Introduction V1

Emotions are an important part of the human species. Ever since we were born, we have been guided by our feelings. Even if we only feel them internally, we usually also show them externally. This helps our fellow human beings to understand what emotional state we are in without having to engage in a verbal exchange. However, it is not always easy to read the emotions of strangers from their facial expressions. So our question is, can we train a neural network to recognize human emotions from facial expressions? Such a machine learning model would enable us to automatically record and process data on emotions. This would allow us to analyse and improve advertising, films, series or any content that aims to trigger certain emotions in viewers using statistical data.

Introduction V2

## Emotions: The Unspoken Language of Humanity

From our very first cries as newborns, emotions have served as a powerful internal compass, guiding our thoughts and actions. But emotions are not just limited to the inner realm. We naturally express them outwardly through facial expressions, offering a window into our emotional state for those around us. This nonverbal communication is fundamental to human interaction, allowing us to connect and understand each other without speaking a single word.

However, deciphering the emotions of strangers, particularly through facial expressions, can be a complex task. This is where the exciting potential of artificial intelligence emerges. Can we harness the power of neural networks to train a system that can recognize human emotions from facial expressions?

Imagine a machine learning model capable of automatically interpreting and recording emotional data. This would unlock a treasure trove of insights, allowing us to analyze and optimize content across various mediums – advertising, film, television – with the power of data-driven emotional response. By understanding how viewers react emotionally, we could create more impactful and targeted content, ensuring our message resonates on a deeper level.

This exploration into emotional recognition through AI is not just about technological advancement; it's about unlocking a new layer of understanding in human communication. It's about empowering machines to bridge the gap between the internal world of emotions and the external world of expression.

Literature review:

Smote:

Literature review: SMOTE for imbalanced classification with application to emotion recognition

**Introduction**

Class imbalance is a major challenge in machine learning classification tasks where one or more classes have far fewer data points compared to the majority class. This can lead to biased models that favor the majority class and perform poorly on the minority class. The Synthetic Minority Oversampling Technique (SMOTE) is a widely used technique that corrects the imbalance between classes by oversampling the minority class, creating a more balanced data set for training.

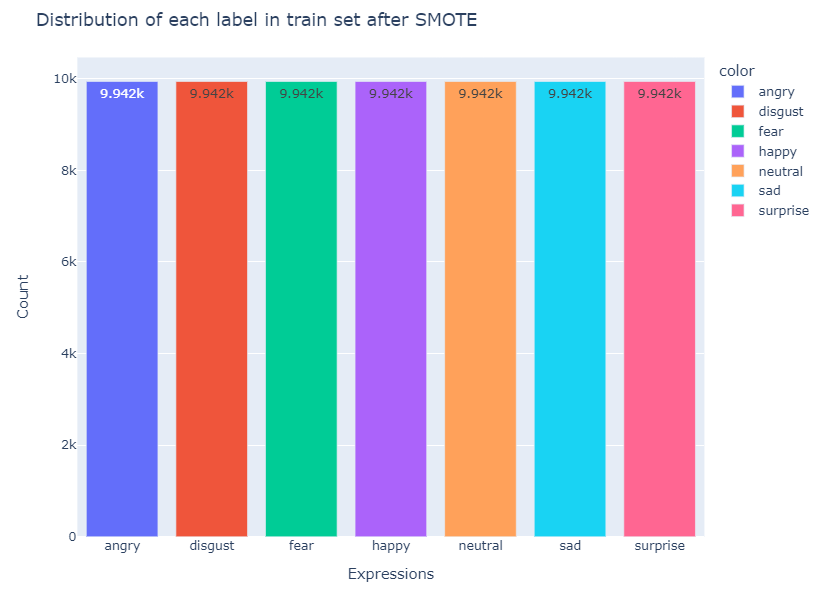
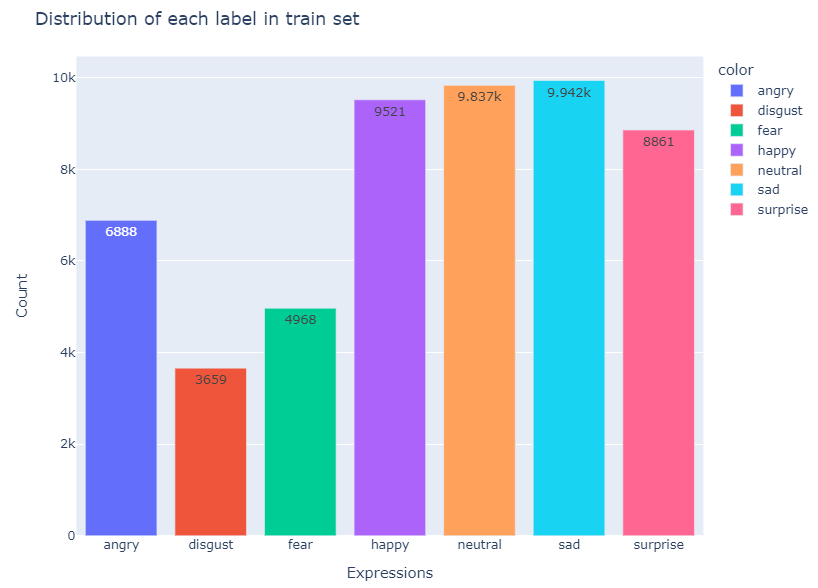
**SMOTE methodology**

The source code "SMOTE for Imbalanced Classification with Python" by Machine Learning Mastery provides a detailed explanation of the SMOTE algorithm. SMOTE generates synthetic data points for the minority class. These new data points are generated by interpolating between existing samples of the minority class. This involves selecting a minority class sample, identifying its nearest neighbors, and then generating new data points along the line segments connecting the sample to its neighbors.

**Application in the recognition of emotions**

In our case, we combined the expw and fer2013 datasets to improve the training data for an AI model for emotion recognition. However, despite combining the datasets, the problem of class imbalance remained. SMOTE was used to compensate for the minority emotion classes and effectively equalize the overall dataset for training.

Figure 1: Distribution before and after SMOTE



**Advantages of SMOTE for emotion recognition**

Using SMOTE to eliminate class imbalance in emotion recognition offers several advantages:

Improved minority class recognition: By balancing the emotion classes, SMOTE ensures that the model recognizes both majority emotions and also adequately represents those of the minority, which leads to better recognition of minority emotions.

Reduced bias: SMOTE mitigates bias towards the majority class and prevents the model from favoring one emotion over others.

Improved overall performance: By effectively recognizing all emotions, the overall performance of the model in recognizing emotions is likely to be improved.

**Implementation considerations**

When implementing SMOTE for emotion recognition, the following factors need to be considered:

Selection of hyperparameters: SMOTE has hyperparameters, such as the number of nearest neighbors and the extent of oversampling, that can affect its performance. These hyperparameters should be carefully tuned based on the specific dataset and task.

Evaluation metrics: For emotion recognition, appropriate metrics such as F1-score or precision-recall curves should be used to evaluate the model's performance for both majority and minority emotion classes.

Possible limitations: SMOTE may introduce noise or overfitting into the data. Careful evaluation and validation is required to ensure that SMOTE is useful for the task at hand.

**Conclusion**

SMOTE is a valuable tool for addressing class imbalances in emotion recognition tasks. By balancing the emotion classes, SMOTE helps to improve the model's ability to effectively recognize a wider range of emotions. However, successful implementation requires careful consideration of hyperparameters, evaluation measures and potential limitations.

**Additional application examples**

In addition to the emotion recognition application described in the source text, SMOTE can also be used for various other classification tasks where class imbalance is a problem. For example, SMOTE can be used to equalize data sets in:

Fraud detection: identification of fraudulent transactions from a large number of legitimate transactions.

Medical diagnosis: Classification of patients based on their symptoms and medical history.

Churn prediction: Predicting which customers are unlikely to use a service.

By effectively handling class imbalances, SMOTE can contribute to improved performance and more reliable decision making on a variety of classification problems.

Brownlee, J. (2021, March 17). *SMOTE for Imbalanced Classification with Python*. Machine Learning Mastery. Retrieved June 7, 2024, from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Methods

Using SMOTE to Balance Data in Facial Expression Recognition Model

In our project, we developed an AI model using VGG16 and MobileNet architectures to recognize facial expressions. The primary aim was to enhance content creation, such as films, by providing producers with data on viewer emotions. One of the significant challenges we faced was dealing with an unbalanced dataset, which consisted of 8624 images for 'angry', 4542 for 'disgust', 6209 for 'fear', 11858 for 'happy', 12324 for 'neutral', 12477 for 'sad', and 11062 for 'surprise'.

To address this imbalance, we employed the Synthetic Minority Over-sampling Technique (SMOTE) from the imblearn.over\_sampling library. SMOTE helps in creating synthetic samples for minority classes by interpolating between existing samples, thereby balancing the dataset.

#### **Technical Overview of SMOTE**

SMOTE works by generating synthetic samples for the minority class rather than simply duplicating existing ones. It selects two or more similar instances from the minority class and generates new instances that are convex combinations of these instances. Specifically, SMOTE performs the following steps:

**Selection of Minority Class Instances**: For each instance in the minority class, SMOTE selects k-nearest neighbors.

**Generation of Synthetic Samples**: For each selected instance, synthetic samples are created along the line segments joining the instance and its k-nearest neighbors. This is done by randomly choosing a point along each line segment.

**Integration into Dataset**: The synthetic samples are then added to the original dataset, resulting in a more balanced class distribution.

The technique's effectiveness lies in its ability to provide more diverse and informative synthetic samples, which helps in training models that generalize better to unseen data Chawla et al., 2002.

#### **Implementation of SMOTE**

**Data Preparation**: We reshaped our input data to a format acceptable by SMOTE, which requires data in the shape (n\_samples,n\_channels×height×width) This reshaping was crucial for SMOTE to function correctly.

**Applying SMOTE**: We initialized SMOTE with random\_state=62 to ensure reproducibility. This step generated synthetic samples to balance our dataset.

**Model Training**: We conducted a comparative analysis by running short training sessions with and without SMOTE. The results were significant:

**With SMOTE**: The model achieved a validation accuracy of 0.41, a categorical accuracy of 0.51, a loss of 1.27, and a validation loss of 1.52.

**Without SMOTE**: The model had a validation accuracy of 0.204, a categorical accuracy of 0.24, a loss of 1.8, and a validation loss of 1.9.

Based on these results, it was clear that applying SMOTE improved the performance metrics substantially. Consequently, we decided to continue training the model using the balanced dataset generated by SMOTE.

Overall, SMOTE proved to be an effective solution for dealing with data imbalance, allowing our facial expression recognition model to perform more reliably across different classes of expressions.

Bowyer, W., Chawla, V., Hall, O., & Kegelmeyer, P. (2011). 11] SMOTE: Synthetic Minority Over-sampling Technique. In *ArXiv*. Retrieved June 5, 2024, from <https://arxiv.org/abs/1106.1813>

# Detailed Explanation of Data Augmentation Techniques

The data augmentation techniques employed in this project play a crucial role in enhancing the diversity and robustness of the training dataset, which is essential for training an effective facial expression recognition model.

**Rotation**: Images were randomly rotated within a range of ±20 degrees. Rotation helps the model generalize better by ensuring it can recognize facial expressions regardless of slight variations in head orientation in real-world scenarios.

**Shift**: Both width and height shifts were applied randomly within ±10% of the image dimensions. This transformation helps simulate different positions of the face within the image, ensuring the model doesn't become overly reliant on the exact positioning of facial features.

**Horizontal Flip**: Images were flipped horizontally randomly. This transformation helps the model learn from facial expressions appearing in both orientations, enhancing its ability to recognize expressions irrespective of whether they appear on the left or right side of the face.

**Brightness Adjustment**: Random adjustments to brightness were made within the range of 0.8 to 1.2. This technique ensures the model is robust to varying lighting conditions, a common challenge in real-world applications.

**Zoom**: Random zooming of images by up to 10% was implemented. Zooming helps the model learn from facial expressions appearing at different scales, ensuring it can recognize expressions both close-up and at a distance.

#### Implementation Using ImageDataGenerator

The ImageDataGenerator from tensorflow.keras.preprocessing.image was utilized to implement these transformations seamlessly within the training pipeline. This generator allows for real-time data augmentation during model training, enhancing efficiency and reducing the need for additional preprocessing steps.

#### How Data Augmentation Works

Data augmentation works by applying these transformations randomly to the training images before feeding them into the model. By introducing variations such as rotations, shifts, flips, brightness adjustments, and zooms, the augmented data helps the model generalize better.

Here’s how each aspect contributes:

**Variation Introduction**: Each transformation introduces variability into the training data, preventing the model from memorizing specific examples and instead learning to extract essential features of facial expressions.

**Improved Robustness**: By training on augmented data, the model becomes more robust to variations it might encounter during inference, such as different head angles, lighting conditions, or facial positions within the image.

**Generalization**: Augmentation reduces overfitting by making the model less sensitive to small variations in the training data, thereby improving its performance on unseen validation or test datasets.

#### Validation of Augmentation Strategy

The effectiveness of the data augmentation strategy was validated through careful monitoring of performance metrics, particularly validation accuracy and loss. The observed improvements in these metrics indicate that the model trained with augmented data generalizes better to unseen data, which is crucial for real-world deployment.

The introduction of data augmentation had the following impact on model performance metrics. This was measured in two test runs and each training was performed in 50 epochs.

Without Augmentation:

* Categorical Accuracy: 0.968
* Loss: 0.09
* Validation Categorical Accuracy: 0.425
* Validation Loss: 3.47

With Augmentation:

* Categorical Accuracy: 0.87
* Loss: 0.34
* Validation Categorical Accuracy: 0.52
* Validation Loss: 2.26

#### Conclusion

In conclusion, data augmentation techniques such as rotation, shift, flip, brightness adjustment, and zoom are essential for training a facial expression recognition model that performs well across diverse conditions. These techniques enhance dataset diversity, improve model generalization, and mitigate overfitting, ultimately leading to more reliable and accurate facial expression recognition in practical applications.

Bownlee, J. (2019, July 5). *How to Configure Image Data Augmentation in Keras*. machinelearningmastery.com. Retrieved June 11, 2024, from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

MLOPS

In this part we will explore the deployment of our emotion recognition AI model on Google Cloud. This document details the configuration, execution, and considerations surrounding this deployment strategy.

### System Architecture

The core of our deployment resides on a Google Cloud virtual machine (VM) instance. We opted for a Ubuntu or Linux-based VM situated in a geographically appropriate region to minimize latency. The VM itself utilizes a standard network configuration and possesses sufficient storage for the model and essential files. Given the nature of our workload, hardware acceleration is not required for this deployment.

### Deployment Process

The deployment process commences with establishing the VM instance on Google Cloud. We leverage the "Compute Engine" section of the Cloud Console to create a new VM instance, configuring the boot disk with the chosen operating system. Region, zone, and VM size are selected based on our performance requirements and cost considerations. Importantly, SSH access is enabled during VM creation to facilitate remote management.

Next, we transfer the Python script responsible for model execution, the emotion recognition model itself, and the haarcascade\_frontalface\_default.xml file onto the VM. Manual upload via the Cloud Console's SSH functionality allows us to securely transfer these files. Once uploaded, we ensure the Python script possesses executable permissions for proper execution.

### Model Execution

Executing the Python script initiates the emotion recognition process. We achieve this by navigating to the script's directory within the VM's SSH console and running the script using the python script.py command.

The script is designed to process a live camera feed. It leverages the haarcascade\_frontalface\_default.xml file to detect faces within the video stream. Subsequently, the emotion recognition model is applied to the detected faces, identifying the most prominent emotions. The script then visualizes the results, displaying the recognized emotions and their corresponding probabilities alongside the live camera feed within a graphical window. This real-time visualization offers a user-friendly interface for interacting with the emotion recognition capabilities.

The file named haarcascade\_frontalface\_default.xml plays a crucial role in your emotion recognition AI model deployment. It's a cascade classifier file used for frontal face detection within images and videos. Here's a breakdown of its functionality:

Haar Cascades: This refers to a machine learning technique for object detection. It utilizes a series of features extracted from images to identify specific objects.

Frontal Face Detection: The haarcascade\_frontalface\_default.xml file is specifically trained to recognize human faces in a frontal orientation. This means it can detect faces that are looking directly at the camera.

XML File Format: The information about the trained cascade classifier is stored in the XML format. This file contains details about the features used for detection and how they are combined to identify faces effectively.

In essence, this file acts as a pre-trained model for face detection within your emotion recognition system. By incorporating it into your Python script, you can efficiently locate faces in the camera feed before applying your emotion recognition model for further analysis.

Due to the complexity and the time it would take to create such a file, we have decided to use it from a Git repository. (link zu citation)

### Scaling and Monitoring

While a VM provides a suitable platform for initial testing and demonstration, its scalability might be limited for handling significant processing demands. To address this in the future, we may consider horizontal scaling by provisioning additional VMs and distributing the workload amongst them. This approach would enhance the system's capacity to accommodate increased usage.

Monitoring the deployed VM is crucial for maintaining its health and performance. Google Cloud Console offers valuable tools for tracking system metrics such as CPU utilization, memory consumption, and network traffic. By monitoring these metrics, we gain insights into the VM's resource usage and can identify potential bottlenecks.

### Security Considerations

Data security remains a primary concern when dealing with any AI application. In this instance, we prioritize secure VM access by utilizing an SSH key. Additionally, adhering to general cloud security best practices, such as applying regular security updates and patches, further strengthens the system's defenses.

### Conclusion

Deploying our emotion recognition AI model on a Google Cloud VM offers a practical and cost-effective solution for our current use case. This approach allows us to leverage the power of Google Cloud while maintaining control over the deployment environment. By carefully considering the configuration, execution, scaling, monitoring, and security aspects outlined in this document, we ensure the ongoing effectiveness of our deployed emotion recognition system. As the field of emotion recognition AI continues to evolve, we can revisit this deployment strategy and incorporate advancements to maintain optimal performance.

*Compute engine*. (n.d.). Google Cloud. <https://cloud.google.com/products/compute?hl=en>

Vpisarev. (2013). *haarcascade\_frontalface\_default.xml [Online forum post]*. GitHub. Retrieved May 21, 2024, from https://github.com/opencv/opencv/blob/4.x/data/haarcascades/haarcascade\_frontalface\_default.xml

## Keeping an Eye on Our Emotion Recognition AI with Wandb

With all the training runs we’re doing you can quickly lose the overview. That's where Wandb comes in! It's like a fitness tracker for our model, keeping an eye on all the important things during training.

### What Does Wandb Track?

Think of Wandb as a scorekeeper during training. It tracks a bunch of stats for every training round. Here's what it keeps tabs on:

**Every Mini-Training (Batch):**

* **Step:** Tracks where we are in each small training chunk (batch).
* **Accuracy:** How good the model is at guessing emotions for each batch.
* **Loss:** Tells us how wrong the model's guesses are on average for each batch.

**Every Training Round (Epoch):**

* **Accuracy:** How good the model is at guessing emotions overall after a complete training round.
* **Round Number:** Keeps track of how many training rounds we've done.
* **Loss (Average):** Tells us how wrong the model's guesses are on average for the entire training round.
* **Test Accuracy:** Checks how well the model does on unseen data to avoid overfitting.
* **Test Loss**: Loss on the unseen data, similar to regular loss.

On top of that, Wandb remembers all the settings we used to build the model, like the number of brain cells (neurons) in each layer and what kind of math it uses. This helps us understand how these choices affect the model's performance. (*What Is W&B? | Weights & Biases Documentation*, n.d.)

### Charts:

Wandb doesn't just keep track of numbers, it shows them off too! We can use Wandb to create charts and graphs that show how the accuracy and loss change over time. These are like visual stories of how our model is learning. By looking at the charts, we can see if the model is getting better or stays stuck. (*What Is W&B? | Weights & Biases Documentation*, n.d.)

### Setting Up Wandb

We tell Wandb about our project by calling a special function in our code. This lets Wandb know what to track and gives our training Run a name so we can identify the run. Wandb also keeps our experiments organized, so we can compare how different models we build perform.(*What Is W&B? | Weights & Biases Documentation*, n.d.)

By using Wandb, we can keep a close eye on our emotion recognition AI and make sure it's on the right track to becoming the best Model.

*What is W&B? | Weights & Biases Documentation*. (n.d.). https://docs.wandb.ai/guides