sad, and Angry Using CNN Model

Ali Nobakhtnamini

202180090107

December 2024

1. Introduction

1.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be. The human brain processes a huge amount of information the second we see an image.

Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes data only in its receptive field as well. The layers are arranged in such a way that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can enable sight to computers.

1.2 System Overview

Facial expressions are one of the most fundamental ways humans convey emotions. Identifying emotions like happiness, sadness, and anger from facial expressions is a significant challenge in the field of computer vision and machine learning. This project aims to develop a robust image sentiment analysis model capable of classifying human facial expressions into three categories: Happy, Sad, and Angry. Using a dataset sourced from Kaggle, this project explores the design, implementation, and testing of a convolutional neural network (CNN) for this purpose. The dataset includes images categorized into three folders, each representing one emotion, and is compiled from various sources. The motivation for this project stems from the importance of emotion recognition in applications such as mental health assessments, human-computer interaction, and security systems, where accurate and efficient emotion detection can enhance functionality.

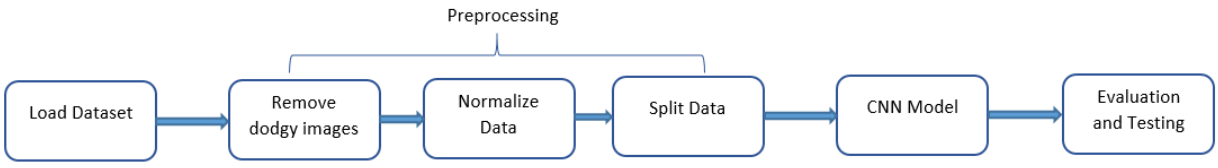

Fig 1 – System Overview Structure

2. Implementation Process

2.1 Preprocessing Steps

The implementation process began with data preparation, a critical step to ensure the dataset is clean and consistent. This involved validating image formats against a predefined list of acceptable extensions (jpeg, jpg, bmp, png) and removing corrupted files. A script was developed to iterate through all dataset directories, checking each image's format using libraries like cv2 and imghdr, and logging any issues encountered. Additionally, the directory structure was inspected to confirm alignment with TensorFlow's image_dataset_from_directory requirements, ensuring seamless data loading and preprocessing. GPU support was enabled using TensorFlow's experimental memory growth settings to optimize hardware utilization during training. The dataset was organized into a directory structure compatible with TensorFlow's image_dataset_from_directory utility, enabling efficient loading and preprocessing of images.

Images were resized to a uniform size of 128x128 pixels to ensure consistency across the dataset and reduce computational complexity. A batch size of 64 was chosen to balance memory constraints and training efficiency. Valid image formats were enforced by checking files against a predefined list of extensions (jpeg, jpg, bmp, png), and invalid or corrupted files were removed. This step was implemented using a script that iterated through the dataset, validating image formats and handling exceptions gracefully.

The dataset consisted of 2,625 images distributed across three categories: Happy, Sad, and Angry. A class frequency analysis was conducted to visualize the distribution of images in each category. Bar plots were generated to display class frequencies, providing insights into dataset balance. This step ensured that class imbalances were identified early in the process, potentially informing decisions about data augmentation or weighting during training.

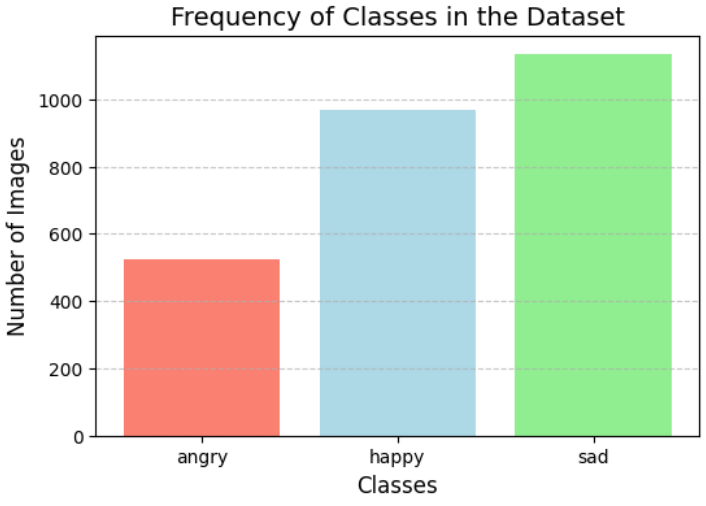

Fig 2 - Data visualization of the frequency of classes

Pixel values were normalized to the range [0, 1] to standardize input data and improve model convergence. This normalization was achieved using TensorFlow's mapping function, which divided pixel intensities by 255.0. Batch statistics were then verified, with sample image shapes and label distributions printed for confirmation. Normalized images were visualized using Matplotlib to ensure preprocessing steps preserved image integrity.

The dataset was split into training, validation, and test subsets with an 80:10:10 ratio. TensorFlow's take and skip methods facilitated this division, ensuring that each subset was appropriately partitioned. The training set, comprising 33 batches, was used for model training, while the validation set (4 batches) monitored performance during training. The test set (5 batches) evaluated the model's generalization capability on unseen data. This structured split helped maintain a robust evaluation pipeline for the model.

2.2 Model Design

The design of this project centers on building a CNN-based architecture capable of handling image data efficiently. CNNs were chosen because of their proven effectiveness in image classification tasks, leveraging convolutional layers to automatically extract important features from images. Unlike traditional machine learning methods that require manual feature extraction, CNNs learn features directly from raw image data, making them particularly suitable for complex tasks like sentiment analysis from facial images.

The input to the model is a 128x128 RGB image, resized to standardize the data and ensure compatibility with the network architecture. Resizing is essential as it reduces computational load while maintaining the integrity of the image's features. The architecture includes three convolutional layers with increasing filter sizes (32, 64, and 128) to capture progressively complex features. For instance, the initial layers focus on detecting basic patterns such as edges, while deeper layers extract more abstract features like facial structures.

Each convolutional layer is followed by a max-pooling layer to reduce spatial dimensions and mitigate overfitting by downsampling the feature maps. This step not only reduces computational complexity but also ensures the network focuses on the most critical features. The network then flattens the spatial data into a single vector, which is passed through dense layers. A dropout layer with a rate of 0.5 is included for regularization, randomly disabling neurons during training to prevent the model from relying too heavily on specific features. This design decision addresses overfitting, a common challenge in deep learning.

The final dense layer employs a softmax activation function, ensuring the output represents probabilities for each emotion class. This approach is particularly effective for multi-class classification tasks, as it allows the network to predict the likelihood of each class. For optimization, the Adam optimizer was selected due to its adaptive learning rate capabilities, ensuring faster convergence compared to traditional methods like stochastic gradient descent. The categorical cross-entropy loss function was used to measure the discrepancy between predicted and actual labels, a standard choice for multi-class classification tasks. Metrics such as

accuracy were chosen to evaluate performance during training and validation, providing an intuitive measure of the model's effectiveness.

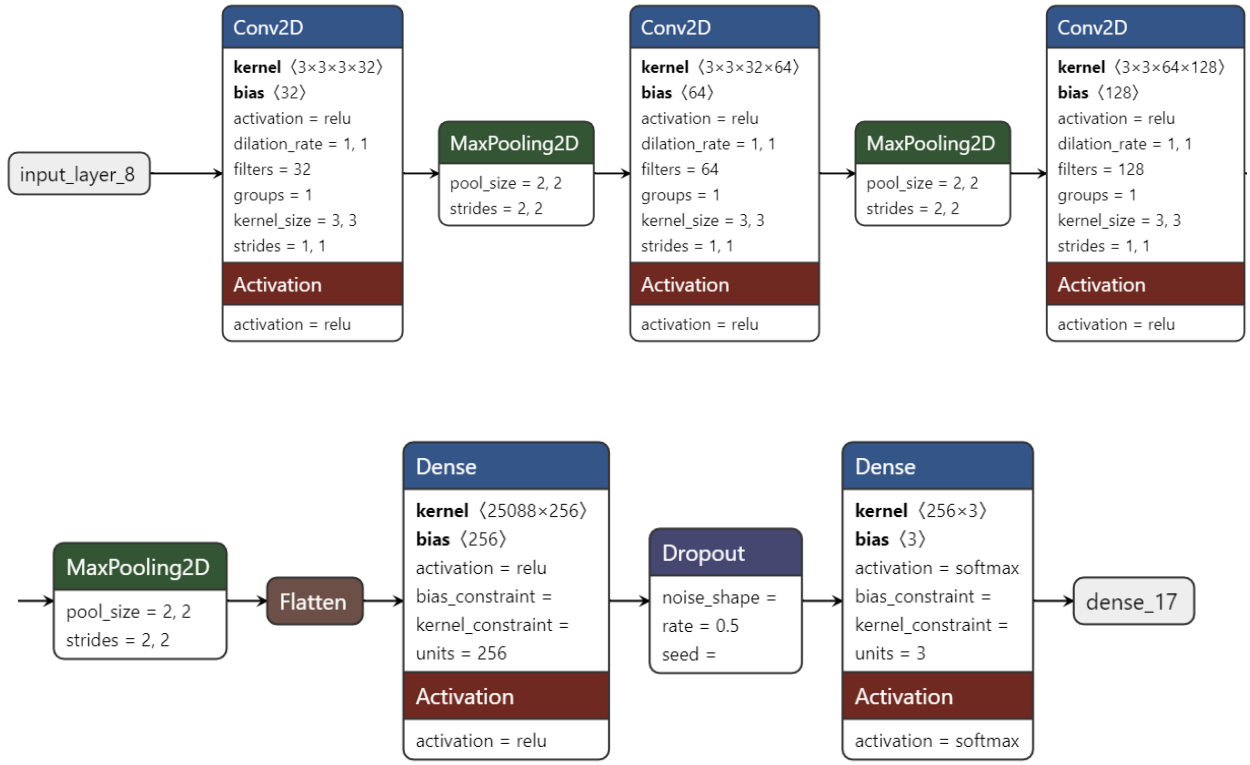

Fig 3 - CNN Architecture

2.3 Testing

The testing phase evaluated the model using the test set, which the model had not encountered during training. The test results showed a **high accuracy of over 98%**, indicating the model's effectiveness in classifying emotions. Validation metrics were also analyzed, demonstrating consistent improvement and alignment with the training metrics, confirming the model's generalization capability. Testing also involved analyzing specific cases where the model's predictions were incorrect.

Sample predictions were conducted on individual images to test the model's real-world applicability. Images were preprocessed similarly to the training data, ensuring compatibility. The model output a predicted class index, which was mapped to the corresponding emotion label (Happy, Sad, or Angry). For example, an image of an angry expression was correctly classified with high confidence, showcasing the model's robustness. These predictions highlighted the model's ability to generalize to unseen data effectively.

Fig 4 – Angry class test

Error analysis revealed a few misclassifications, particularly in cases where facial expressions were ambiguous or partially obscured. This suggests that incorporating additional data and advanced techniques like transfer learning could enhance model performance further. Visualizing the Accuracy and Loss Function provided a comprehensive view of the model's performance across all classes, highlighting areas where the model excelled and where it struggled.

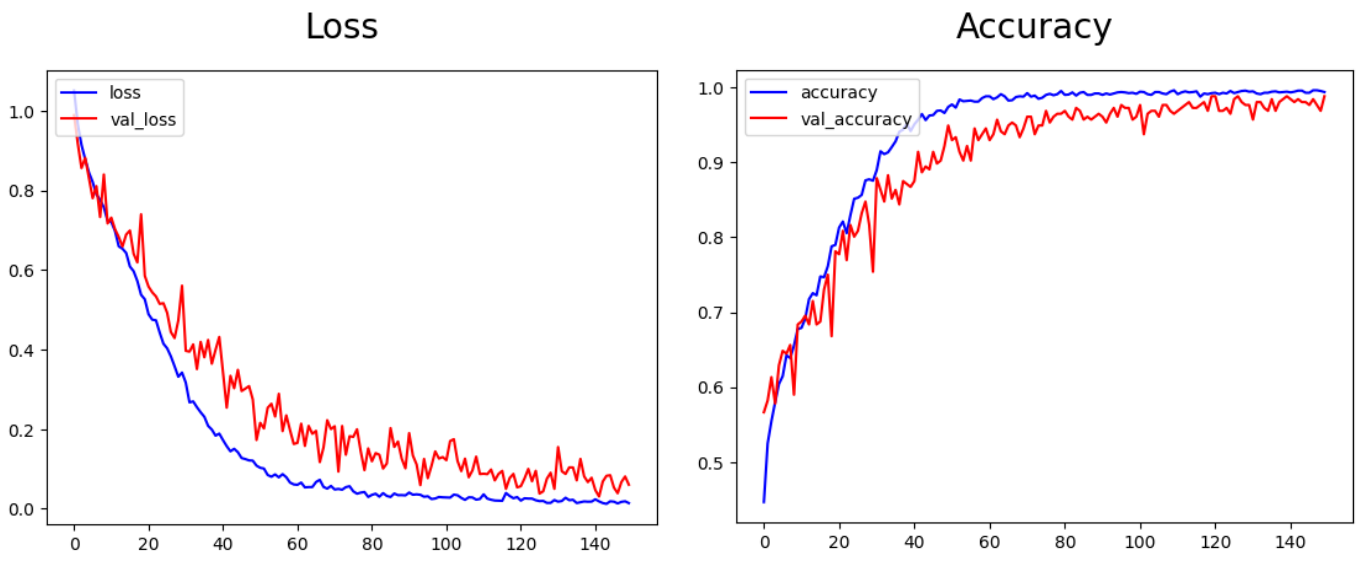

Fig 5 – Loss and Accuracy Diagram

3. Deployment

To make the model accessible, a web application was developed using Flask. Flask was chosen for its simplicity and flexibility, making it ideal for deploying machine learning models. The app allowed users to upload an image, which was then processed and classified by the model. The Flask application included routes for the homepage and a /predict endpoint for handling image uploads and predictions. The /predict endpoint was designed to accept an image file uploaded by the user via an HTTP POST request. Upon receiving the image, the backend validated its format and size to ensure compatibility. The image was then preprocessed to match the model's input specifications, including resizing it to 128x128 pixels and normalizing pixel values. After preprocessing, the image was passed through the trained model to generate predictions, and the resulting emotion label along with the confidence score was sent back as a JSON response. This approach ensured the endpoint was efficient, robust, and user-friendly. Uploaded images were validated for format and size, and errors were handled gracefully, providing informative responses to users.

The user interface was designed to be intuitive, enabling users to interact with the model seamlessly. The backend processed the image, resized it to 128x128, normalized it, and passed it through the trained model. The resulting prediction and confidence score were displayed to the user, ensuring transparency and reliability. The deployment process demonstrated the practical utility of the model, making it accessible to non-technical users through a simple web interface.

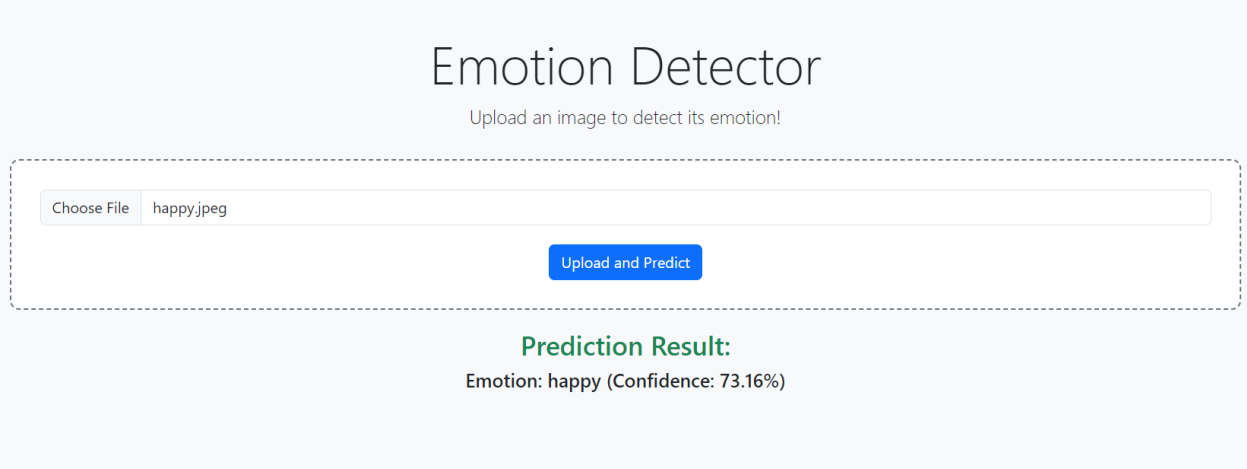

Fig 6 - Interface

4. Challenges

Several challenges were encountered during the project. Handling diverse image formats and ensuring compatibility with the model required thorough data preprocessing. Overfitting was mitigated through dropout layers and careful monitoring of validation metrics. Deploying the

model as a web application introduced challenges in ensuring efficient image handling and response times. These challenges underscored the importance of robust preprocessing pipelines, well-designed architectures, and efficient deployment strategies.

This project highlighted the importance of robust data preprocessing, a well-designed model architecture, and effective deployment strategies. It also emphasized the need for continuous evaluation and iteration to improve performance. The success of this project demonstrates the potential of deep learning in solving complex problems like emotion recognition, showing the way for further advancements in this field.

5. Conclusion and Future Work

The project successfully developed a CNN model capable of classifying facial expressions into Happy, Sad, and Angry categories with high accuracy. The deployment as a web application further demonstrated its practical utility. Future enhancements could include using a larger and more diverse dataset, implementing data augmentation, and experimenting with advanced architectures like transfer learning or pre-trained models. Deploying the model on a scalable cloud platform could also increase accessibility and efficiency. Additionally, exploring real-time emotion recognition using video data could expand the model's applications in fields such as security and interactive systems.

References

- Kaggle Dataset: https://www.kaggle.com/datasets/tonny22/dataset-of-faces-as-per-emotions