Contents

Introduction

Ever since we are born and start crying, emotions have been our internal compass, guiding how we think and act. But emotions are not just inside us. We show them on our faces, giving people around us a peek into how we feel. This nonverbal communication is super important for us to connect and understand each other without saying anything.

But figuring out strangers' emotions just by looking at their faces can be tricky. That's where artificial intelligence comes in. Could we use neural networks to train a system that recognizes human emotions from facial expressions?

Imagine a machine learning model that can automatically interpret and record emotional data. This could give us tons of insights, helping us analyse and optimize content in advertising, film, and TV with data-driven emotional responses. By understanding how people emotionally react, we could create more impactful and targeted content, making sure our message hits home.

Exploring emotional recognition through AI is not just about advancing technology; it's about unlocking a new understanding of human communication. It is about helping machines bridge the gap between our internal emotions and external expressions.

Literature Review

Real-Time Facial Expression Recognition

Facial expression recognition in real-time using deep learning has become popular lately because it has so many uses, like improving how people interact with computers, making security systems better, and helping with psychological research. This review looks at the latest developments and methods that researchers have been using in this field.

1. Real-Time Facial Emotion Recognition using Deep Learning

This research presents a system aimed at recognizing facial emotions in real-time using convolutional neural networks (CNNs). The authors highlight how important preprocessing tasks such as detecting and aligning faces are to improve how accurately emotions can be recognized. They also discuss challenges in processing emotions in real-time, like making sure the system runs fast enough and does not lag (Sinha et al., 2019).

2. A Real-Time Facial Expression Recognizer Using Deep Neural Networks

Jeon and Park have developed a real-time system for recognizing facial expressions using deep neural networks, tailored for mobile devices. Their research highlights the importance of optimizing models and employing lightweight architectures to strike a good balance between performance and computational efficiency (Jeon et al., 2016).

3. Facial Emotion Recognition-Based Real-Time Learner Engagement Detection

In this study, researchers explore using facial emotion recognition to detect learner engagement in live online classes. They use deep learning techniques to analyse facial expressions and measure how engaged students are, which can provide teachers with useful information. The authors also discuss the challenges of processing emotions in real-time during online classes, where conditions can change quickly and aren't always predictable (Gupta et al., 2022).

4. Facial expression recognition via ResNet-50

In this research, the authors examined ResNet-50 for its use in recognizing facial expressions, demonstrating its capability to learn intricate features from facial images effectively. They showed that this deep residual network significantly improves the accuracy of facial expression recognition by leveraging its deep structure to capture intricate patterns in facial expressions (Li & Lima, 2021).

5. Facial Expression Recognition using CNN: State of the Art

This study explores the use of Convolutional Neural Networks (CNNs) to automate facial expression recognition (FER), evaluating the impact of different architectures on performance. It criticizes current FER research for heavily relying on basic CNN designs and argues for the adoption of advanced deep CNNs to significantly improve accuracy. The research acknowledges the challenge of dataset bias in the widely used FER2013 benchmark and emphasizes the necessity of larger, more diverse datasets. Additionally, it suggests future research directions, such as developing specialized methods for augmenting FER data and creating a new comprehensive FER dataset (Pramerdorfer & Kampel, 2016).

6. Recognizing Facial Expressions using Deep Learning

In this study, convolutional neural networks (CNNs) are used to classify seven basic human emotions based on facial expressions. The research leverages datasets from Kaggle and Karolinska Directed Emotional Faces (KDEF), evaluating two CNN architectures, VGG-16 and ResNet50, as well as ensemble learning and transfer learning techniques to improve accuracy. The approach achieves a notable 78.3% accuracy on the KDEF dataset, surpassing the Kaggle challenge winner's 71.2% result. Looking ahead, the paper suggests future research directions such as incorporating additional facial and image features, extending emotion recognition to color images and videos, and exploring micro-expressions to enhance the precision of emotion identification methods (Savoiu & Wong, 2021).

7. EmotionNet Nano: An Efficient Deep Convolutional Neural Network for Real-Time FER

EmotionNet Nano is a streamlined and effective deep convolutional neural network designed specifically for quickly recognizing facial expressions in real-time. The study emphasizes the importance of balancing model complexity with fast inference speeds, crucial for applications needing immediate responsiveness. The authors demonstrate the model's effectiveness in various practical settings, including real-time video analysis, highlighting its suitability for real-world applications (Lee et al., 2021).

Conclusion

In conclusion, recent advances in facial expression recognition using deep learning, such as ResNet-50 and VGG-16, have improved how quickly emotions can be detected across different applications. Researchers are refining CNN designs and exploring new techniques like ensemble and transfer learning for better accuracy. They emphasize the need for diverse datasets to ensure these systems work well in various situations. Future research may focus on refining emotion detection with micro-expressions, adding more facial details, and expanding to colour images and videos. These innovations promise to enhance everyday interactions with computers, improve security measures, and deepen our understanding of emotions through advanced deep learning technologies.

Own Contribution

We are combining two datasets, FER2013 and Expressions-in-the-Wild, to enhance our model's robustness and generalizability for real-time applications. Additionally, we are employing data augmentation and oversampling techniques to further strengthen our model's performance.

Data Processing

Data Collection

For our facial expression recognition project, we utilized two datasets: the FER-2013 dataset and the Expressions in the Wild (ExpW) dataset. The FER-2013 dataset was sourced from a Kaggle challenge (Challenges in Representation Learning: Facial Expression Recognition Challenge | Kaggle, 2013; Goodfellow et al., 2013). The ExpW dataset was acquired from a research project on learning social relation traits from face images (Zhang et al., 2015a; Zhang et al., 2015b).

We stored both datasets in Google Drive, making them accessible to all team members for further usage and analysis.

FER2013

We initially planned to use only the FER-2013 dataset, which consists of 35,887 48x48-pixel grayscale images of faces expressing various emotions. These emotions are categorized into seven classes as integer values: 0 (angry), 1 (disgust), 2 (fear), 3 (happy), 4 (sad), 5 (surprise), and 6 (neutral). The dataset is provided as a CSV file, already split into a training set with 28,709 images and a testing set with 7,178 images. The CSV file includes columns for pixel values and the associated emotions.

For our data analysis, we used a Jupyter Notebook. We loaded the dataset into a pandas DataFrame and replaced the integer values of the emotions with their corresponding string labels. We then took samples from the dataset and plotted the images with their associated emotions using Python's Image Library and Matplotlib.

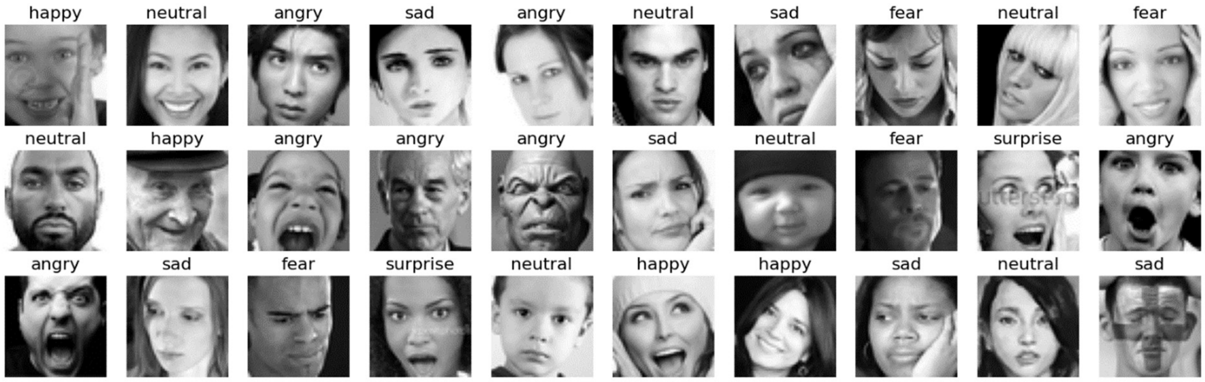


Figure 1: A sample of FER2013 images with associated emotion

We observed that the images were clean and already cropped around the faces, with most captions being accurate in our opinion. However, there were a few instances where we felt the images could have been labelled differently. We recognized that cultural differences might influence how we perceive emotions. Due to time constraints and limited resources, we decided not to alter the labels.

Next, we examined the distribution of the labels. Using Plotly Express, we plotted a bar chart showing the label counts for each emotion. The chart revealed that the dataset is imbalanced, with very few images labelled as "disgust" and a significantly higher number of images labelled as "happy."

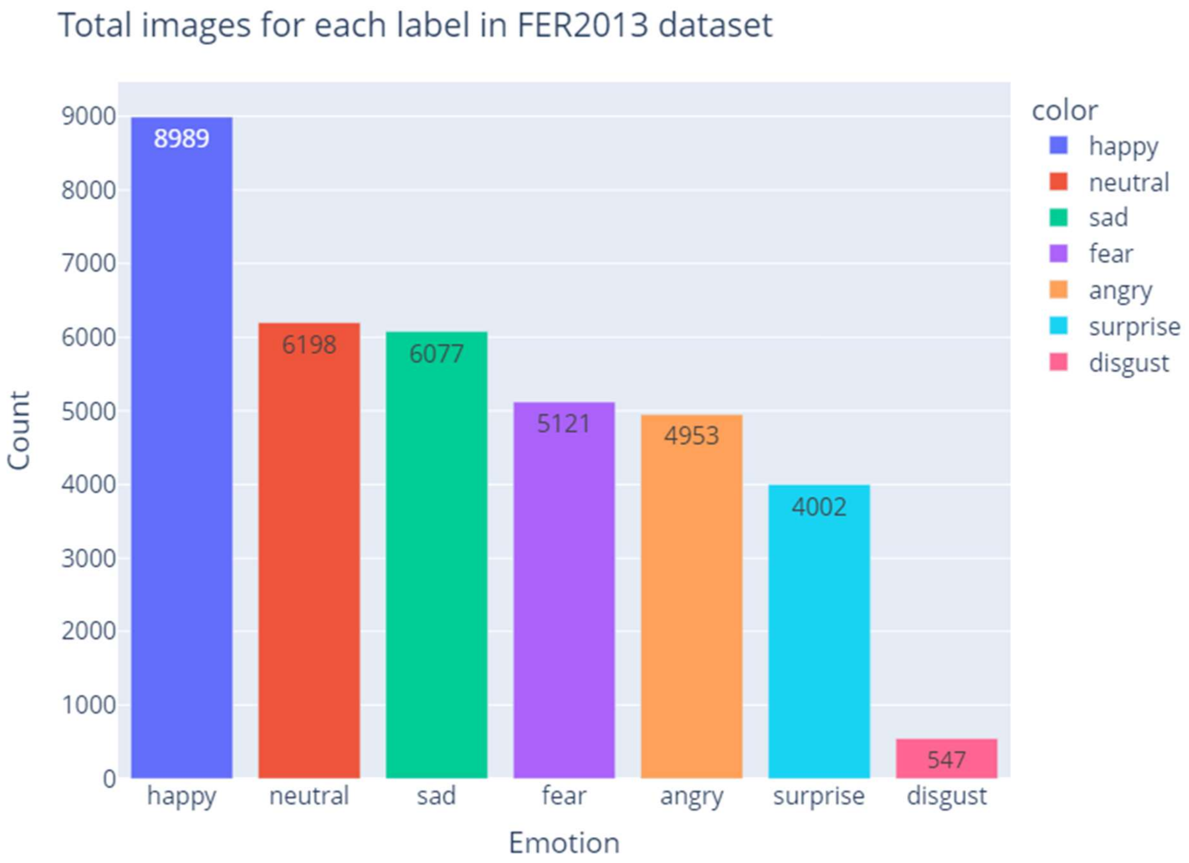


Figure 2: Distribution of emotions in FER2013 dataset

This imbalance can lead to misclassifications because the model lacks sufficient data to learn all classes effectively. To address this issue, we considered searching for an additional dataset. By combining the two datasets, we aimed to achieve a more balanced distribution of emotions and improve the model's performance.

Expressions-in-the-Wild (ExpW)

As previously mentioned, we wanted to address the imbalance in the FER-2013 dataset and find a more diverse dataset with images in various situations to make our model more robust. We discovered the “Expressions in the Wild” dataset, which contains a folder with 91,793 colorful images of various sizes and a separate file with labels. The labels were initially in a List file (.lst), which we converted into a CSV file.

The labels are as follows:

- **image_name:** Name of the image file
- **face_id_in_image:** Indicates the specific face within the image. There are images with multiple faces.
- **face_box_top, face_box_left, face_box_right, face_box_bottom:** These labels define the bounding box coordinates of the detected face
- **face_box_confidence:** Represents the confidence score associated with the face detection
- **expression_label:** 0: angry, 1: disgust, 2: fear, 3: happy, 4: sad, 5: surprise, 6: neutral

For the analysis of this dataset, we used a new Jupyter Notebook. We created a pandas DataFrame for the labels and added headers for each column, as the original file did not include them. We then took a sample of 30 items from the dataset using the pandas sample function with a random state of 42. Using the image names, we loaded the images with the OpenCV library and used the face box coordinates to draw rectangles around the faces with a 10% margin. Finally, we plotted the images using Matplotlib to get an initial feel for the data.

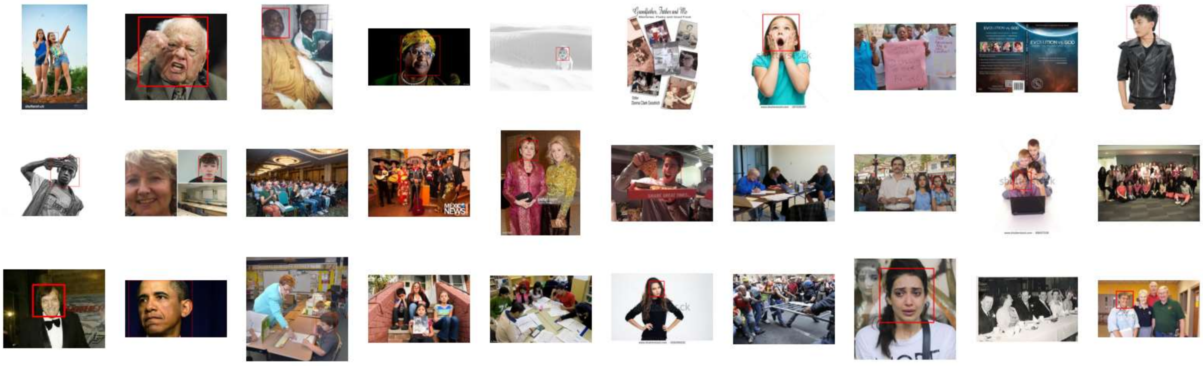


Figure 3: Sample of images with faceboxes before further processing

As you can see, determining the purity of the data in this form is challenging. To address this, we took samples of 30 items with a face box confidence below 50% and 30 items with a face box confidence above 50%, using the same sampling method as before. We then cropped the images around the faces using the face box coordinates and plotted the images using Matplotlib. This allowed us to visually assess the quality and accuracy of the face detections.

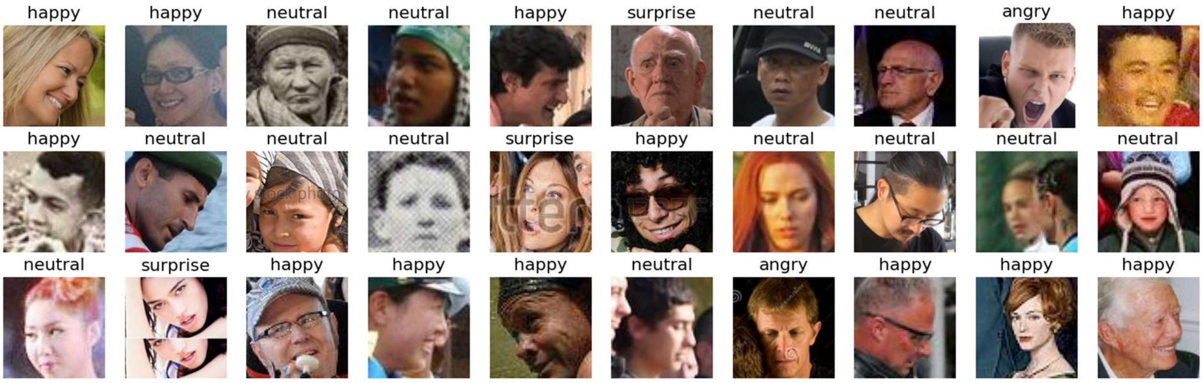


Figure 4:Sample images with facebox confidence lower than 50%

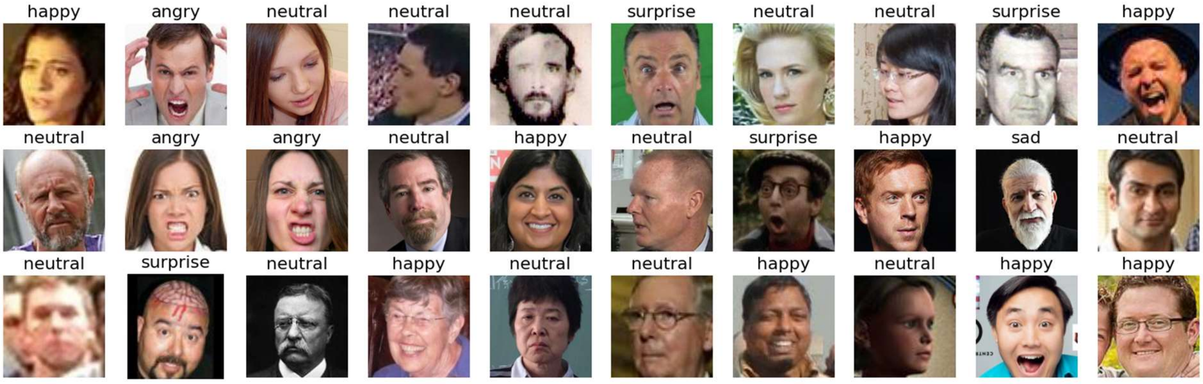


Figure 5: Sample images with facebox confidence higher than 50%

After examining the samples, we found that the images are correctly labelled and of sufficient quality for training, even those with low confidence values. Additionally, the images are more varied, which should help in making our model more robust. Importantly, they resemble the FER-2013 dataset, which is crucial if we decide to merge both datasets.

Similar to the FER-2013 dataset, we analysed the distribution of this dataset and unfortunately found similar issues. There was an overabundance of images labelled as neutral or happy compared to the other expressions.

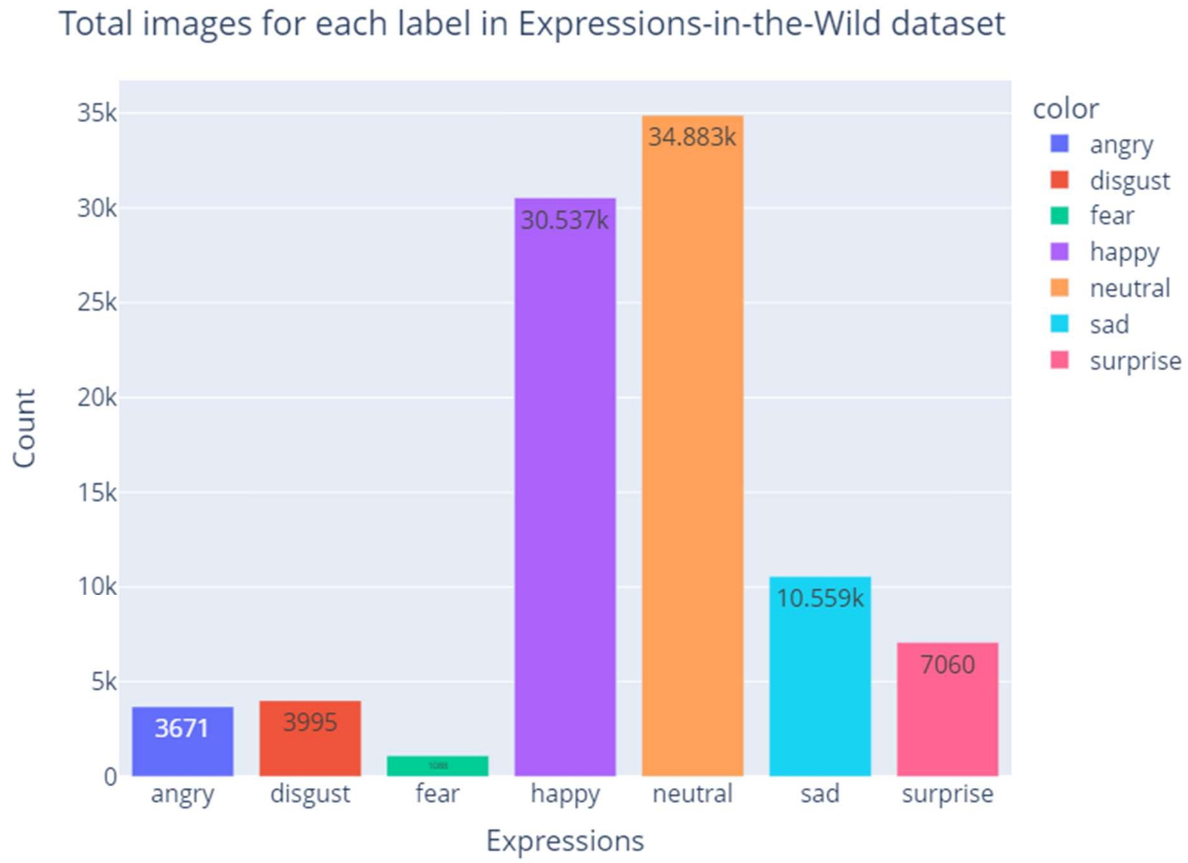


Figure 6: Distribution of emotions in Expressions-in-the-Wild dataset

Given the significant imbalance in both datasets, we opted to merge them and check afterwards how to address this issue.

Combining FER2013 and ExpW-datasets

Before merging the two datasets, we created a new CSV file containing the combined labels from both datasets. Since the FER-2013 dataset lacked image names, we assigned a name to each image, simplifying later image processing tasks.

Once we merged the labels, we re-evaluated the distribution of emotions in the combined dataset.

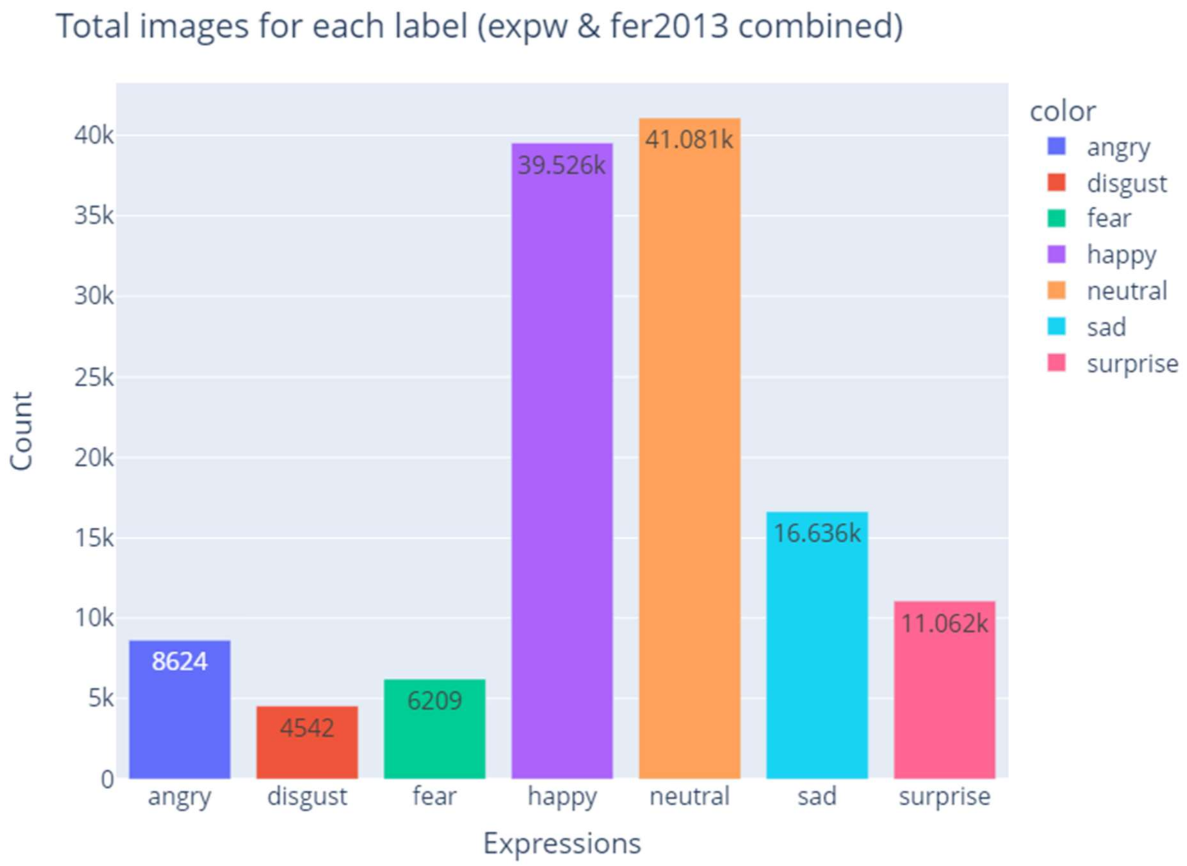


Figure 7: Distribution of emotions in Expressions-in-the-Wild and FER2013 dataset combined

Combining the datasets improved the distribution of angry, disgust, and fear labels, but there was still an overabundance of neutral and happy images. To address this imbalance, we implemented undersampling of the majority classes. Specifically, we randomly dropped 70% of all happy and neutral images, and 25% of the sad images. This adjustment significantly improved the overall distribution of our data.

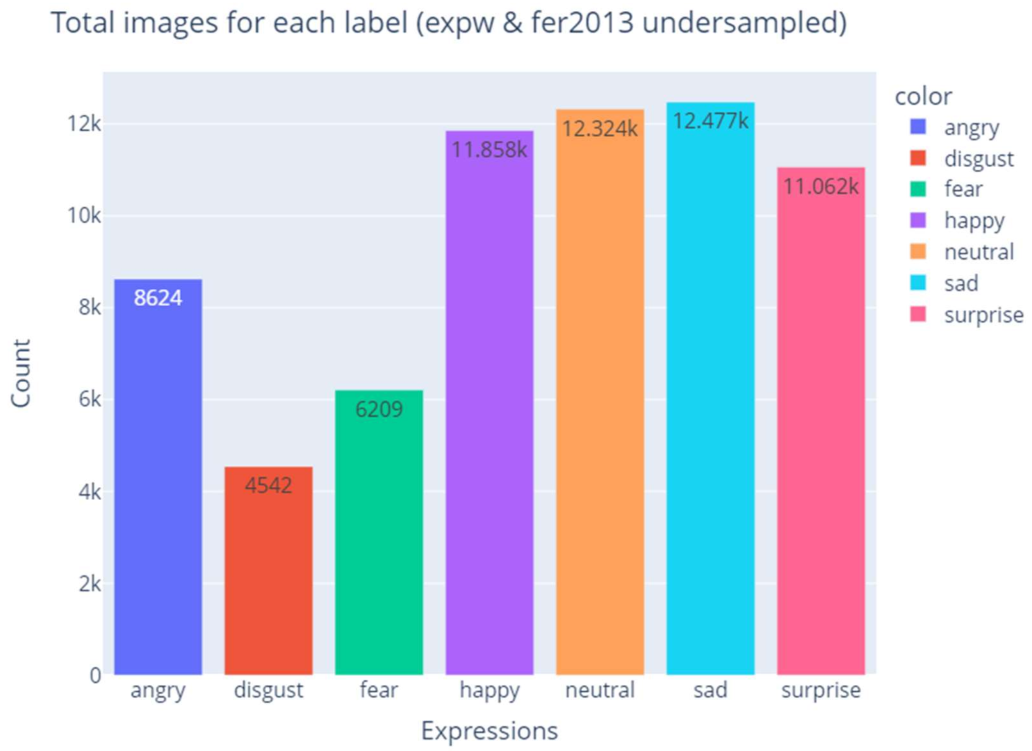


Figure 8: Distribution after undersampling

Our data now shows improved balance, particularly for happy, neutral, sad, and surprise expressions. While the other three expressions are still somewhat underrepresented, we plan to address this using an oversampling technique called SMOTE (more details on this later).

Next, we processed all images by cropping them around the faces, converting them to grayscale, and resizing them to 48x48 pixels (the original size of FER-2013 images). These processed images, along with the corresponding CSV file containing labels, were then saved into a new folder. We compressed the folder into a ZIP file and stored it on Google Drive, ensuring all team members have access to the new dataset.

Methods

Using SMOTE to tackle imbalance in the dataset

In our project, we developed an AI model using VGG16 and MobileNetV2 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=42 to ensure reproducibility. This step generated synthetic samples to balance our dataset. After applying SMOTE our data was now equally distributed.

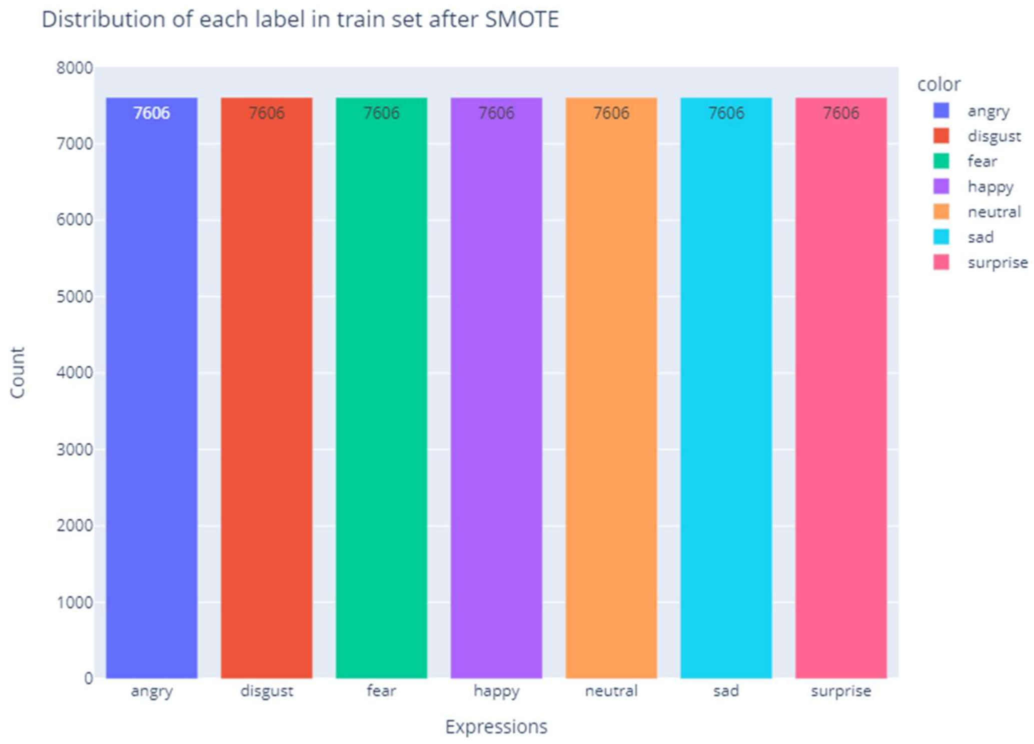


Figure 9: Distribution of data after using SMOTE

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

	Without SMOTE	With SMOTE
Training categorical accuracy	0.24	0.41
Training Loss	1.8	1.27
Validation categorical accuracy	0.2	0.51
Validation loss	1.9	1.52

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.

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 pre-processing 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 (Brownlee, 2019).

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 with MobileNetV2 as our base model and each training was performed in 50 epochs.

	Without Augmentation	**With Augmentation**
Training categorical accuracy	0.97	0.87
Training Loss	0.09	0.34
Validation categorical accuracy	0.43	0.52
Validation loss	3.47	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.

Transfer Learning: MobileNetV2, VGG16, and ResNet

Transfer learning leverages pre-trained models to adapt to new tasks, significantly reducing training time and improving performance, especially when data is limited. Our AI for recognizing emotions employs transfer learning with convolutional neural networks (CNNs), and selecting the right architecture is crucial. Here, we'll explore three prominent architectures for transfer learning: MobileNet, VGG16, and ResNet.

1. MobileNet:

Imagine a world where your emotion recognition AI can run on your phone! MobileNetV2 thrives in this domain. It's a lightweight CNN, meaning it requires less processing power compared to its counterparts. This efficiency makes it ideal for mobile applications where resources are limited. Despite its compact size, MobileNetV2 delivers impressive accuracy, making it a compelling choice for our project. Transfer learning with MobileNetV2 enables rapid adaptation to emotion recognition tasks with limited computational resources. (Howard et al., 2017)

2. VGG16

VGG16, a veteran in the CNN arena, established itself as a benchmark for image recognition tasks. It boasts a straightforward architecture built on numerous convolutional layers. While powerful, VGG16 can be computationally expensive, requiring significant processing resources. This characteristic might make it less suitable for resource-constrained environments. (Simonyan & Zisserman, 2014)

3. ResNet

ResNet stands out for its innovative approach. It tackles the vanishing gradient problem, a hurdle encountered in deep neural networks, by introducing shortcut connections. These connections allow the network to learn from its past layers more effectively, enabling deeper and more accurate models. However, this added complexity can also translate to higher computational demands. While ResNet offered promising accuracy, it proved too slow for processing our live video feed in real-time. Transfer learning with ResNet provides robust feature extraction capabilities, but the computational overhead may not be suitable for real-time applications. (He et al., 2015)

Decision

Considering the need for real-time performance on a video stream, MobileNetV2 emerged as the most suitable candidate for our project. Its lightweight design balances efficiency with acceptable accuracy, allowing for smooth processing of live video data. Transfer learning with MobileNetV2 enabled us to achieve our goal of real-time emotion recognition within the resource constraints of our deployment environment.

Tracking performance with Weights & Biases (W&B)

When we run multiple training sessions, it is easy to lose track of everything. That is why we use W&B. It acts like a fitness tracker for our model, keeping tabs on all the crucial aspects during training.

What do we monitor with W&B?

Every Mini-Training (Batch):

- **Step:** Tracks where we are in each small training chunk (batch).
- **Categorical Accuracy:** Tracks how accurately the model predicts emotions for each batch.
- **Loss:** Tracks the average error in the model's predictions for each batch.

Every Training Round (Epoch):

- **Categorical Accuracy:** Tracks overall prediction accuracy after completing a training round.
- **Round Number:** Counts the total training rounds completed
- **Loss:** Tracks the average prediction error across the entire training round
- **Categorical Validation Accuracy:** Tracks overall prediction accuracy in the validation set.
- **Validation Loss**: Tracks prediction error in the validation set.

In addition, W&B stores all the model-building settings we used, such as architecture and batch size. This information helps us analyze how these choices impact the model's performance

Categorical Accuracy

In our project, we are training a model to distinguish between six distinct emotions: happy, sad, angry, surprise, fear, and neutral. These emotions are separate categories. Categorical accuracy measures how well the model can correctly assign an image to its corresponding emotion category.

For instance, if the model sees a picture of someone smiling and correctly identifies it as "happy," this contributes positively to the categorical accuracy. The model's goal is not to estimate the intensity of happiness but rather to categorize the image into the correct emotional group.

In simpler terms, categorical accuracy indicates how effectively the model places images into the correct emotional categories.

Charts

W&B not only records numerical data but also visualizes it. We utilize W&B to generate charts and graphs that illustrate the evolution of accuracy and loss throughout our training sessions. These visuals act like narratives, depicting how our model learns over time. Examining these charts allows us to discern whether the model is improving or encountering obstacles.

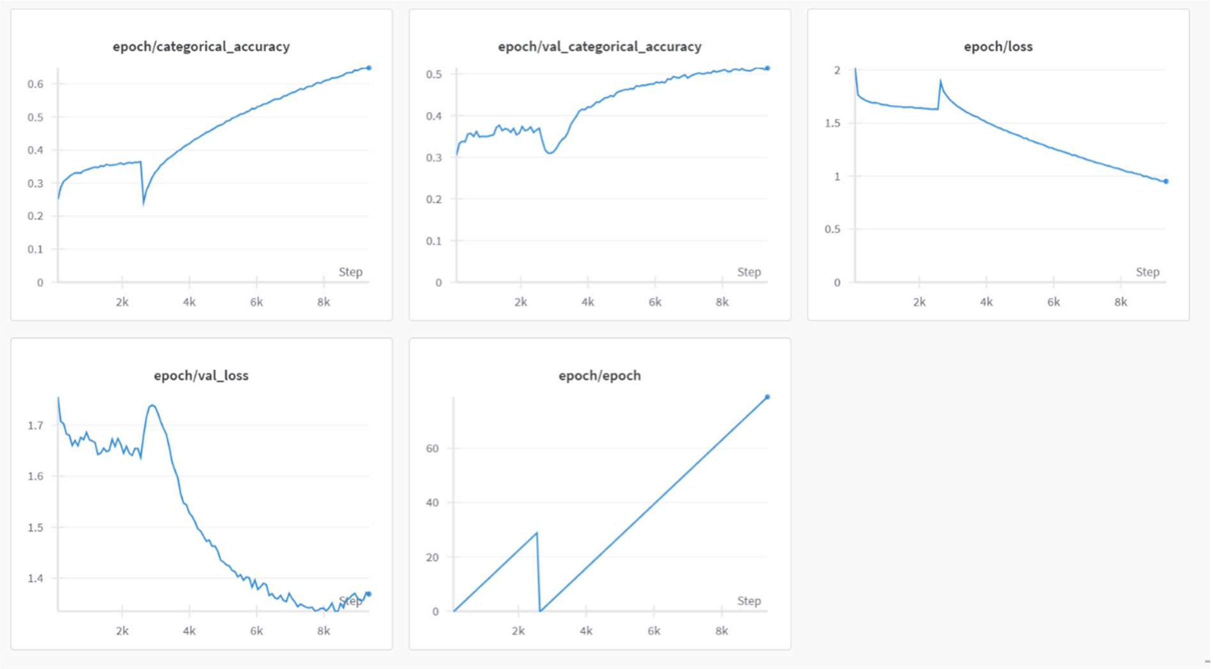


Figure 10: W&B metrics tracker panel

Setting Up Wandb

To integrate W&B into our project, we start by calling the “wandb.init()”-function within our code. This action informs W&B about our project details, including the metrics to monitor, and assigns a unique name to each training run for easy identification. We then use the WandbMetricsLogger class to automatically track metrics during training. This class can be passed as a callback into the “fit()”-method of the tensorflow.keras model.

Model Validation

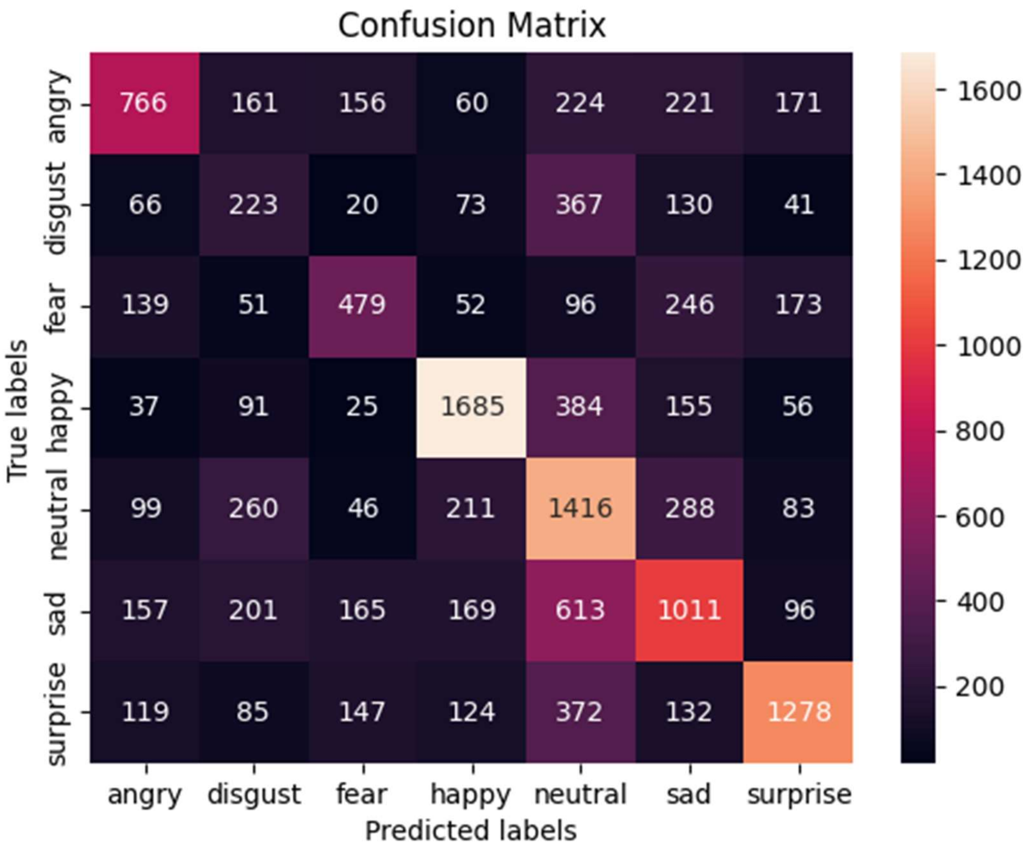
Evaluation With Test Set

After training and tuning our model's parameters, we assessed its accuracy using a test set, which was created with sklearn's "train_test_split" method and set aside before training began.

We used Keras's built-in evaluation method, “model.evaluate()”, with X_test (the pixel values of the images) and y_test (the labels). This evaluation provided us with the accuracy and loss metrics:

- **Categorical Accuracy**: 0.53
- **Loss**: 1.34

Then we used “model.predict()” to calculate the predictions , which we then visualized using a confusion matrix.



	angry	disgust	fear	happy	neutral	sad	surprise
angry	766	161	156	60	224	221	171
disgust	66	223	20	73	367	130	41
fear	139	51	479	52	96	246	173
happy	37	91	25	1685	384	155	56
neutral	99	260	46	211	1416	288	83
sad	157	201	165	169	613	1011	96
surprise	119	85	147	124	372	132	1278

Figure 11: Confusion matrix test set

We can see that most predictions for each class were accurate. However, misclassifications often fell into the neutral category. This suggests that the model struggles with distinguishing expressions that have subtle changes.

Evaluation With Own Generated Test Data

After evaluating our model on the test set, we wanted to assess its performance with images not included in the FER2013 and ExpW datasets. We used an AI image generator to create 10 images per expression, resulting in a total of 70 images. These images were then annotated by a diverse group of people, with the most-voted annotations used as our test labels.

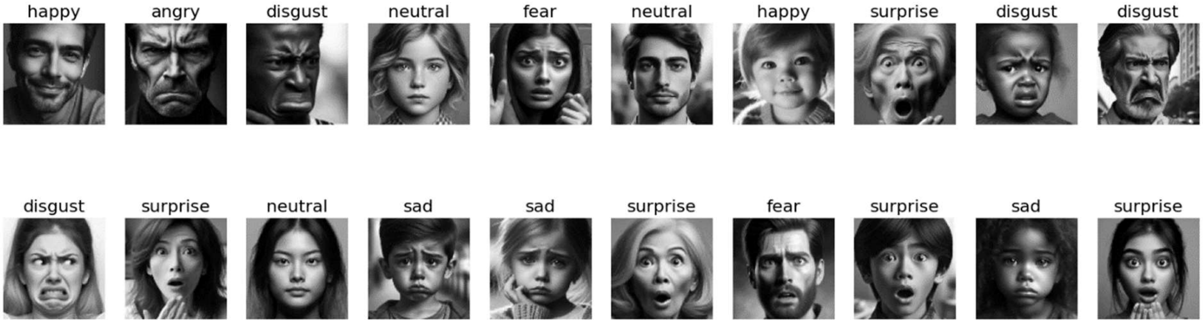


Figure 12: Sample of own generated test data

We utilized Keras's built-in evaluation method, “model.evaluate()”, with X_test (the pixel values of the images) and y_test (the labels). This evaluation provided us with the following metrics:

- **Categorical Accuracy:** 0.54
- **Loss:** 1.61

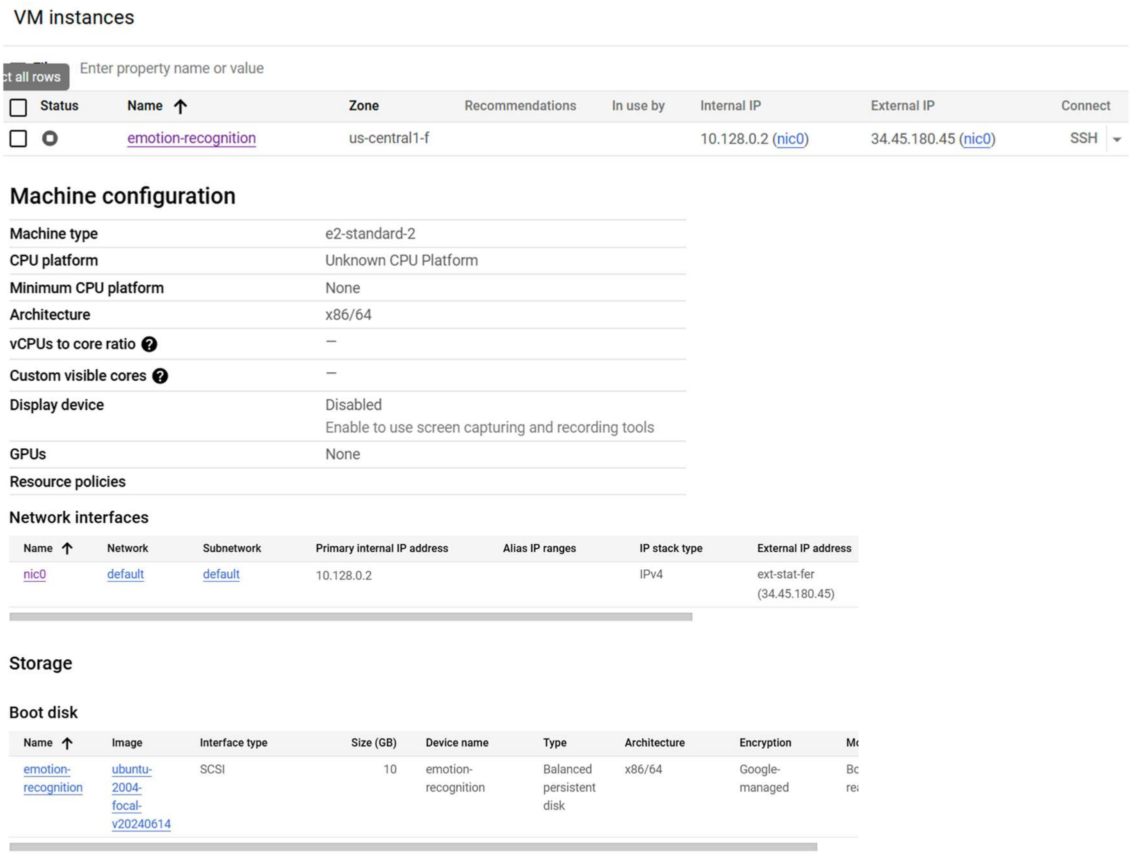
These results indicate that our model performs well with unseen data and appears to be well generalizable.

Machine Learning Operations (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 based VM. 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.



VM instances

ct all rows Enter property name or value

□	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
□	◉	emotion-recognition	us-central1-f			10.128.0.2 (nic0)	34.45.180.45 (nic0)	SSH ▾

Machine configuration

Machine type	e2-standard-2
CPU platform	Unknown CPU Platform
Minimum CPU platform	None
Architecture	x86/64
vCPUs to core ratio ?	—
Custom visible cores ?	—
Display device	Disabled Enable to use screen capturing and recording tools
GPUs	None
Resource policies	

Network interfaces

Name ↑	Network	Subnetwork	Primary internal IP address	Alias IP ranges	IP stack type	External IP address
nic0	default	default	10.128.0.2		IPv4	ext-stat-fer (34.45.180.45)

Storage

Boot disk

Name ↑	Image	Interface type	Size (GB)	Device name	Type	Architecture	Encryption	Mc
emotion-recognition	ubuntu-2004-focal-v20240614	SCSI	10	emotion-recognition	Balanced persistent disk	x86/64	Google-managed	Bc rea

Figure 13: VM configurations GCP

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 (Opencv, 2013).

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.

Table of Figures

References

Brownlee, J. (2019, July 5). *How to configure image data augmentation in Keras*. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

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

Challenges in Representation Learning: Facial Expression Recognition Challenge | Kaggle. (2013). Retrieved February 29, 2024, from https://www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/data

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

Goodfellow, I. J., Erhan, D., Carrier, P. L., Courville, A., Mirza, M., Hamner, B., Cukierski, W., Tang, Y., Thaler, D., Lee, D., Zhou, Y., Ramaiah, C., Feng, F., Li, R., Wang, X., Athanasakis, D., Shawe-Taylor, J., Milakov, M., Park, J., . . . Bengio, Y. (2013, July 1). *Challenges in Representation Learning: A report on three machine learning contests*. arXiv.org. https://arxiv.org/abs/1307.0414

Gupta, S., Kumar, P., & Tekchandani, R. K. (2022). Facial emotion recognition based real-time learner engagement detection system in online learning context using deep learning models. *Multimedia Tools and Applications*, *82*(8), 11365–11394. https://doi.org/10.1007/s11042-022-13558-9

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). *Deep residual learning for image recognition*. arXiv.org. https://arxiv.org/abs/1512.03385

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017, April 17). *MobileNets: efficient convolutional neural networks for mobile vision applications*. arXiv.org. https://arxiv.org/abs/1704.04861

Jeon, J., Park, J., Jo, Y., Nam, C., Bae, K., Hwang, Y., & Kim, D. (2016). A Real-time Facial Expression Recognizer using Deep Neural Network. *The ACM Digital Library*. https://doi.org/10.1145/2857546.2857642

Lee, J. R. H., Wang, L., & Wong, A. (2021). EmotionNet Nano: an efficient deep convolutional neural network design for Real-Time facial Expression recognition. *Frontiers in Artificial Intelligence*, *3*. https://doi.org/10.3389/frai.2020.609673

Li, B., & Lima, D. (2021). Facial expression recognition via ResNet-50. *International Journal of Cognitive Computing in Engineering*, *2*, 57–64. https://doi.org/10.1016/j.ijcce.2021.02.002

Opencv. (2013). *opencv/data/haarcascades at 4.x · opencv/opencv*. GitHub. Retrieved May 21, 2024, from https://github.com/opencv/opencv/blob/4.x/data/haarcascades

Pramerdorfer, C., & Kampel, M. (2016, December 9). *Facial Expression Recognition using Convolutional Neural Networks: State of the Art*. arXiv.org. https://arxiv.org/abs/1612.02903

Savoiu, A., & Wong, J. (2021). Recognizing facial expressions using deep learning. In Stanford University. http://vision.stanford.edu/teaching/cs231n/reports/2017/pdfs/224.pdf

Simonyan, K., & Zisserman, A. (2014, September 4). *Very deep convolutional networks for Large-Scale image recognition*. arXiv.org. https://arxiv.org/abs/1409.1556

Sinha, A., R P, A., Indian Institute of Technology, & Regional Centre IHRD. (2019). Real Time Facial Emotion Recognition using Deep Learning. *IJIIE - International Journal of Innovations & Implementations in Engineering*, *1*, 1–2. https://ijiie.org/download/IJIIE_2019DEC1001VOL1.pdf

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

Zhang, Z., Luo, P., Loy, C. C., & Tang, X. (2015a). *Learning Social Relation Traits from Face Images*. Retrieved April 12, 2024, from https://mmlab.ie.cuhk.edu.hk/projects/socialrelation/index.html

Zhang, Z., Luo, P., Loy, C. C., & Tang, X. (2015b, September 14). *Learning Social Relation Traits from Face Images*. arXiv.org. https://arxiv.org/abs/1509.03936

Appendix

Github Repository and Kanban Board

- **Github Repo:** https://github.com/DSPRO2-Group-9/emotion_recognition
- **Kanban Board:** https://github.com/orgs/DSPRO2-Group-9/projects/3