EMOTION RECOGNITION WITH DEEP LEARNING

ADRIAN REDONDO FERNANDEZ

Thesis supervisor: GERARD ESCUDERO BAKX (Department of Computer Science)

Degree: Bachelor Degree in Informatics Engineering (Computing)

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

26/06/2023

Contents

Contents 2
List of figures and table 5
1. Context and Scope 6
    1.1 Context 6
        1.1.1 Introduction 6
        1.1.2 Situation 6
        1.1.3 Concepts 6
        1.1.4 Problem to be resolved 7
        1.1.5 Stakeholders 7
    1.2 Justification 8
        1.2.1 Previous Studies 8
        1.2.2 Justification 10
    1.3 Scope 10
        1.3.1 Objectives and sub-objectives 10
        1.3.2 Requirements 11
        1.3.3 Potential obstacles and risks 11
    1.4 Methodology and rigor 11
        1.4.1 Methodology 11
        1.4.2 Validation 12
    1.5 References 1 12
2. Temporal Planning 14
    2.1 Description of the tasks 14
        2.1.1 Task definition 14
        2.1.2 Summary of the tasks 15
        2.1.3 Resources 17
    2.2 Risk management 18
        2.2.1 Deadline of the project 18
        2.2.2 Lack of experience in the field 18
        2.2.3 Technical issues 18
    2.3 Gantt 19
    2.4 References 2 20
3. Budget and sustainability 21
    3.1 Budget 21
        3.1.1 Staff costs 21
        3.1.2 Generic costs 23
        3.1.3 Other costs 23
        3.1.5 Management control 24
    3.2 Sustainability 25
        3.2.1 Self-assessment 25
        3.2.2 Economic dimension 25
        3.2.3 Environmental dimension 25

3.2.4 Social dimension 26
3.3 References 3 26
4. Theoretical Background 27
4.1 Deep Learning and Neural Networks 27
4.2 Transfer Learning and Fine Tuning (TL/FT) 27
4.3 Convolutional Neural Networks (CNN) 28
4.4 References 4 30
5. Dataset 31
5.1 Previous research and decision 31
5.2 FER2013 dataset 32
5.2.1 Basic review 32
5.2.2 Limitations 33
5.3 Data source and acquisition 34
5.3.1 Data source 34
5.3.2 Data loading 35
5.4 Preprocessing 35
5.4.1 Image data generator 35
5.4.2 Rescaling 36
5.4.3 Data augmentation 36
5.4.4 Data split 36
5.4.5 Preprocessing for Transfer Learning 37
5.5 References 5 37
6. Transfer Learning and Fine tuning 39
6.1 Overview 39
6.2 TL Models 39
6.2.1 Model overviews 39
6.2.2 Model implementations 43
6.2.3 Model training 46
6.2.4 Model results 46
6.2.5 Decision 50
6.3 Fine tuning 51
6.3.1 FT Model architecture 51
6.3.2 FT Additional Training and Results 53
6.5 References 6 56
7. CNN 57
7.1 Architecture of the model 57
7.1.1 Convolutional Layers 57
7.1.2 Pooling Layers 58
7.1.3 Flattening 59
7.1.4 Fully Connected Layers (Dense) 59
7.1.5 Output Layer 60
7.1.6 Summary 60
7.2 Hyperparameters 61
7.2.1 Batch size 61

7.2.2 Dropout rate and regularization 62
7.2.3 Learning rate 62
7.2.4 Number of epochs 64
7.3 Training process 65
7.3.1 Loss function 65
7.3.2 Optimizer 65
7.4 Results 66
7.4.1 Model Accuracy 66
7.4.2 Confusion Matrix 66
7.4.3 Learning Curves 68
7.6 References 7 72
8. Application of the Deep Learning Model 74
8.1 Introduction 74
8.1.1 Objective 74
8.1.2 Overview 74
8.2 Test Images Generation and Loading 75
8.2.1 Craiyon 75
8.2.2 Image Generation Criteria 75
8.2.3 Image Generation Limitations 76
8.2.4 Images Loading 76
8.3 Face recognition and FER() Detector 77
8.3.1 Face recognition 77
8.3.2 FER() Detector 78
8.4 Model application results and analysis 79
8.4.1 Emotion labeling from the 3 methods 79
8.4.2 Predicted values for the correct emotion 81
8.5 References 8 83
9. Conclusions 84
9.0 indice doc 84
9.1 Overview and Work Development 84
9.2 Results Summary 84
9.3 Results Discussion 84
9.4 Limitations and Areas for Improvement 85
9.5 Final Conclusions 85

List of figures and table

1.1 Image Papers with Code table filtering by "Models not using extra data" 9
2.1 Summary of the tasks 15
2.2 Gantt diagram 19
3.1 Cost per hour of the different roles 21
3.2 Estimated time per task 22
3.3 Estimated cost per task 22
3.4 Total cost of the staff 22
3.5 Incidental costs 23
3.6 Total costs of the project 24
4.1 Neural network conceptual image 27
4.2 Transfer learning visual example 28
4.3 Convolutional neural network conceptual image 28
4.4 Convolutional neural network structure image 29
5.1 FER2013 dataset image example 32
5.2 Usability score description 34
5.3 Data size and type specifications 35
6.1 TL training time histogram 47
6.2 TL complexity histogram 47
6.3 TL accuracy per epoch graph 49
6.4 TL loss per epoch graph 49
6.5 FT accuracy and loss per epoch graphs 1% unfreeze 51
6.6 FT accuracy and loss per epoch graphs 5% unfreeze 52
6.7 FT accuracy and loss per epoch graphs 10% unfreeze 52
6.8 FT accuracy and loss per epoch graphs additional 5 epoch training 53
6.9 FT confusion matrix 54
7.1 CNN confusion matrix 67
7.2 CNN loss function per epoch 1-25 graph 68
7.3 CNN accuracy per epoch 1-25 graph 69
7.4 CNN loss function per epoch 26-55 graph 70
7.5 CNN accuracy per epoch 25-55 graph 71
8.1 Craiyon interface image 76
8.2 PyPI face recognition documentation features example 77
8.3 PyPI FER documentation example 78
8.4 Table of predictions of the 3 models 79
8.5 FER correct emotion values mean histogram 81
8.6 FT correct emotion values mean histogram 81
8.7 CNN correct emotion values mean histogram 82

1. Context and Scope

1.1 Context

1.1.1 Introduction

Emotions play a crucial role in human interactions and influence our decisions, thoughts, and behaviors. Recognizing and understanding emotions is important in many fields, such as psychology, education, marketing, and healthcare. With the recent advances in deep learning and artificial intelligence, automated emotion recognition from visual data, such as facial images, has become an active area of research.

Recognizing emotions from facial images is a challenging task, as emotions can be subtle and complex. However, deep learning techniques have demonstrated remarkable capabilities in extracting meaningful features and patterns from images, which provides promise for the potential success of these techniques in emotion recognition. As such, automated emotion recognition from facial images is an exciting and active area of research, with the potential to make significant contributions to various fields.

1.1.2 Situation

This is a Bachelor Thesis of the Computer Engineering Degree, specialization in Computing, done in the Facultat d'Informàtica de Barcelona of the Universitat Politècnica de Catalunya directed by Gerard Escudero Bakx, professor from the Computer Science Department.

1.1.3 Concepts

The reader must be familiarized with the following concepts to better understand the following sections of this thesis.
In section "4 Theoretical Background", the deep learning theoretical concepts will be further explained. Also the FER2013 dataset will be dissected in section "5 Dataset"

FER2013 dataset

The FER2013 dataset is a collection of facial images labeled with one of seven emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. The dataset contains 35,887 grayscale images with a size of 48x48 pixels. It was compiled for the Facial Expression Recognition 2013 Challenge, and is widely used in research related to emotion classification using deep learning techniques.

Keras

Keras is a popular open-source deep learning framework written in Python. It provides an easy-to-use interface for building and training deep neural networks, and is known for its user-friendliness and simplicity. Keras has become a standard in the deep learning community due to its versatility, reliability and performance.

Transfer Learning and Fine tuning

Transfer learning and fine tuning are techniques commonly used in deep learning for image recognition. Transfer learning consists in using a pre-trained model, trained on a large dataset, as a starting point for a new model with a smaller dataset. Fine tuning involves retraining the last layers of the pre-trained model on the new dataset. Both can lead to significant improvements in model performance. Examples of pre-trained models include MobilNet, Inception, ResNet, VGG and VGGNet, and DenseNet.

Convolutional neural network (CNN)

Convolutional neural networks are a type of neural network specifically designed for image recognition tasks. They are composed of multiple layers of filters that extract features from the input image, followed by fully connected layers that make the final classification decision. CNNs have been widely used in the field of emotion classification using facial images, and have achieved state-of-the-art results on the FER2013 dataset.

1.1.4 Problem to be resolved

Deep learning is a field of study that has grown a lot in recent years in which many new techniques and methods arise frequently. For this reason it is important to carry out feasibility and comparative studies about the different techniques.
The goal of this project is to explore and compare different deep learning techniques, such as transfer learning, fine tuning and convolutional neural networks (CNNs), for emotion recognition from facial images using the FER2013 facial image dataset.
This will be achieved by attempting to create two custom models (one using transfer learning and fine-tuning, and another by building a CNN) and comparing them with the 'fer' Python library.

1.1.5 Stakeholders

Besides the researcher, the primary stakeholders are the thesis director, Gerard Escudero Bakx, and the GEP tutor, Joan Sardà Ferrer. They are responsible for supervising and guiding the development of the thesis project, ensuring that it meets the academic standards and requirements of the institution. Their feedback and support are crucial for the successful completion of the project.

Another stakeholder is the academic community, particularly those interested in the field of deep learning and computer vision. The thesis project could contribute to the advancement of knowledge in this area by exploring and evaluating the effectiveness of different techniques for emotion recognition from facial images.

The FER2013 dataset is also a significant stakeholder. The quality and availability of the dataset will have a direct impact on the project's success. The dataset was created by the researchers in the Computer Vision Center at the Autonomous University of Barcelona, and making use of it for the thesis will be an opportunity to contribute to the recognition of their work.
Lastly, the wider community could benefit from the project's results if they are used to develop more accurate and reliable models for emotion recognition from facial images. Potential applications include facial expression recognition systems for people with autism spectrum disorders or as a component of human-robot interaction.

1.2 Justification

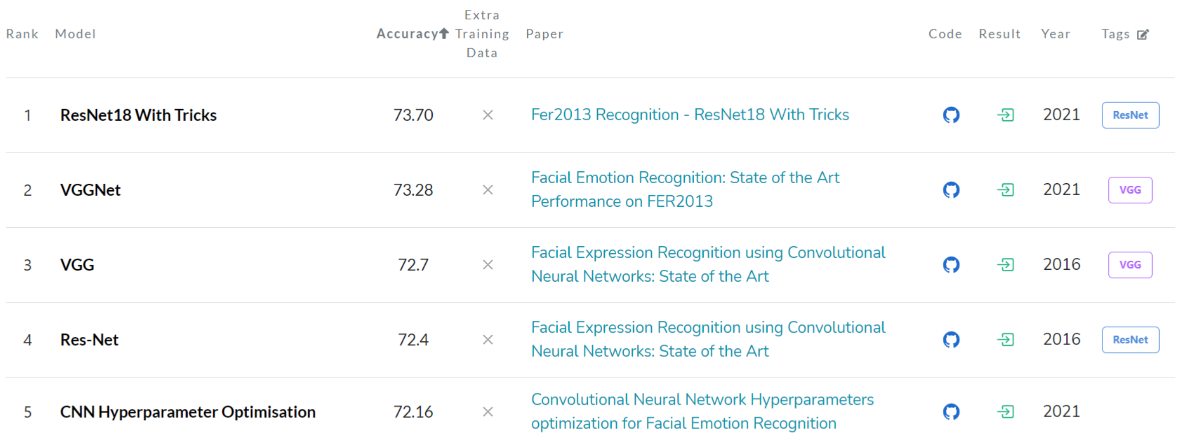
1.2.1 Previous Studies

The website "Papers with Code" provides a comprehensive summary of the state-of-the-art (SOTA) methods for facial expression recognition on the FER2013 dataset. The site offers an extensive collection of papers, along with their associated code repositories, showcasing the latest advancements in this field.

In this website, the listed models present a range of approaches for facial emotion recognition, highlighting the continuous efforts to improve accuracy in this field. Several key observations can be made from it:

1. Models with Additional Training Data: Notably, the top five models on the list have been trained with additional data, leading to higher accuracy scores. This suggests that incorporating supplementary datasets can positively impact model performance and generalization.
2. Average Accuracy Comparison: Comparing the average accuracy of the top five models with additional training data (76.36%) to the average accuracy of the models ranked 6 to 10 without additional data (73.05%), a notable difference of approximately 3.31% is observed. This indicates that utilizing additional training data can significantly contribute to improved facial emotion recognition results.
3. Architecture Importance: Various architectures, such as ResMaskingNet, VGG-19, ResNet, and LHC-Net, have been successfully employed in the top-performing models. This highlights the importance of selecting appropriate architectures tailored to the specific task of facial emotion recognition.
4. Attention Mechanisms: The inclusion of attention mechanisms, as seen in LHC-Net and Self-AttentionResNet, demonstrates their efficacy in capturing essential facial features for accurate emotion recognition.
5. Optimization Techniques: Models such as ResNet18 With Tricks and CNN Hyperparameter Optimization emphasize the importance of implementing optimization techniques to enhance model performance. This suggests that carefully choosing and fine-tuning model parameters can lead to improved accuracy.

The top 5 models shown in Papers with Code use extra data. However, filtering for “Models not using extra data” reveals that the top accuracy models use either transfer learning pretrained models or focus on optimizing hyperparameters of a convolutional neural netw



Rank	Model	Accuracy	Extra Training Data	Paper	Code	Result	Year	Tags
1	ResNet18 With Tricks	73.70	×	Fer2013 Recognition - ResNet18 With Tricks			2021	ResNet
2	VGGNet	73.28	×	Facial Emotion Recognition: State of the Art Performance on FER2013			2021	VGG
3	VGG	72.7	×	Facial Expression Recognition using Convolutional Neural Networks: State of the Art			2016	VGG
4	Res-Net	72.4	×	Facial Expression Recognition using Convolutional Neural Networks: State of the Art			2016	ResNet
5	CNN Hyperparameter Optimisation	72.16	×	Convolutional Neural Network Hyperparameters optimization for Facial Emotion Recognition			2021	

(1.1 Image Papers with Code table filtering by "Models not using extra data")

VGGNet and VGG:

- Accuracy: 73.28% and 72.7%, respectively.
- Papers: "Facial Emotion Recognition: State of the Art Performance on FER2013" (2021) and "Facial Expression Recognition using Convolutional Neural Networks: State of the Art" (2016), respectively.
- Noteworthy: Both models employ VGG architectures, with the former achieving a slightly higher accuracy. These results highlight the effectiveness of VGGNet in facial emotion recognition tasks.

Res-Net and ResNet:

- Accuracy: 72.4% for Res-Net and unspecified for ResNet.
- Paper: "Facial Expression Recognition using Convolutional Neural Networks: State of the Art" (2016)
- Noteworthy: These models, utilizing Residual Networks (ResNets), provide competitive accuracy. Residual connections enable the models to effectively capture and learn intricate facial features.

CNN Hyperparameter Optimization:

- Accuracy: 72.16%
- Paper: "Convolutional Neural Network Hyperparameters optimization for Facial Emotion Recognition" (2021)
- Noteworthy: This work emphasizes the significance of hyperparameter optimization techniques in CNNs for enhancing facial emotion recognition performance. The achieved accuracy demonstrates the impact of carefully tuning hyperparameters on model effectiveness.

VGGNet demonstrates superior accuracy compared to VGG, highlighting its effectiveness in accurately recognizing facial emotions. Similarly, Res-Net, with an accuracy of 72.4%, showcases competitive performance by effectively capturing intricate facial features through Residual Networks.

Furthermore, the work on CNN hyperparameter optimization emphasizes the significance of fine-tuning hyperparameters, resulting in a commendable accuracy score of 72.16%. This underscores the impact of optimizing hyperparameters to maximize the model's effectiveness in facial emotion recognition tasks.
Understanding that models with an accuracy of around 70% effectiveness are considered very good models, we can consider our models to have an acceptable accuracy if they approach this figure around 60%, and they are considered very good if they significantly exceed it, around 80%.

1.2.2 Justification

Although there have been several previous studies on the use of deep learning techniques for emotion classification from facial images using the FER2013 dataset, there is still room for improvement and further exploration of different approaches.

Transfer learning, CNNs have shown great promise in achieving considerable accuracy in emotion classification tasks. However, there is still a need to compare and evaluate the feasibility and effectiveness of these techniques.

Therefore, a project that studies the feasibility and comparative of the mentioned deep learning techniques for emotion classification from facial images will hopefully contribute to the advancement of the field of Computer Vision and facilitate the development of more accurate and efficient models for emotion recognition. This will have practical implications for various industries such as marketing, healthcare, and entertainment.

1.3 Scope

1.3.1 Objectives and sub-objectives

The general objective of this project is to explore and compare the performance of different deep learning techniques for emotion classification from facial images using the FER2013 dataset, specifically through the application of transfer learning and fine tuning, and convolutional neural networks (CNNs). To achieve this objective, the project can be broken down into several sub-objectives:

Theoretical Part
- Study the theoretical concepts of deep learning techniques, including transfer learning, fine tuning and CNNs.
- Explore and compare the different architectures and variations of transfer learning, CNNs, and VAEs for emotion classification from facial images.
- Analyze the strengths and weaknesses of each technique and compare them to other state-of-the-art methods in the field.

Practical Part
- Implement and program the selected deep learning techniques using Keras with Tensorflow backend.
- Train and evaluate the performance of the different models using standard evaluation metrics such as accuracy and loss.
- Compare the results obtained from the different models and analyze the strengths and weaknesses of each technique for emotion classification from facial images.
- Apply the trained models to a set of images and compare their performance with the python ‘fer’ library.

1.3.2 Requirements

- Ensure that the data from the dataset can be transferred without errors to the models for training.
- Optimize the hyperparameters of the different deep learning models.
- Use a suitable evaluation process to select the best model for each architecture during the training period.
- Use good programming practices, with a readable style and minimal complexity.

1.3.3 Potential obstacles and risks

- Deadline of the project: Meeting the project deadline and ensuring that the work is well-distributed to avoid rushing can be a major risk. Currently, I have a part time job that limits the amount of time I can dedicate to this project.
- Lack of experience in the field: The field of deep learning and computer vision requires a high level of technical knowledge. Personally, I have some experience with machine learning models and using python for data science. However it is the first time for me trying to apply deep learning models to a real life problem. This lack of experience may cause an increase in the time devoted to research and learning tasks.
- Technical issues: We have to account for the possibility of technical issues with online tools, which may lead to delays in training and testing the models. Mainly the tool with more risk is Google Colab since a lot of hours will be devoted to working with it. The overuse of the GPU mode in Google Colab can cause timeouts. Furthermore, the GPU mode can only be used in a notebook at a time so this will limit the possibility of working with two scripts at the same time.

1.4 Methodology and rigor

1.4.1 Methodology

The working methodology for this project is based on the Agile methodology, which is a flexible and iterative approach that allows for continuous improvement and adaptation throughout the project. The work will be divided into sprints, with each sprint focusing on a series of work objectives, which in turn will be divided into subtasks. Of course, in order to make planning easier the subtasks will have to be assigned a score based on the estimated time they will require. Examples of subtasks would be implementing a data preprocessing function, researching CNN implementation examples, researching transfer learning pre-trained models compatible with the dataset, etc.

1.4.2 Validation

At the beginning of each sprint, the subtasks will be identified and prioritized based on their importance and feasibility, and work will commence accordingly. Throughout the sprint, regular meetings with the tutor using Google Meet and Google Calendar will take place to review the progress, address any issues or challenges that may arise, and make necessary adjustments to the plan. If the work is clear and straightforward the review can be discussed by email instead of a meeting.

At the end of each sprint, there will be a review to evaluate the progress made, the subtasks completed, and the quality of the work delivered. This review will also provide an opportunity to reflect on the lessons learned and identify any areas for improvement in the upcoming sprints.

1.5 References 1

- *FER-2013. (2020b, July 19). Kaggle.* https://www.kaggle.com/datasets/msambare/fer2013 *[13/06/2023]*
- *Team, K. (n.d.-b). Keras: Deep Learning for humans.* https://keras.io/ *[20/03/2022]*
- *Team, K. (n.d.-c). Keras documentation: Why choose Keras?* https://keras.io/why_keras/ *[13/06/2023]*
- *Team, K. (n.d.). Keras documentation: Transfer learning & fine-tuning.* https://keras.io/guides/transfer_learning/ *[13/06/2023]*
- Alberto, R. (2022, January 6). Redes Neuronales: ¿Qué es Transfer Learning y Fine Tuning? *Medium.* https://rubialesalberto.medium.com/redes-neuronales-qu%C3%A9-es-transfer-learning-y-fine-tuning-8259a81cfdbc *[13/06/2023]*

- Singhal, A. (2021, December 26). Facial expression detection using Machine Learning in Python. *Medium.* *https://medium.com/analytics-vidhya/facial-expression-detection-using-machine-learning-in-python-c6a188ac765f [13/06/2023]*
- *face-recognition. (2020, February 20). PyPI.* *https://pypi.org/project/face-recognition/ [13/06/2023]*
- *Papers with Code - FER2013 Benchmark (Facial Expression Recognition (FER)). (n.d.).* *https://paperswithcode.com/sota/facial-expression-recognition-on-fer2013 [13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

2. Temporal Planning

2.1 Description of the tasks

2.1.1 Task definition

Project Planning:

In order to ensure a successful project, project planning is crucial. For it we define the following tasks:

• T1.1 Context and Project Scope: Definition of the project scope in the context of its study, indicating the objective of the thesis, the relevance of the area, and how it is going to be developed.
• T1.2 Temporal Planning: Planning of the entire execution of the work, providing a description of the phases, resources, and requirements needed.
• T1.3 Budget and Sustainability: Analysis of the sustainability and the economic dimension of the project.
• T1.4 Integration in Final Document: Bring together all the tasks, correcting the parts that were mistaken.
• T1.5 Meetings: Meetings with the supervisor will be scheduled every two weeks with the purpose of ensuring that the thesis is evolving correctly and fulfilling the time plan.

Research:

Before starting to work with the different techniques, one important task is to familiarize oneself with the basics of deep learning, specifically CNNs and VAEs, and their application in emotion classification. As it is mandatory to study the techniques extensively, we are going to perform the following tasks:

• Research and learn about transfer learning and fine tuning and how they can adapt to the problem of emotion classification.
• Research about CNN and how it can be applied to emotion classification.
• Additional research of other methods or techniques
• Research the performance of the techniques on different datasets, including FER2013.

Practical Implementation:

In order to compare the mentioned techniques in emotion classification, we have to program each one of them. We divide the practical implementation in the following tasks:

• Learn TensorFlow and Keras. It is of utmost importance to have a solid understanding of these tools before trying to implement the models.
• Load and preprocess the FER2013 dataset.

• Program a Fine tuned model with transfer learning using pre-trained models.
• Program a CNN model for emotion classification.
• Program the application of the models and the comparison with the ‘fer’ library.
• Code cleaning, testing and code optimization.

Experimentation and Conclusion:

In this section, we will define the tasks relative to the experiments, and conclusions of our project.

• Experiment with the Transfer Learning/Fine tuning model.
• Experiment with the CNN model.
• Make tests regarding the application of the models
• Conclusions: Analyze the results from the experimentation with the models and draw some conclusions.
• Prepare the oral defense

Furthermore, the documentation of the project is an implicit task that must be carried out throughout the thesis.

2.1.2 Summary of the tasks

ID	Name	Time(h)	Dependencies	Resources
T1	**Project management**	100		
T1.1	Context and Scope	25		PC, Google Docs, GEPTutor, Researcher(R)
T1.2	Temporal Planning	15		PC, Google Docs, Asana(gantt), GEPTutor, R
T1.3	Budget and Sustainability	20	T1.3	PC, Google Docs, GEPTutor, R
T1.4	Final project definition	20	T1.2, T1.3, T1.4	PC, Google Docs, GEPTutor, R
T1.5	Meetings	20		Tutor, R

T2	**Research**	125		PC, Google, Kaggle, ChatGPT, R
T2.1	Research TL/FT	30		PC, Google, Kaggle, ChatGPT, R
T2.2	Research CNN	40		PC, Google, Kaggle, ChatGPT, R
T2.3	Research other	45		PC, Google, Kaggle, ChatGPT, R
T2.4	Research dataset	10		PC, Google, Kaggle, ChatGPT, R
T3	**Practical implementation**	180		
T3.1	Learn Keras and Tensorflow	30		PC, Google, Keras, ChatGPT, R
T3.2	Load and preprocess dataset	10	T2.4	PC, Google Colab, Kaggle, ChatGPT, R
T3.3	Program a Transfer Learning model	40	T2.1, T3.1, T3.2	PC, Google, Google Colab, Keras, Kaggle, ChatGPT, R
T3.4	Program a CNN model	40	T2.2, T3.1, T3.2	PC, Google, Google Colab, Keras, Kaggle, ChatGPT, R
T3.5	Program App	50	T2.3, T3.1, T3.2	PC, Google, Google Colab, Keras, Kaggle, ChatGPT, R
T3.6	Code cleaning	10	T3.3, T3.4, T3.5	PC, Google, Google Colab, Keras, ChatGPT, R
T4	**Experimentation and conclusions**	115		
T4.1	Experiments with TL/FT	25	T3.3	PC, Google, Google Colab, Keras, ChatGPT, R

T4.2	Experiments with CNN	25	T3.4	PC, Google, Google Colab, Keras, ChatGPT, R
T4.3	Tests App	30	T3.5	PC, Google, Google Colab, Keras, ChatGPT, R
T4.4	Conclusions	10	T4.1, T4.2, T4.3	PC, results obtained, R
T4.5	Prepare oral defense	25	T4.4	PC, Google Drive, Google, Google Slides, results obtained, ChatGPT, R
T5	**Project documentation**	50		PC, results obtained, Google Docs, Google Chrome Markers, R
	TOTAL	570		

(2.1 Summary of the tasks)

2.1.3 Resources

Human resources

The main human resource of the thesis is the researcher, since is the one working on it. The university tutor Gerard Escudero Bakx will mentor the researcher on its task, so he is a fundamental human resource. Also, the GEP tutor, Joan Sardà Ferrer, is in charge of correcting the project management part, so he will have a big influence in the scope and organization of the work.

Tool resources

Considering that research projects are grounded on previous works, there is a certain need for material resources such as books or papers. Furthermore, I will need some software and hardware resources to make the practical part and the experiments:

• PC. The experiments will be done with a Lenovo YOGA 720 laptop computer, with 8GB of RAM and Intel(R) Core(TM) i5-7200U CPU.
• Google Drive. Used for document management and keeping all the thesis-related files in one place, accessible for anyone who needs to review them.
• Google Slides: Tool used for creating visual presentations.

• Google Docs: Used to write the documentation of the thesis.
• Atenea. Used to communicate with the professor in charge of GEP, learn about project management and deliver the work.
• Kaggle. A platform to access and download datasets for experimentation and analysis.
• Google Colab. A cloud-based service that provides free access to GPUs and TPUs for running deep learning experiments.
• Keras. A high-level neural networks API, written in Python and capable of running on top of TensorFlow or other deep learning libraries.
• ChatGPT. A language model trained by OpenAI, which can assist with project-related queries and provide guidance on various topics.

2.2 Risk management

2.2.1 Deadline of the project

• Impact: Medium
• Proposed solution: If the deadline of the project is compromised, depending on the stage the project is a possible solution could be to devote more hours to the implementation and experimentation rather than the documentation. Also establishing a priority hierarchy over the models would be useful. CNN is the most important as it is the basic one, followed by the VAE model and lastly the Transfer Learning and Fine tuning approach. In this way the deadline of the project will not be compromised and no new tasks will be needed.
• Risk plan: Identify that the deadline of the project is at risk -> Evaluate the progress of the project -[if the project is advanced]-> Reassign 10 hours from documentation tasks to experimentation. -[else]-> Decide which model to drop from the thesis and focus on the implementation and experimentation of the others.

2.2.2 Lack of experience in the field

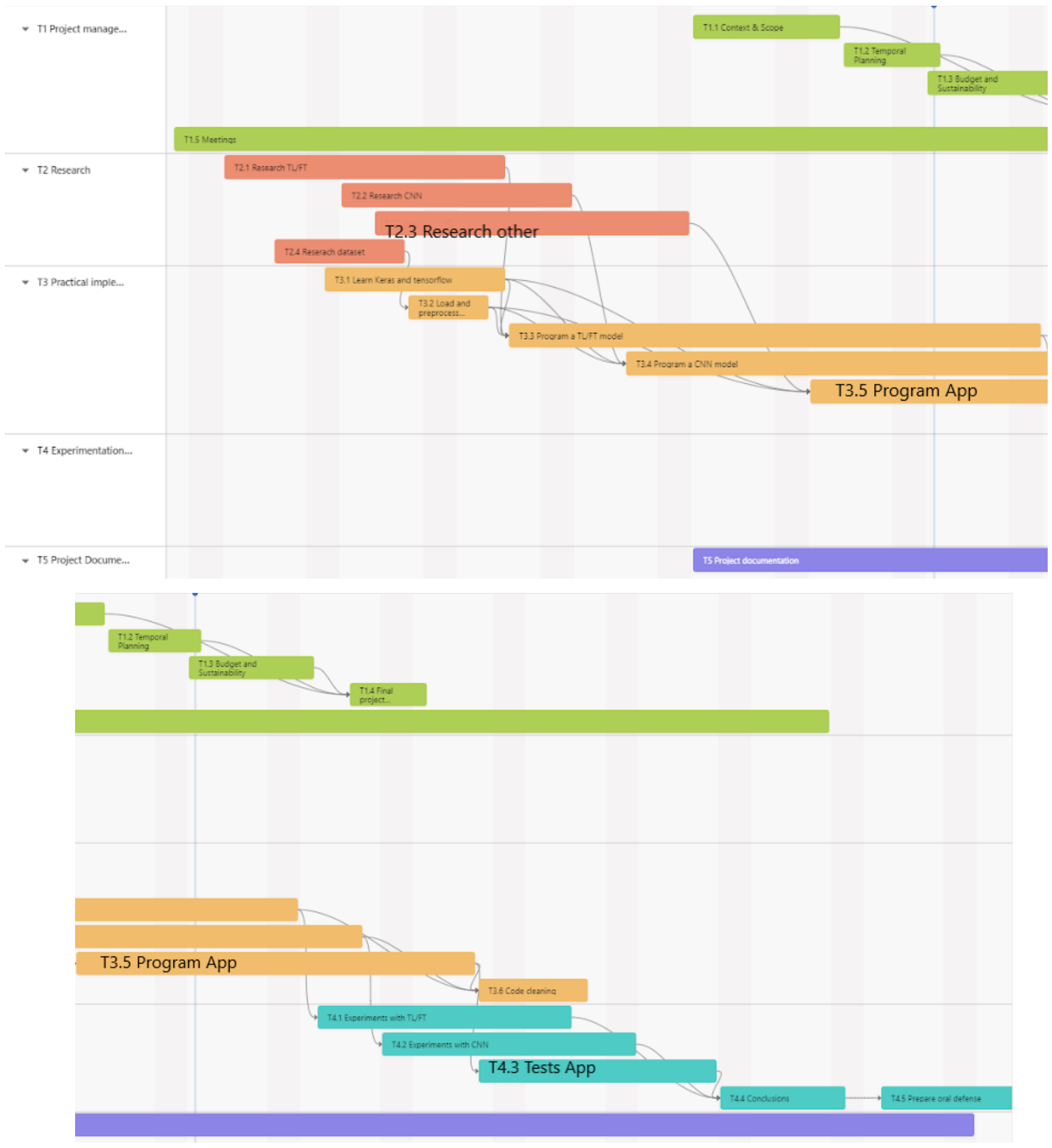
• Impact: Low
• Proposed solution: On one hand, more meetings with the tutor could be arranged in order to compensate for the lack of experience. On the other hand, more hours could be destined to research tasks and learning.
• Risk plan: Identify doubts and queries about the field -> Make a list about doubts -> Search online for answers -[if some doubts uncleared]-> set up a meeting with the tutor.

2.2.3 Technical issues

• Impact: High
• Proposed solution: Google Colab is a fundamental tool, if the project encounters technical issues with this programme it would be at risk. In order not to be so dependent on Google Colab, we could explore other alternatives like Kaggle Kernel or others. Other tools that could have technical issues are easily replaceable.

• Risk plan: Experience technical issues with Google Colab -> Decide if continuing with Google Colab or search for alternatives -[if search for alternatives]-> Dedicate 2-3 hours to research and evaluate Kaggle Kernel or others ->Migrate the project to the new platform.

2.3 Gantt



(2.2 Gantt diagram)

2.4 References 2

- *Racó UPC FIB*. (n.d.-b). https://raco.fib.upc.edu/projectes/estat.jsp/ *[13/06/2023]*
- *Atenea Campus Virtual UPC* (n.d.). https://atenea.upc.edu/ *[13/06/2023]*
- *Google Drive - Google*. (n.d.). https://www.google.com/intl/es/drive/ *[13/06/2023]*
- *Google Slides* (n.d.). https://docs.google.com/presentation/u/0/ *[13/06/2023]*
- *Google Docs* (n.d.). https://docs.google.com/document/u/0/?hl=es *[13/06/2023]*
- *Google Colaboratory*. (n.d.). https://colab.research.google.com/?hl=es *[13/06/2023]*
- *Kaggle: Your Machine Learning and Data Science Community. (n.d.). https://www.kaggle.com/ [13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

3. Budget and sustainability

3.1 Budget

3.1.1 Staff costs

To accurately estimate the personnel costs of a project, it is necessary to determine the specific roles that are required, their respective hourly wages, and how the tasks will be distributed among them. The amount that each worker will earn is calculated by multiplying their wage per hour by the number of hours they are expected to work on the project. Furthermore, the cost of each task is determined by adding up the wages of all the workers who are involved in completing it.
The roles required for this project are the following:
- **Project Manager**: Responsible for overseeing the project, managing resources, and ensuring that the project is completed on time and within budget.
- **Junior Machine Learning Engineer**: Responsible for developing and implementing machine learning algorithms and models.
- **Analyst**: Responsible for analyzing the results and drawing conclusions.
- **Tester**: Responsible for testing the code and verifying its correct functioning.
- **Technical Writer**: Responsible for documenting the project and creating user manuals and other technical documentation.

Role	Cost (€)/h
Project Manager	23
Junior Machine Learning Engineer	25
Analyst	13
Tester	8
Technical Writer	20

(3.1 Cost per hour of the different roles)

Tarea	Project Manager	Junior Machine Learning Engineer	Analyst	Tester	Technical Writer
T1	75	0	0	0	25
T2	0	125	0	0	0

T3	0	170	0	10	0
T4	10	80	25	0	0
T5	0	0	0	0	50

(3.2 Estimated time per task)

Tasks	**Cost(€)**
Project management	2325
Research	3750
Practical implementation	5875
Experimentation and conclusions	3275
Project documentation	1000

(3.3 Estimated cost per task)

Role	**Hours**	**Cost (€)**
Project Manager	85	1955
Junior Machine Learning Engineer	375	9375
Analyst	25	325
Tester	10	80
Technical Writer	75	1500
TOTAL	570	13235

(3.4 Total cost of the staff)

3.1.2 Generic costs

Amortization of the resources
The resource we have to calculate the amortization cost of is mainly my laptop (Lenovo Yoga 720) since all the software resources are open source.
To calculate the amortization cost of a Lenovo Yoga 720 laptop purchased for 900€ and used for 570 hours, we need to determine the depreciation rate first.
Depreciation rate = Cost of laptop / Expected total lifetime usage
The expected lifetime usage of the laptop is 5 years or 15,000 hours of usage.
Depreciation rate = 900€ / 15,000 hours = 0.06 €/hour
Now, we can calculate the amortization cost for the 570 hours of usage:
Amortization cost = Depreciation rate x Number of hours used
Amortization cost = 0.06 €/hour x 570 hours = 34.2 €
Therefore, the amortization cost of the Lenovo Yoga 720 laptop for 570 hours of usage is 34.2 €.

Indirect costs
- Internet cost: The internet cost is around 90e per month. Hence, the total cost would be 35€ /month * 5 months, 175€.
- Electricity cost: The kwH price[14] is 0.1199 e /kWh. Given that the power of an average desktop is 400 kwH, the total cost would be (0.10 € /kWh) * 0,88 kwH * 570 hours adds up to 50.16€
- Work space: The thesis will be developed at my home, located in Vilanova i la Geltrú. The cost of living in vilanova in an apartment similar to where I live is approximately 800€/month. In my apartment there lives another person so we can estimate that my cost per month is 400€ and the project duration is 5 months, thus the cost would add up to 2.000€.

In summary the total indirect costs would approximately be 2225.16€ .

3.1.3 Other costs

Incidental costs
In the following table it is represented the costs associated with each difficulty or incident this project may face.

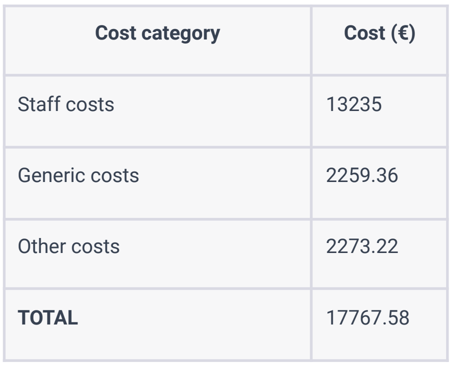
Incident	Estimated cost (€)	Risk(%)	Cost (€)
Deadline of the project	500	20	100
Lack of experience in the field	300	25	75
Technical issues	0	70	0
TOTAL	800	-	175

(3.5 Incidental costs)

Contingency
Every project can have unexpected issues, so it's important to be prepared. That's why we need a contingency fund to prevent delays in the project.
CPA (cost per activity): Due to the possibility of delays in the project and the need for additional hours, the contingency cost assigned to PCA will be 15%.
GC (general costs): Since this will be a 5 month project the probability of needing additional resources is low. Therefore the contingency cost assigned will be 5%.
The total contingency fund is 2098.22€

3.1.4 Total costs

The total costs of the project are displayed in a table, which provides a comprehensive overview of all expenses associated with the project.



Cost category	Cost (€)
Staff costs	13235
Generic costs	2259.36
Other costs	2273.22
TOTAL	17767.58

(3.6 Total costs of the project)

3.1.5 Management control

In order to detect cost deviations we establish some control mechanisms which indicate the deviation from the initial budget.
- Human resources deviation = (Estimated cost per hour - Real cost per hour) * Total hours consumed.
Amortization Deviation = (Estimated usage hours - Real usage hours) * Price per hour.
- Incidental cost deviation = (Estimated incidental hours - Real incidental hours) * Total incidental hours.

3.2 Sustainability

3.2.1 Self-assessment

The world is facing environmental problems due to human actions and we need to act quickly. We must consider the impact of projects on the environment, as well as their social and economic impact. In the Bachelor Final Project, students are encouraged to assess the sustainability of their work, including its environmental, social, and economic impact. This ensures that they consider the consequences of their work and whether it is worth implementing. Some students may not have thought about these issues before, but it's important to understand how their work affects different areas.

3.2.2 Economic dimension

Regarding PPP: Reflection on the cost you have estimated for the completion of the project:
The Budget section provides details about the expected costs of the project, including both human and material expenses. It also takes into account any potential unforeseen expenses that may arise during the project's execution.

Regarding life expectancy, how is it solved the problem you are trying to solve? Does your solution provide any improvement economically?
Currently, emotion recognition from facial images is mostly solved through manual annotation by humans, which is time-consuming and costly. By using deep learning techniques and the FER2013 dataset to apply transfer learning, CNNs, and VAEs, the process can be automated, resulting in significant cost savings for businesses and organizations that rely on emotion recognition technology.

3.2.3 Environmental dimension

Regarding PPP: Have you estimated the environmental impact of the project?
The use of machine learning techniques may require significant computing resources, which could have an impact on the environment through increased energy consumption and carbon emissions.

Regarding PPP: Did you plan to minimize its impact, for example, by reusing resources?
In order to minimize the impact we could optimize algorithms to reduce computing requirements and minimize waste and resource consumption throughout the project's lifecycle.

Regarding life expectancy, how does it solve the problem you are trying to solve? Does your solution provide any improvement in the environmental impact?
By automating the process of emotion recognition from facial images using deep learning techniques and the FER2013 dataset, the need for manual annotation by humans is

reduced, resulting in a lower carbon footprint and reduced environmental impact from the use of energy and resources required for manual annotation.

3.2.4 Social dimension

Regarding the PPP, how do you think this project will enrich you personally?
This project will help me gain hands-on experience in cutting-edge technologies and techniques. The project can be an opportunity to deepen my knowledge of machine learning and its applications, particularly in the field of image recognition.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)? How will your solution improve the quality of life (social dimension) with respect to other existing solutions?
Emotion recognition from facial images has a wide range of potential applications, from improving customer service to detecting emotional distress in patients. By using deep learning techniques and the FER2013 dataset to improve the accuracy of emotion recognition, the quality of life for individuals in various industries can be improved. For example, detecting emotional distress in patients can lead to earlier intervention and treatment, improving their overall well-being.

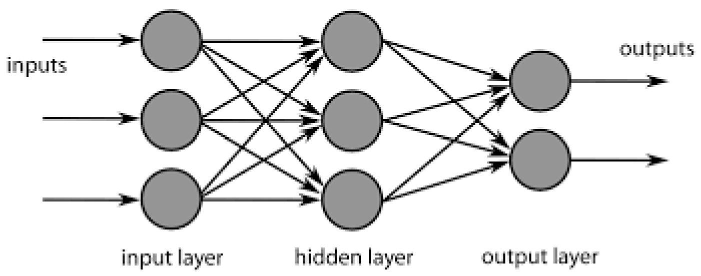
3.3 References 3

- *Racó UPC FIB*. (n.d.-b). https://raco.fib.upc.edu/projectes/estat.jsp/ *[13/06/2023]*
- *Atenea Campus Virtual UPC* (n.d.). https://atenea.upc.edu/ *[13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

4. Theoretical Background

4.1 Deep Learning and Neural Networks

Deep learning is a subfield of machine learning that focuses on using neural networks to solve intricate problems. Neural networks are based on the structure of the human brain and consist of interconnected nodes or neurons that process information and extract features from input data.
Neurons are the basic unit of neural networks and receive input signals from other neurons to produce an output signal that is transmitted to other neurons. The connections between neurons are modeled by weights, which are adjusted during training to optimize the network's performance.



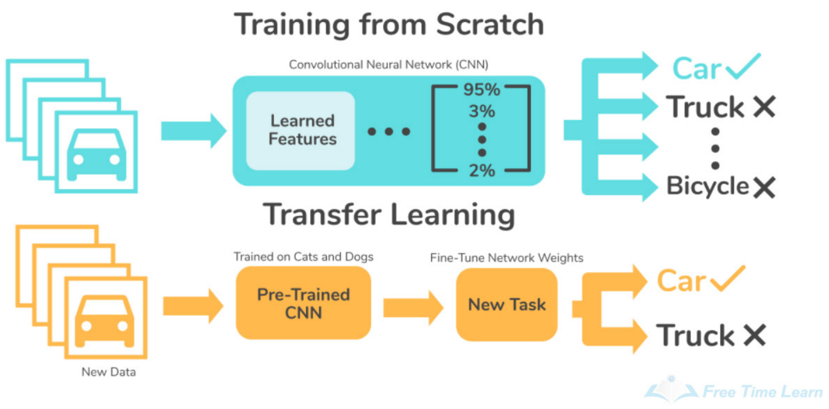
(4.1 Neural network conceptual image)

Deep neural networks are known for their depth, meaning they have several layers of neurons. This enables them to learn intricate representations of input data, leading to better performance on tasks like natural language processing and image recognition.
Training a neural network involves providing it with a substantial dataset of labeled examples and adjusting the weights of the connections between neurons to minimize a loss function that measures the difference between predicted and actual output. This process is known as backpropagation and uses gradient descent to update the weights in the network iteratively.
Deep learning has transformed several areas of artificial intelligence, including speech recognition, natural language processing, and computer vision. It has allowed machines to exceed human-level performance on various tasks, including game playing and image classification.
In the context of emotion recognition from facial images, deep learning has displayed encouraging results. Training a neural network on a vast labeled dataset of images can help learn features that indicate different emotions, including anger, happiness, and sadness. These features can then classify new images and estimate the subject's emotional state.

4.2 Transfer Learning and Fine Tuning (TL/FT)

Transfer learning and fine-tuning are techniques commonly used in deep learning to leverage existing pre-trained models for a new task.
Transfer learning involves using a pre-trained model on a large dataset as a starting point for a new model on a smaller dataset. By using the pre-trained model as a feature extractor, the new model can leverage the learned features for the new task. This is particularly useful

when the new dataset is small and does not contain enough data to train a deep neural network from scratch.
Fine-tuning involves taking a pre-trained model and continuing training on a new dataset. The pre-trained model is used as an initialization point for the new model, and the weights are fine-tuned on the new dataset to adapt the model to the new task. Fine-tuning is typically used when the new dataset is similar to the original dataset used to train the pre-trained model.

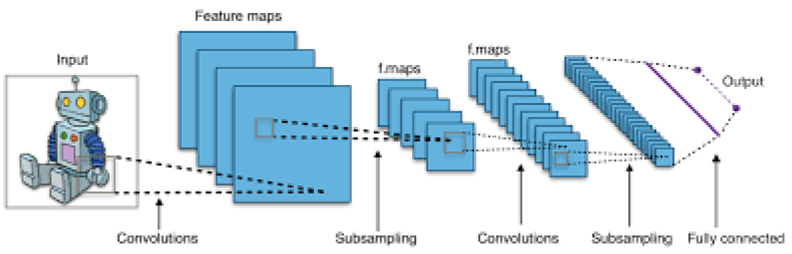


(4.2 Transfer learning visual example)

Both transfer learning and fine-tuning can lead to significant improvements in model performance, especially when the new task is related to the original task used to train the pre-trained model. Additionally, using transfer learning and fine-tuning can save significant time and resources compared to training a deep neural network from scratch on a new dataset.
In the context of emotion recognition from facial images, transfer learning and fine-tuning can be used to leverage pre-trained models for image classification tasks such as object recognition. By fine-tuning pre-trained models on a new dataset of labeled facial images, it is possible to achieve state-of-the-art results in emotion recognition tasks with significantly less training data and computation time.

4.3 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are a type of neural network that is particularly effective at processing image data. They have revolutionized computer vision tasks such as object recognition, face detection, and image classification.

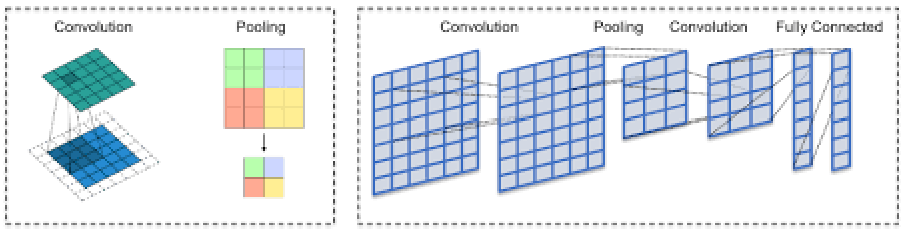


(4.3 Convolutional neural network conceptual image)

The architecture of a CNN consists of three main types of layers: convolutional layers, pooling layers, and fully connected layers. In a convolutional layer, the network performs a convolution operation on the input data using a set of filters or kernels. Each filter detects a specific feature in the image, such as edges, corners, or textures.

The output of the convolutional layer is then passed through a pooling layer, which reduces the spatial dimensions of the feature maps while retaining the most important information. This helps to reduce the computational complexity of the network and prevent overfitting.

Finally, the output of the pooling layer is flattened and passed through one or more fully connected layers, which perform the classification or regression task.



(4.4 Convolutional neural network structure image)

Training a CNN involves feeding it a large dataset of labeled images and optimizing the network's parameters to minimize a loss function that measures the difference between the predicted output and the true label. This is typically done using backpropagation and gradient descent.

CNNs have several advantages over traditional neural networks for image processing tasks. They can learn features that are invariant to translation, rotation, and scaling, making them robust to variations in the input data. They also use parameter sharing and sparsity to reduce the number of parameters and improve generalization performance.

In the context of emotion recognition from facial images, CNNs have shown promising results. By training a CNN on a large dataset of labeled images, it is possible to learn features that are indicative of different emotions, such as happiness, sadness, and anger. These features can then be used to classify new images and predict the emotional state of the subject.

Overall, CNNs are a powerful tool for image processing tasks and have transformed the field of computer vision. Their ability to automatically learn features from raw data has enabled them to achieve state-of-the-art performance on a wide range of tasks.

4.4 References 4

- AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference? (2022). *IBM*. https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks *[13/06/2023]*
- GeeksforGeeks. (2023). Neural Networks   A beginners guide. *GeeksforGeeks*. https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/ *[13/06/2023]*
- Hughes, C. (2022, February 26). Transfer Learning on Greyscale Images: How to Fine-Tune Pretrained Models on Black-and-White Datasets. *Medium*. https://towardsdatascience.com/transfer-learning-on-greyscale-images-how-to-fine-tune-pretrained-models-on-black-and-white-9a5150755c7a *[13/06/2023]*
- Alberto, R. (2022b, January 6). Redes Neuronales: ¿Qué es Transfer Learning y Fine Tuning? *Medium*. https://rubialesalberto.medium.com/redes-neuronales-qu%C3%A9-es-transfer-learning-y-fine-tuning-8259a81cfdbc *[13/06/2023]*
- Codificando Bits. (2019, March 23). *¿Qué son las REDES CONVOLUCIONALES?* [Video]. YouTube. https://www.youtube.com/watch?v=HyZFfBU0ADg *[13/06/2023]*
- *What are Convolutional Neural Networks? | IBM*. (n.d.). https://www.ibm.com/topics/convolutional-neural-networks *[13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

5. Dataset

5.1 Previous research and decision

Since in deep learning the dataset that feeds the model is essential for the learning process, it is vital to choose a suitable dataset to train it with. In this research articles and data science blog posts are particularly useful found in medium.com (online publishing platform for articles and blog posts on various topics) or more niche sites that are equally useful, such as Analytics India Magazine.
After the dataset research two datasets were seriously considered to be the main dataset for the thesis:

Google facial expression comparison dataset

- Description: This dataset is a large-scale facial expression dataset consisting of face image triplets along with human annotations that specify which two faces in each triplet form the most similar pair in terms of facial expression. Each triplet in this dataset was annotated by six or more human raters. This dataset is quite different from existing expression datasets that focus mainly on discrete emotion classification or action unit detection.
- Size: 156K face images and 200MB
- Link: https://research.google/resources/datasets/google-facial-expression/

FER2013 dataset

- Description: The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. Facial expressions are divided into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).
- Size: The training set consists of 28,709 examples and the public test set consists of 3,589 examples. The size of the files is 56.51 MB.
- Utility Link: https://www.kaggle.com/datasets/msambare/fer2013

FER2013 vs Google facial expression comparison dataset

Pros of using FER2013 dataset:

- FER2013 has a large number of labeled images of different emotions, making it a good dataset for training deep learning models.
- FER2013 is publicly available and easy to access.
- FER2013 requires minimal preprocessing, which reduces the time and effort required for data cleaning and preparation.

Cons of using FER2013 dataset:

- FER2013 is limited to a specific age group (18-60 years) and ethnicities, which may not be representative of the general population.

- FER2013 has a limited number of images for some emotions, which can affect the accuracy of the model for those emotions.

Pros of using Google Facial Expression Comparison dataset:

- Google's dataset has a larger number of images than FER2013, which can be beneficial for improving the accuracy of the model.
- Google's dataset has a wider age range and more diverse ethnicities, which can make the model more robust and representative of the general population.

Cons of using Google Facial Expression Comparison dataset:

- Google dataset requires significant preprocessing to extract relevant facial features, which can be time-consuming and require specialized expertise.
- Google dataset is not publicly available and requires permission from Google for its use.

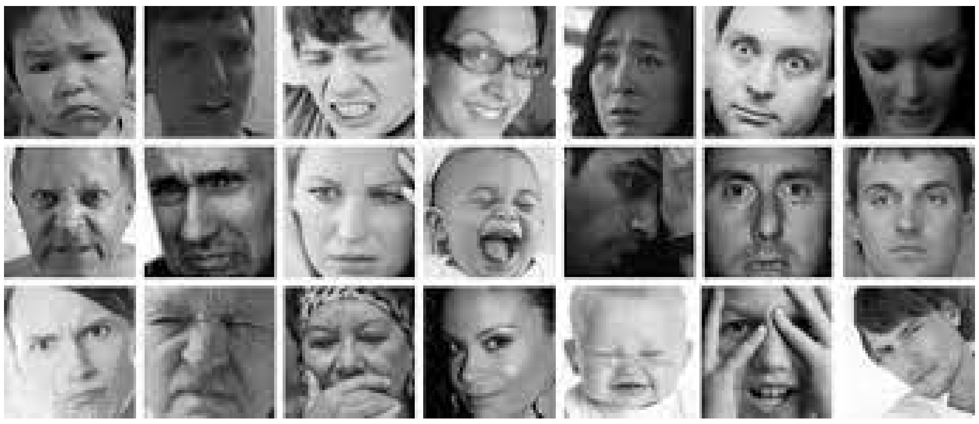
Final decision

The final decision is to use as the main dataset the FER2013 dataset. Although it has fewer images than the Google Facial Expression Comparison dataset, FER2013 has enough to train a model and achieve good accuracy in recognizing emotions. The most decisive factor when choosing has been ease of use, as FER2013 is publicly available unlike the Google dataset, and requires minimal preprocessing. Since this project specifically focuses on deep learning rather than data science as a whole, a dataset that can avoid delays and potential problems with data preprocessing is more useful to us than one that requires a lot of preprocessing.

5.2 FER2013 dataset

5.2.1 Basic review

The FER2013 dataset is a widely used dataset in the field of computer vision, specifically in the task of facial expression recognition.



(5.1 FER2013 dataset image example)

Here is a basic overview of the FER2013 dataset:

- Creator: The FER2013 dataset was created by Pierre-Luc Carrier and Aaron Courville, researchers from the Université de Montréal, Canada. The dataset was developed for their research on facial expression recognition.
- Creation Date: The FER2013 dataset was released in 2013.
- Characteristics: The FER2013 dataset consists of grayscale images of facial expressions. It contains a total of 35,887 images, which are divided into three subsets: training, validation, and testing. The distribution of images across these subsets is as follows: 28,709 images for training, 3,589 images for validation, and 3,589 images for testing.
- Image Size and Format: Each image in the FER2013 dataset has a resolution of 48x48 pixels. The images are provided in the Portable Network Graphics (PNG) format.
- Labels: The FER2013 dataset includes seven different facial expression labels: anger, disgust, fear, happiness, sadness, surprise, and neutral. Each image is labeled with one of these seven emotion categories.

5.2.2 Limitations

1. Data imbalance: It is important to note that the FER2013 dataset suffers from class imbalance, meaning that some emotion categories may have a significantly larger number of examples than others. This data imbalance can affect the training and evaluation of models, particularly for emotion categories with fewer samples.
2. Limited representation of certain emotions: The imbalance in the FER2013 dataset can result in limited representation of certain emotions, particularly those with fewer samples. Emotion categories that are less frequently encountered in the dataset may not be adequately learned by the model, leading to lower accuracy and potentially poorer generalization to real-world scenarios.
3. Generalizability: The FER2013 dataset may have limitations in terms of generalizability to diverse demographics and cultural contexts. The dataset was collected using images primarily from the internet, which may introduce biases and variations in facial expressions across different populations. The lack of diversity in the dataset can limit the model's ability to generalize well to real-world scenarios and individuals from different backgrounds.
4. Limited image resolution and grayscale format: The FER2013 dataset provides grayscale images with a resolution of 48x48 pixels. This limited resolution and absence of color information may impact the model's ability to capture fine-grained facial features and subtle variations in expressions. It may also restrict the model's performance in scenarios where color cues are important for emotion recognition.

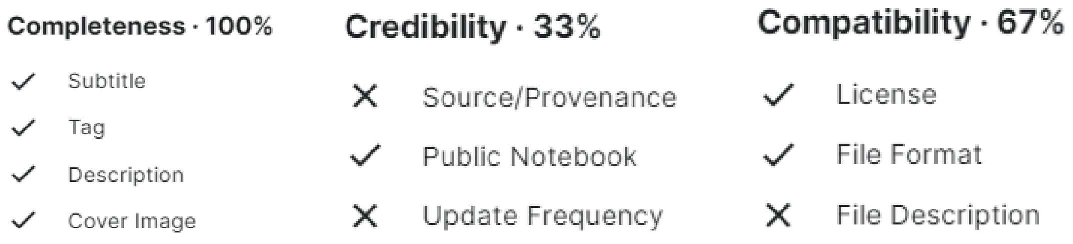
5.3 Data source and acquisition

5.3.1 Data source

Kaggle is an online platform where people interested in data science can find and explore datasets, participate in competitions, and collaborate with others. It's a place to learn and apply data science skills, solve real-world problems, and share ideas and code. Kaggle provides access to a wide range of datasets and tools for analyzing and modeling data. For the development of this thesis the data is obtained from the following kaggle:

The FER-2013 Kaggle dataset, created by msambare in 2020, is licensed under the Database Contents License (DbCL) v1.0.
The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

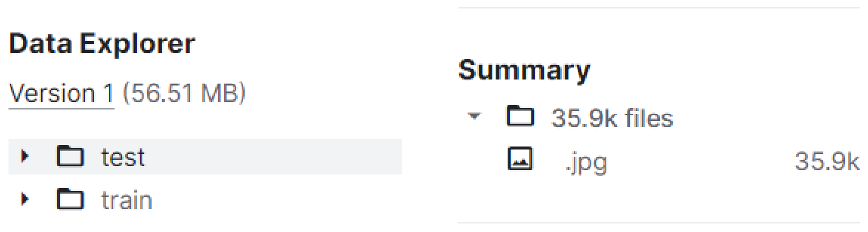
This kaggle has a usability score of 7.5, indicating its overall usability. The dataset has a completeness score of 100%, providing a subtitle, tags, description, and cover image for effective understanding and organization of the data.
In terms of credibility, the dataset scores 33%, lacking source/provenance information. However, it offers the availability of a public notebook, which adds to its credibility. The update frequency is not specified.
Regarding compatibility, the dataset scores 67%. It provides a clear license for usage and ensures compatibility with the file format. However, a detailed file description is not provided.



Completeness · 100%	Credibility · 33%	Compatibility · 67%
✓ Subtitle	✗ Source/Provenance	✓ License
✓ Tag	✓ Public Notebook	✓ File Format
✓ Description	✗ Update Frequency	✗ File Description
✓ Cover Image		

(5.2 Usability score description)

The dataset available on Kaggle has a total size of 56.51MB and consists of 35.9k files in the .jpg format. These files are primarily images that make up the dataset. The size of the dataset indicates the amount of data available for analysis and model training.



(5.3 Data size and type specifications)

Overall, the FER-2013 Kaggle dataset offers a high level of completeness and usability, with room for improvement in terms of credibility and file description. Researchers and practitioners can leverage this dataset for facial emotion recognition tasks while considering its limitations.

5.3.2 Data loading

The "opendatasets" library is a Python library that provides a convenient interface for downloading and working with datasets from various sources. It simplifies the process of accessing and using datasets for data analysis, machine learning, and other data-related tasks.
In order to download the fer2013 dataset the opentadasets library is imported and with the method .download() we can download the dataset from kaggle by simply passing the kaggle link as an argument to this method. When executing this code, Kaggle will prompt us to enter our username and associated key to download the data and store it in the Colab notebook's files. The data is saved in the "fer2013" folder. It is worth noting that this process needs to be repeated in each session because this type of file in Colab is not saved when the session ends.
The code then defines two variables: "train_path" and "val_path". These variables store the paths where the training and validation data will be located within the downloaded dataset. In this case, the training data is located in the "fer2013/train/" directory, and the validation data is located in the "fer2013/test/" directory.

5.4 Preprocessing

5.4.1 Image data generator

The ImageDataGenerator is a method in Keras. It is specifically designed for image data preprocessing and augmentation. The ImageDataGenerator allows us to generate augmented images in real-time while training our deep learning models.
This method provides various options to apply transformations to the input images, such as rotation, zooming, shifting, shearing, and flipping. It also supports resizing, rescaling, and normalizing the images.
By using the ImageDataGenerator, we can efficiently create data generators that automatically load and preprocess batches of images during training. This helps in enhancing the diversity of the training data and preventing overfitting.

5.4.2 Rescaling

Rescaling is a preprocessing technique commonly used in image data augmentation to normalize the pixel values of images. It involves scaling down or up the pixel values to a specific range. In the given code, the rescaling is implemented using the rescale parameter of the ImageDataGenerator method.
The ImageDataGenerator class instances have the rescale parameter set to 1/255, which means that each pixel value in the images will be divided by 255. This rescaling operation effectively scales down the pixel values from the original range of 0-255 to the normalized range of 0-1.
This normalization step helps in stabilizing the training process and improving the convergence of the neural network model. It also helps in reducing the impact of lighting conditions and color variations in the images.

5.4.3 Data augmentation

In the training ImageDataGenerator class instance the following data augmentation techniques are implemented:

- Rotation Range: It randomly rotates the images by a specified angle within the range of -20 to +20 degrees. This helps the model become invariant to different orientations of objects in the images.
- Width Shift Range: It randomly shifts the images horizontally (left or right) by a fraction of the total width. The value of 0.2 indicates that the images can be horizontally shifted up to 20% of the total width. This augmentation helps the model learn robustness to slight variations in object positions.
- Height Shift Range: Similar to width shift, it randomly shifts the images vertically (up or down) by a fraction of the total height. Again, the value of 0.2 means the images can be vertically shifted up to 20% of the total height. This augmentation aids in capturing the variability of object positions.
- Shear Range: It randomly applies shear transformations to the images within the range of -20 to +20 degrees. Shearing distorts the shape of the objects in the image while preserving their orientation. This augmentation helps the model learn to recognize objects despite deformations.
- Zoom Range: It randomly zooms into the images by a factor within the range of 0.8 to 1.2. Zooming alters the scale of objects in the image, allowing the model to handle different object sizes and improve its generalization capability.
- Horizontal Flip: It randomly flips the images horizontally. This augmentation introduces variations in the object's left-right orientation and helps the model learn to recognize objects from different viewpoints.

These data augmentation techniques increase the diversity of the training data by creating variations of the original images. This can help the model generalize better and improve its performance on unseen images.

5.4.4 Data split

In the code, two instances of the ImageDataGenerator class are created, train_datagen and val_datagen.

The flow_from_directory method is then used to generate data batches from the directory specified by train_path and val_path.
The train_path variable represents the path to the training data folder, which contains subfolders representing different classes or categories of data. Each subfolder contains images belonging to that specific class. Similarly, the val_path variable represents the path to the validation data folder, which follows the same structure as the training data folder.
By using the flow_from_directory method with the appropriate parameters, the training and validation data are loaded from their respective directories, and batches of augmented and rescaled images are generated for training and evaluation purposes.

5.4.5 Preprocessing for Transfer Learning

In transfer learning and fine-tuning scenarios, the adapt2ImageNetFormat function has been created.
This function is added to the preprocessing field when creating the ImageDataGenerator. Its purpose is to adapt the images to the pre-trained models that are trained on RGB images of size 224x224. If the images are not adapted, training would result in an error due to the structure of the pre-trained transfer learning models.

5.5 References 5

- Mahendrakumaran, K. (2021, December 28). Emotion Recognition Datasets - Analytics Vidhya - Medium. *Medium*. https://medium.com/analytics-vidhya/emotion-recognition-datasets-8a397590c7d1 *[13/06/2023]*
- Choudhury, A. (2020). Top 8 Datasets Available For Emotion Detection. *Analytics India Magazine*. https://analyticsindiamag.com/top-8-datasets-available-for-emotion-detection/ *[13/06/2023]*
- Google Research. (n.d.). *Google facial expression comparison dataset – Google Research*. https://research.google/resources/datasets/google-facial-expression/ *[13/06/2023]*

- *FER-2013*. (2020c, July 19). Kaggle. https://www.kaggle.com/datasets/msambare/fer2013
- *Database Contents License (DbCL) v1.0 — Open Data Commons: legal tools for open data*. (n.d.). https://opendatacommons.org/licenses/dbcl/1-0/ *[13/06/2023]*
- *opendatasets*. (2022, April 4). PyPI. https://pypi.org/project/opendatasets/ *[13/06/2023]*
- Team, K. (n.d.-c). *Keras documentation: Image data loading*. https://keras.io/api/data_loading/image/ *[13/06/2023]*
- *Bhandari, A. (2023). Image Augmentation on the fly using Keras ImageDataGenerator! Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/ [13/06/2023]*
- *Rosebrock, A. (2023, June 8). Keras ImageDataGenerator and Data Augmentation - PyImageSearch. PyImageSearch. https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/ [13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

6. Transfer Learning and Fine tuning

6.1 Overview

In this section, we provide an overview of the experimentation plan undertaken in this work. The goal was to explore the effectiveness of transfer learning and fine-tuning techniques in the context of our specific problem. The plan involved training and evaluating five different pre-trained models using a limited number of epochs with the available data. Based on the evaluation results, the best-performing model would be selected for further fine-tuning.
The experimentation plan can be summarized as follows:

1. Model Selection: We identified five pre-trained models for transfer learning: MobileNet V2, Inception V3, VGG16, ResNet, and DenseNet. These models were chosen based on their popularity and success in various computer vision tasks.
2. Model Implementation: Each selected model was implemented using the respective pre-trained weights available in TensorFlow Hub or Keras Applications. We utilized the Keras framework for model implementation and customization.
3. Model Training: The selected models were trained using the available dataset for a limited number of epochs. This allowed us to evaluate their performance and identify the most promising model for further experimentation.
4. Model Evaluation: After training, each model was evaluated using a validation dataset. Metrics such as accuracy, loss, and confusion matrix were used to assess their performance. The evaluation results were analyzed to determine the model that achieved the highest performance.
5. Fine-tuning: Based on the evaluation results, the model with the best performance was selected for fine-tuning. This decision was made considering factors such as accuracy, generalization ability, and computational efficiency.

6.2 TL Models

6.2.1 Model overviews

MobileNet V2

MobileNet V2 is a lightweight convolutional neural network architecture designed specifically for mobile and embedded vision applications. It aims to provide an efficient and compact solution for image classification tasks. Some key aspects of MobileNet V2 are as follows:
Brief Description:
MobileNet V2 employs depthwise separable convolutions, which split the standard convolutional layer into a depthwise convolution and a pointwise convolution. This separation significantly reduces the computational complexity and number of parameters while maintaining a reasonable level of accuracy. Additionally, MobileNet V2 introduces inverted residual blocks, linear bottlenecks, and a width multiplier parameter to optimize model efficiency.
Strengths:

- Efficiency: MobileNet V2's depth wise separable convolutions enable the model to achieve a good balance between accuracy and efficiency. It significantly reduces the computational load, making it suitable for resource-constrained environments.

- Compactness: The architecture of MobileNet V2 is designed to be compact, allowing it to be easily deployed on mobile and embedded devices with limited memory and processing power.
- Flexibility: The width multiplier parameter in MobileNet V2 offers flexibility in trading off model size and performance, allowing customization based on specific resource constraints.

Weaknesses:

- Reduced capacity: Due to the model's emphasis on efficiency and compactness, MobileNet V2 may have a lower capacity to capture complex patterns compared to larger and more computationally expensive models.
- Limited representation power: The reduced model size and complexity may limit MobileNet V2's ability to capture fine-grained details or handle more complex tasks that require a higher level of representation.

Justification:
MobileNet V2 strikes a balance between model size, efficiency, and accuracy, making it particularly useful for emotion classification tasks. Emotion classification primarily relies on capturing high-level features and patterns in images, rather than intricate details. MobileNet V2's efficient architecture and lightweight design make it suitable for deploying on mobile and embedded devices, where real-time emotion classification is often desired. Its compactness allows for faster inference times, making it an excellent choice for emotion analysis in scenarios where computational resources are limited.

Inception V3
Inception V3 is a convolutional neural network architecture known for its effectiveness in image classification tasks. It was developed to improve upon the limitations of earlier models by addressing the challenges of depth, computational efficiency, and feature representation. Here's an overview of Inception V3:
Brief Description:
Inception V3 utilizes a combination of convolutional layers with different filter sizes and pooling operations to capture features at different scales. It incorporates the concept of "Inception modules," which are responsible for extracting features at multiple levels of abstraction. These modules consist of parallel convolutional layers with different filter sizes and utilize 1x1 convolutions to reduce computational complexity. Inception V3 also employs advanced techniques such as batch normalization and auxiliary classifiers to enhance training and regularization.
Strengths:

- Effective feature extraction: The Inception V3 architecture with its multiple filter sizes and pooling operations enables the model to capture features at different levels of abstraction, improving its capability to represent complex patterns.
- Computational efficiency: The utilization of 1x1 convolutions and the inception modules allows Inception V3 to achieve a good balance between accuracy and computational efficiency.
- Regularization techniques: The inclusion of auxiliary classifiers during training helps in mitigating the vanishing gradient problem and aids in regularizing the model.

Weaknesses:

- Complexity: Inception V3 has a more complex architecture compared to simpler models, which may result in increased computational requirements during training and inference.

- Higher memory footprint: The increased number of parameters and layers in Inception V3 may require more memory resources during training and deployment.

Justification:
Inception V3's strong feature extraction capabilities and computational efficiency make it a suitable choice for emotion classification tasks. Emotion classification often relies on capturing both low-level and high-level features, and Inception V3's multi-scale feature extraction enables it to effectively represent such features. Its ability to handle complex patterns and its regularization techniques contribute to its success in classifying emotions accurately. While Inception V3 may require higher computational resources compared to simpler models, its performance justifies its applicability to emotion classification tasks where accuracy and robust feature representation are crucial.

VGG16
Brief Description:
VGG16 (Visual Geometry Group 16) is a deep convolutional neural network architecture developed by the Visual Geometry Group at the University of Oxford. It was introduced as part of the ILSVRC 2014 competition and has gained popularity due to its simplicity and effectiveness in image classification tasks.
Strengths:

- Deep architecture: VGG16 consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. Its deep structure enables it to learn hierarchical representations of images, capturing both low-level and high-level features.
- Simplicity: VGG16 has a straightforward and uniform architecture, making it easy to understand and implement. Each layer has a fixed configuration of convolutional filters and pooling operations, contributing to its simplicity.

Weaknesses:

- Computational requirements: The deep structure of VGG16 results in a large number of parameters, making it computationally expensive to train and deploy. It requires significant computational resources and may have memory limitations.
- Overfitting potential: VGG16's large number of parameters increases the risk of overfitting, especially when training on smaller datasets. Adequate regularization techniques are necessary to prevent overfitting.

Justification:
VGG16's strength lies in its ability to learn hierarchical representations of images, capturing both low-level and high-level features. Emotion classification tasks often rely on extracting relevant features from images, and VGG16's deep architecture allows it to effectively capture such features. While VGG16 may require more computational resources compared to simpler models, its performance justifies its applicability to emotion classification tasks where accuracy and robust feature representation are crucial. By leveraging transfer learning and fine-tuning, the pre-trained VGG16 model can be further customized and trained on emotion-specific datasets, leading to improved classification results.

ResNet
Brief Description:
ResNet (Residual Neural Network) is a deep convolutional neural network architecture that revolutionized image classification by introducing residual connections. It was introduced by Microsoft Research in 2015 and has been widely adopted in various computer vision tasks.

Strengths:

- Residual connections: ResNet introduced the concept of residual connections, which allow for the direct propagation of information from earlier layers to later layers. This helps address the vanishing gradient problem and enables the training of much deeper networks.
- Deep architecture: ResNet architectures typically consist of tens or even hundreds of layers, allowing them to learn highly complex and abstract representations of images.
- State-of-the-art performance: ResNet has achieved top performance in various image classification benchmarks, demonstrating its effectiveness in capturing and representing intricate image features.

Weaknesses:

- Computational complexity: The deep architecture of ResNet with its numerous layers and residual connections requires substantial computational resources for both training and inference. Training large ResNet models can be time-consuming.
- Memory requirements: ResNet models have a large number of parameters, resulting in a higher memory footprint during training and inference. This can be a limitation when working with resource-constrained environments.

Justification:
ResNet's strength lies in its ability to effectively capture and represent complex image features through its deep architecture and residual connections. Emotion classification tasks often require models that can extract intricate patterns from images, making ResNet a suitable choice. The residual connections alleviate the vanishing gradient problem and enable the successful training of very deep networks. While ResNet may be computationally demanding, its state-of-the-art performance justifies its applicability to emotion classification tasks where accuracy and feature representation are critical. By leveraging transfer learning and fine-tuning, the pre-trained ResNet model can be customized and trained on emotion-specific datasets, further enhancing its performance in emotion classification.

DenseNet

Brief Description:
DenseNet is a convolutional neural network architecture that is known for its dense connections between layers. It was introduced by researchers at Facebook AI Research in 2016 and has gained popularity for its efficient use of parameters and strong feature propagation.

Strengths:

- Dense connections: DenseNet introduces dense connections, where each layer is connected to every subsequent layer in a feed-forward manner. This dense connectivity enhances feature reuse and information flow, allowing the network to efficiently propagate gradients and capture fine-grained details.
- Parameter efficiency: DenseNet's dense connections enable parameter sharing, reducing the number of required parameters compared to traditional architectures. This parameter efficiency makes DenseNet models more memory-efficient and computationally faster.
- Feature propagation: DenseNet facilitates the propagation of features through the network by concatenating the feature maps from all preceding layers. This enables the model to have a deeper understanding of the data and capture complex patterns.

Weaknesses:

- Memory requirements: DenseNet models tend to have a larger memory footprint compared to some other architectures due to the concatenation of feature maps from all preceding layers. This can be a limitation when working with resource-constrained environments.
- Computational complexity: The dense connectivity pattern in DenseNet can result in increased computational complexity during training and inference, especially in deeper models.

Justification:
DenseNet's strengths in efficient parameter usage, feature propagation, and strong gradient flow make it well-suited for emotion classification tasks. Emotion classification often requires models that can capture both global and local features in an image, and DenseNet's dense connections enable effective feature reuse and propagation, facilitating the extraction of relevant emotional cues. Despite potential memory and computational requirements, DenseNet's parameter efficiency and feature-rich representations justify its applicability to emotion classification tasks where accuracy and robust feature extraction are essential. By leveraging transfer learning and fine-tuning, the pre-trained DenseNet model can be adapted and trained on emotion-specific datasets, further enhancing its performance in emotion classification.

6.2.2 Model implementations

MobileNet V2
The MobileNet V2 implementation follows these steps:

1. Import the pre-trained MobileNet V2 model using the hub.KerasLayer from TensorFlow Hub. This allows us to utilize the pre-trained model for feature extraction. The model is loaded from the TensorFlow Hub with the specified input shape of (224, 224, 3).
2. Freeze the layers of the pre-trained model by setting mobilenet_v2.trainable = False. This ensures that the pre-trained weights are not updated during training, preserving the learned features.
3. Create a new sequential model (tf.keras.Sequential) and add the MobileNet V2 model as the first layer. Then, add a dense layer with the number of units equal to the number of classes for the desired task. In this case, the activation function is set to softmax, and a kernel regularizer is applied to the dense layer for regularization.
4. Compile the model using the Adam optimizer and categorical cross-entropy loss function. The chosen metrics for evaluation are accuracy.
5. Train the model by fitting it to the training dataset (train_ds) and validating it on the validation dataset (val_ds) for the specified number of epochs.
6. Measure the training time by recording the start time before training and end time after training. The elapsed time is calculated as the difference between the start and end times.
7. Optionally, save the trained model using model_mobilenet_v2.save("model_mobilenet_v2.h5") for future use or deployment.

Inception V3
The InceptionV3 implementation follows these steps:

1. Load the pre-trained InceptionV3 model from Keras Applications using the InceptionV3 class. The model is initialized with pre-trained weights from the ImageNet dataset. The include_top argument is set to False to exclude the fully connected layers at the top of the network. The specified input shape is (224, 224, 3).
2. Freeze the layers of the base model to prevent them from being updated during training. This is achieved by iterating over each layer in the InceptionV3 model and setting the trainable attribute to False.
3. Add a GlobalAveragePooling2D layer to reduce the dimensionality of the output data from the base model. This layer aggregates the spatial information and calculates the average values across each channel.
4. Add a dense layer with 1024 units and a ReLU activation function on top of the pooling layer. This layer serves as a feature extractor, capturing higher-level representations from the pooled features.
5. Add a final dense layer with the number of classes in the dataset and a softmax activation function for multi-class classification. This layer produces the predicted probabilities for each class.
6. Create a new Keras model using the input and output tensors of the InceptionV3 model. Compile the model using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric.
7. Measure the training time by recording the start time before training and end time after training. The elapsed time is calculated as the difference between the start and end times.
8. Optionally, save the trained model using model_inception_v3.save("model_inception_v3.h5") for future use or deployment.

VGG16

The VGG16 implementation follows these steps:

1. Load the pre-trained VGG16 model from Keras Applications using the VGG16 class. The model is initialized with pre-trained weights from the ImageNet dataset. The include_top argument is set to False to exclude the fully connected layers at the top of the network. The specified input shape is (224, 224, 3).
2. Freeze the layers of the base model to prevent them from being updated during training. This is achieved by iterating over each layer in the VGG16 model and setting the trainable attribute to False.
3. Add a GlobalAveragePooling2D layer to reduce the dimensionality of the output data from the base model. This layer aggregates the spatial information and calculates the average values across each channel.
4. Add a dense layer with 1024 units and a ReLU activation function on top of the pooling layer. This layer serves as a feature extractor, capturing higher-level representations from the pooled features.
5. Add a final dense layer with the number of classes in the dataset and a softmax activation function for multi-class classification. This layer produces the predicted probabilities for each class.
6. Create a new Keras model using the input and output tensors of the VGG16 model. Compile the model using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric.

7. Measure the training time by recording the start time before training and end time after training. The elapsed time is calculated as the difference between the start and end times.
8. Optionally, save the trained model using model_vgg16.save("model_vgg16.h5") for future use or deployment.

ResNet

The ResNet implementation follows these steps:

1. Load the pre-trained ResNet50 model from Keras Applications using the ResNet50 class. The model is initialized with pre-trained weights from the ImageNet dataset. The include_top argument is set to False to exclude the fully connected layers at the top of the network. The specified input shape is (224, 224, 3).
2. Freeze the layers of the base model to prevent them from being updated during training. This is achieved by iterating over each layer in the ResNet50 model and setting the trainable attribute to False.
3. Add a GlobalAveragePooling2D layer to reduce the dimensionality of the output data from the base model. This layer aggregates the spatial information and calculates the average values across each channel.
4. Add a dense layer with 1024 units and a ReLU activation function on top of the pooling layer. This layer serves as a feature extractor, capturing higher-level representations from the pooled features.
5. Add a final dense layer with the number of classes in the dataset and a softmax activation function for multi-class classification. This layer produces the predicted probabilities for each class.
6. Create a new Keras model using the input and output tensors of the ResNet50 model. Compile the model using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric.
7. Measure the training time by recording the start time before training and end time after training. The elapsed time is calculated as the difference between the start and end times.
8. Optionally, save the trained model using model_resnet.save("model_resnet.h5") for future use or deployment.

DenseNet

The DenseNet implementation follows these steps:

1. Load the pre-trained DenseNet121 model from Keras Applications using the DenseNet121 class. The model is initialized with pre-trained weights from the ImageNet dataset. The include_top argument is set to False to exclude the fully connected layers at the top of the network. The specified input shape is (224, 224, 3).
2. Freeze the layers of the base model to prevent them from being updated during training. This is achieved by iterating over each layer in the DenseNet121 model and setting the trainable attribute to False.
3. Add a GlobalAveragePooling2D layer to reduce the dimensionality of the output data from the base model. This layer aggregates the spatial information and calculates the average values across each channel.

4. Add a dense layer with 1024 units and a ReLU activation function on top of the pooling layer. This layer serves as a feature extractor, capturing higher-level representations from the pooled features.
5. Add a final dense layer with the number of classes in the dataset and a softmax activation function for multi-class classification. This layer produces the predicted probabilities for each class.
6. Create a new Keras model using the input and output tensors of the DenseNet121 model. Compile the model using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric.
7. Measure the training time by recording the start time before training and end time after training. The elapsed time is calculated as the difference between the start and end times.
8. Optionally, save the trained model using model_densenet.save("model_densenet.h5") for future use or deployment.

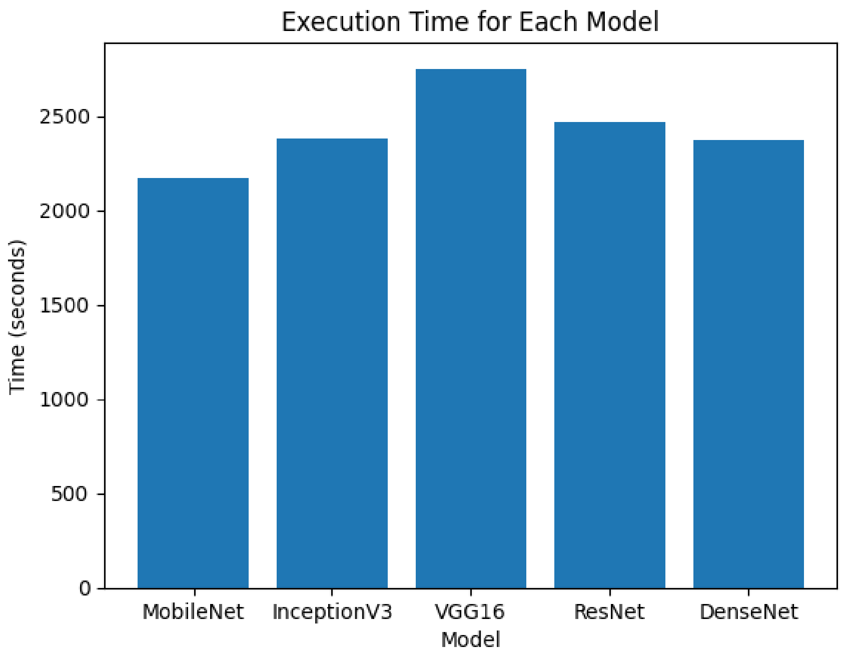
6.2.3 Model training

The five models, namely MobileNet, InceptionV3, VGG16, ResNet, and DenseNet, were trained under the same conditions to evaluate their performance for emotion classification. The training settings, including the learning rate, optimizer, and number of epochs, were kept consistent across all models.
To ensure fairness and comparability, a learning rate of 0.001 was used for all models during training. The Adam optimizer was employed, which is known for its effectiveness in handling large datasets and diverse architectures. This optimizer adjusts the learning rate adaptively during training, allowing the models to converge efficiently.
Regarding the number of epochs, it was determined that training the models for 5 epochs would strike a balance between computational limitations and capturing meaningful patterns in the data. Initially, longer training periods, such as 10 or 20 epochs, were considered. However, due to computational constraints and the extensive training time required, it was decided that five epochs would provide sufficient insights into each model's performance. Training the models for five epochs allowed for a reasonable amount of learning to take place while avoiding excessive training time. It was observed through preliminary experiments that the models achieved noticeable convergence and demonstrated their ability to learn essential features within this timeframe. Therefore, five epochs were chosen as a practical compromise, ensuring a balance between model performance and training time constraints.
By training the models under the same conditions, including learning rate, optimizer, and a consistent number of five epochs, we can effectively compare their performance and determine which model best suits the emotion classification task based on the evaluation metrics.

6.2.4 Model results

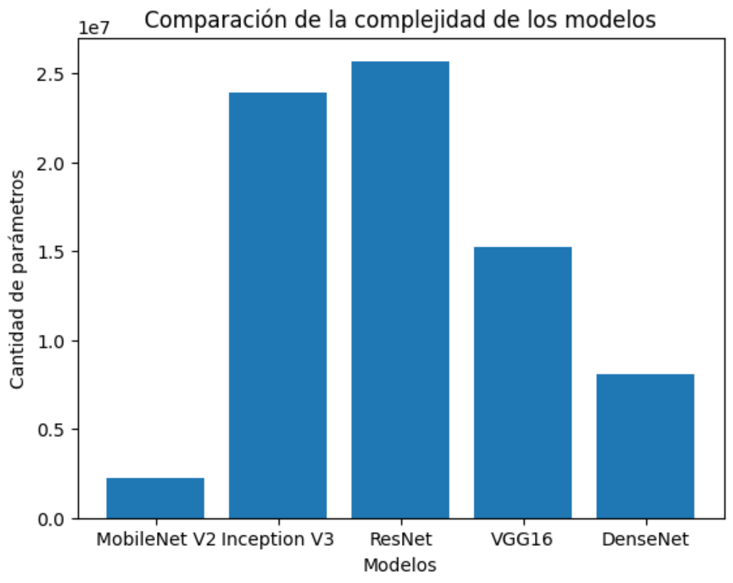
Training time
The first metric to compare among the models is the training time in seconds.

As can be seen in the following histogram, the training time for the 5 models for 5 epochs is very similar and ranges between 2000 and 2500 seconds, averaging between 400 and 500 seconds per epoch.



(6.1 TL training time histogram)

Model complexity (number of parameters)
Complexity is also a crucial characteristic of a model and here is a histogram comparing the number of parameters of each model:



(6.2 TL complexity histogram)

Among these models, MobileNet has the fewest number of parameters with 2,266,951, making it a relatively lightweight model. On the other hand, ResNet has the highest number of parameters with 25,693,063, indicating a more complex and higher-capacity model. InceptionV3 and VGG16 have intermediate numbers of parameters, with 23,908,135 and 15,247,175, respectively. DenseNet falls between the extremes with 8,094,279 parameters.

MobileNet stands out as a lightweight option suitable for scenarios with limited computational resources or when faster inference speed is desired. On the other hand, ResNet and VGG16 offer a higher capacity model with more parameters, making it more suitable for tasks that require capturing complex patterns. InceptionV3, and DenseNet fall within the moderate range of parameter numbers, providing a balance between capacity and efficiency.

Model performance (accuracy and loss)
Finally we must compare the accuracy and loss of the models since its the most relevant aspect of a model.
The accuracy and loss results of the 5 transfer learning models, MobileNet, InceptionV3, VGG16, ResNet, and DenseNet, are as follows:

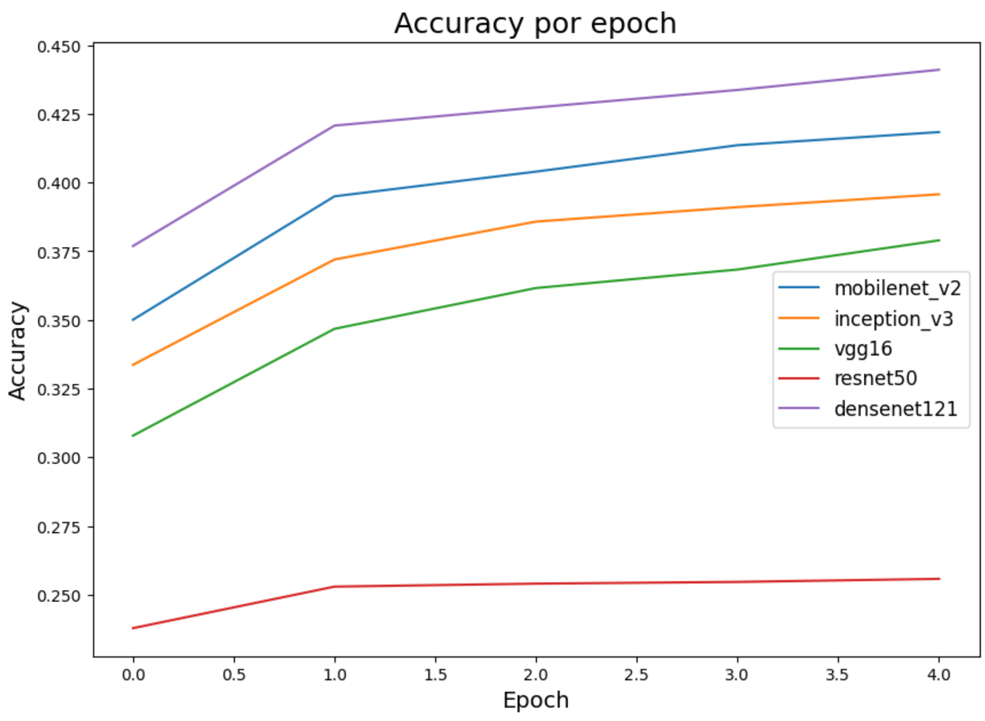
1. MobileNet:
    - Validation Loss: 1.5182
    - Validation Accuracy: 0.4232

2. InceptionV3:
    - Validation Loss: 1.4381
    - Validation Accuracy: 0.4409

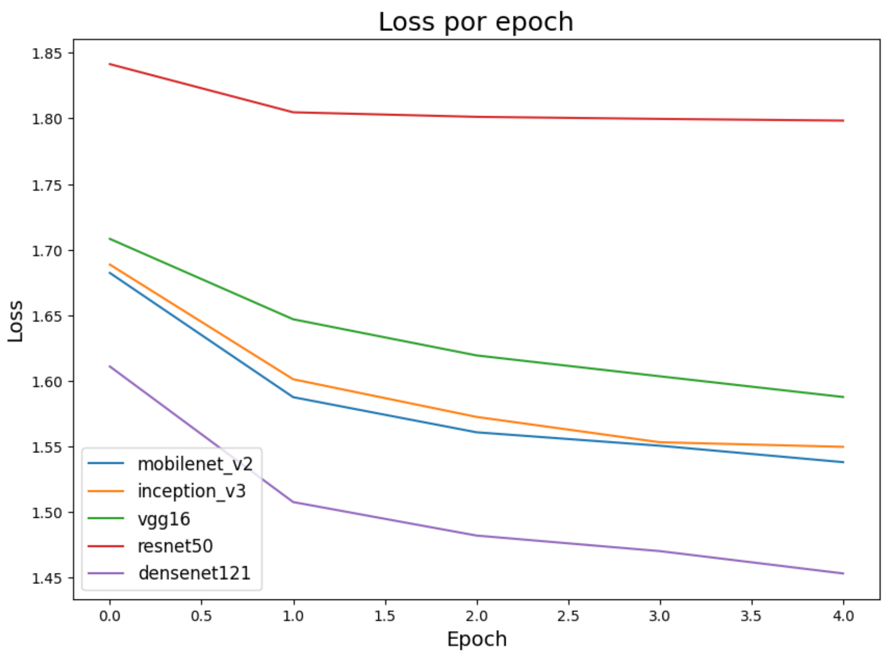
3. VGG16:
    - Validation Loss: 1.6179
    - Validation Accuracy: 0.3566

4. ResNet:
    - Validation Loss: 1.7853
    - Validation Accuracy: 0.2618

5. DenseNet:
    - Validation Loss: 1.3716
    - Validation Accuracy: 0.4719



(6.3 TL accuracy per epoch graph)



(6.4 TL loss per epoch graph)

Comparative Analysis

1. Validation Loss:
   - The model with the lowest validation loss is DenseNet, with a value of 1.3716.
   - InceptionV3 has the second-lowest validation loss, followed by MobileNet, VGG16, and ResNet, in that order.

    - This indicates that DenseNet is able to minimize the error between the predicted and actual values more effectively than the other models.
2. Validation Accuracy:
    - DenseNet achieves the highest validation accuracy among the models, with a value of 0.4719.
    - InceptionV3 has the second-highest validation accuracy, followed by MobileNet, VGG16, and ResNet, in that order.
    - DenseNet's higher accuracy suggests that it performs better in correctly classifying the images compared to the other models.

Overall, DenseNet stands out as the top-performing model based on both validation loss and accuracy. It demonstrates superior performance in minimizing error and achieving higher accuracy, making it a favorable choice for image classification tasks compared to MobileNet, InceptionV3, VGG16, and ResNet.

6.2.5 Decision

The model we will use for fine tuning will be DenseNet.

DenseNet is a suitable choice for transfer learning based on the following factors: accuracy, loss, number of parameters, and similar execution time.

1. Accuracy and Loss: After 5 epochs, DenseNet demonstrates competitive accuracy and loss values compared to other models. Its loss value of 1.3716 and validation accuracy of 0.4719 indicate effective learning and generalization capabilities, which are crucial for transfer learning tasks.
2. Number of Parameters: DenseNet exhibits a very large number of parameters, enabling it to capture intricate features and learn complex representations. This abundance of parameters contributes to its strong performance in recognizing patterns and extracting high-level information from the input data.
3. Execution Time: Although DenseNet has a larger number of parameters, it offers a similar execution time compared to other models. This implies that the additional computational complexity introduced by the increased parameters does not significantly impact the overall runtime. Therefore, the model's efficiency remains comparable to other transfer learning options.

Considering the combination of competitive accuracy and loss metrics, a high number of parameters for comprehensive feature learning, and similar execution time, DenseNet emerges as a favorable choice for transfer learning. Its capacity to extract detailed and informative representations can lead to improved performance across various tasks, making it a valuable asset in leveraging pre-trained models for new datasets or domains.

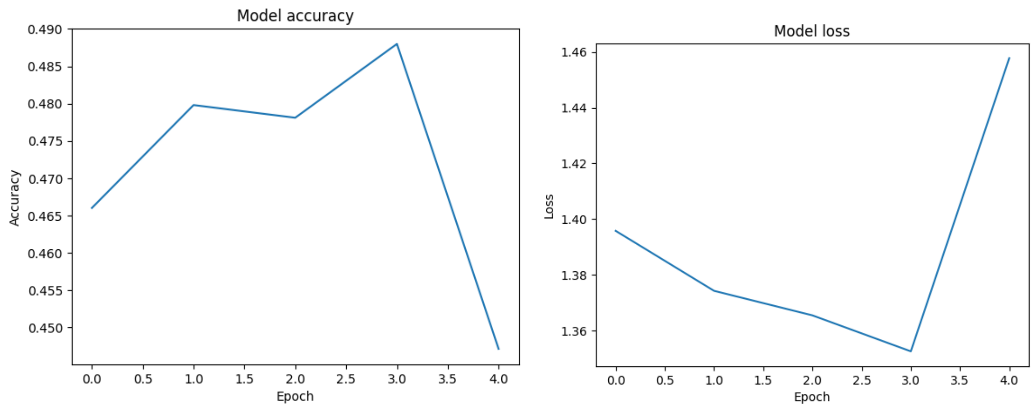
6.3 Fine tuning

6.3.1 FT Model architecture

The base architecture of our model is built upon the transfer learning approach using DenseNet, as explained in section 6.2.2. However, we need to decide how many layers of DenseNet to retrain for our specific task.
To make this decision, we conducted three experiments. In each experiment, we trained DenseNet for 5 epochs with a different percentage of layers unfrozen. Specifically, we unfroze 1% of the layers in the first experiment, 5% of the layers in the second experiment, and finally, 10% of the layers in the third experiment.
Considering that the DenseNet model consists of 427 layers, these percentages correspond to unfreezing 4 layers, 21 layers, and 42 layers, respectively. By varying the number of unfrozen layers, we aim to explore the impact of different levels of fine-tuning on our model's performance.

Fine tuning with 1% unfreeze

The training results indicate a gradual decrease in the training loss over the course of the five epochs, which is a positive trend. However, it is important to note that the validation loss exhibits fluctuations, suggesting potential overfitting of the model. The validation loss reaches its lowest point in the fourth epoch, indicating the best generalization performance. The test results reveal a test loss of 1.4577 and a test accuracy of 44.71%. This relatively high loss value and lower accuracy further emphasize the need for improvements in the model's ability to generalize to unseen data.

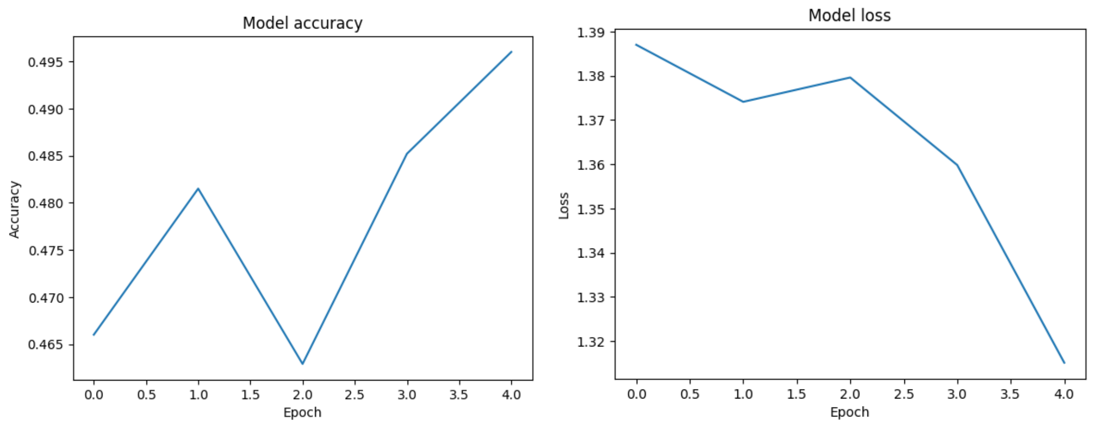


(6.5 FT accuracy and loss per epoch graphs 1% unfreeze)

Fine tuning with 5% unfreeze

In this case, the training results demonstrate a consistent improvement in training accuracy over the five epochs. The model shows a gradual decrease in training loss, indicating effective learning during the training process.
In terms of validation, the results show fluctuations in the validation loss throughout the epochs, suggesting potential overfitting. The lowest validation loss is achieved in the fifth epoch, indicating the best generalization performance. Additionally, the validation accuracy shows a steady increase over the epochs, reaching around 49.60% in the final epoch.

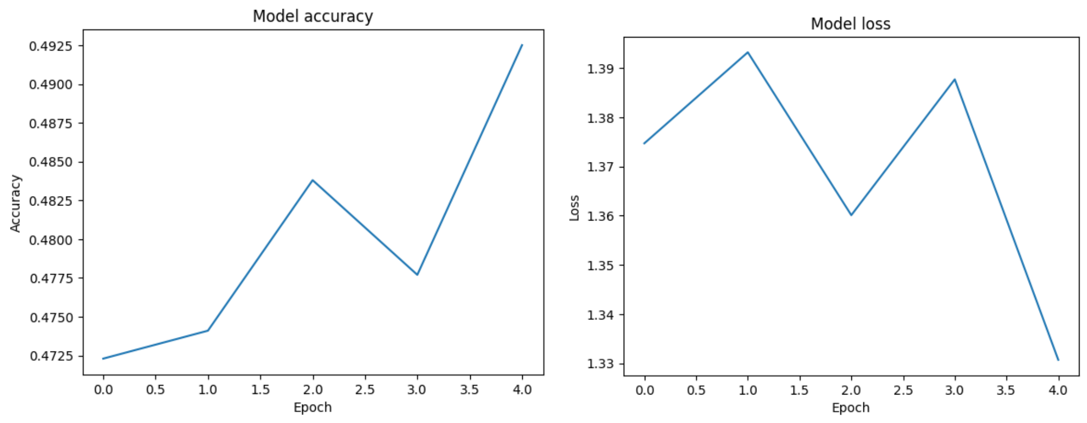
The test results reveal a test loss of 1.3151 and a test accuracy of 49.60%. These values indicate a better performance than the first test and suggest that the model can generalize better to unseen data.



(6.6 FT accuracy and loss per epoch graphs 5% unfreeze)

Fine tuning with 10% unfreeze

Unfreezing 10% of the layers also shows a consistent improvement in training accuracy over the five epochs. The model demonstrates a gradual decrease in training loss, suggesting effective learning during the training process as well as in the 5% training test.
Regarding the validation metrics, the validation loss fluctuates throughout the epochs, indicating possible overfitting. The lowest validation loss is achieved in the first epoch, suggesting better generalization performance at the beginning. However, the subsequent epochs show slightly higher validation losses, indicating a potential lack of improvement in generalization.
On the other hand, the validation accuracy shows a gradual increase over the epochs, reaching around 49.25% in the final epoch. This improvement suggests that the model is learning to classify the validation data more accurately over time.
In terms of the test results, the model achieves a test loss of 1.3307 and a test accuracy of 49.25%. These values indicate a reasonable level of performance, demonstrating that the model can generalize reasonably well to unseen data.

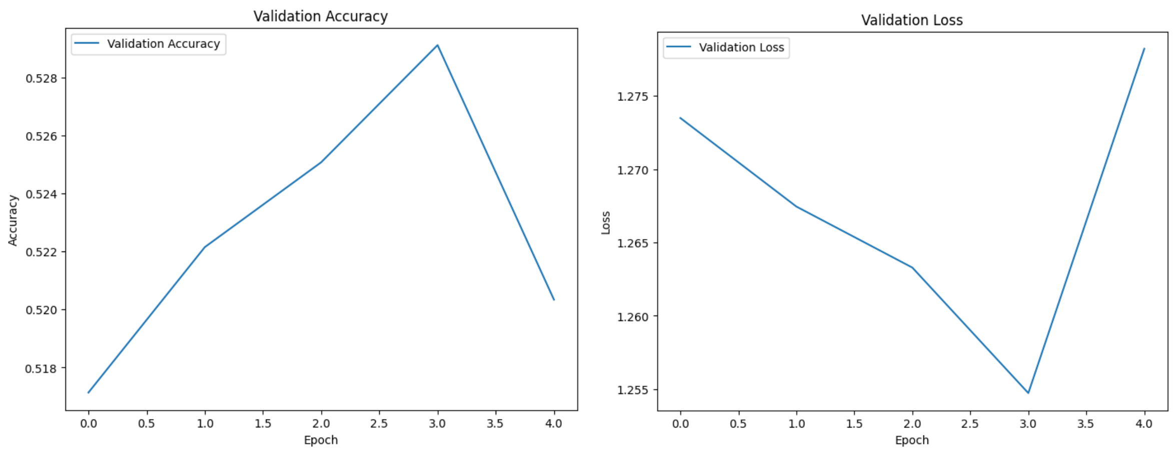


(6.7 FT accuracy and loss per epoch graphs 10% unfreeze)

Decision over the number of unfrozen layers
Since the 5% of unfrozen layers has reached the highest accuracy and lowest loss we will perform fine tuning with this model. Beyond 5%, increasing the number of unfrozen layers does not seem to significantly improve the model’s performance.

6.3.2 FT Additional Training and Results

The training process involved loading a pre-trained DenseNet model, which was previously saved as 'model_densenet_0.05.h5' corresponding to the model saved during the 5% unfrozen training test.
The model was then compiled using the Adam optimizer and the categorical cross-entropy loss function. The optimizer helps adjust the model's weights during training, while the loss function measures the discrepancy between the predicted and actual labels.
To train the model, it was fit to the training data for an additional five epochs using the fit function.
After completing the additional training for five epochs, the trained model was saved as 'model_densenet_0.05_trained5+5.h5'. This allowed preserving the updated model for future use or evaluation.



(6.8 FT accuracy and loss per epoch graphs additional 5 epoch)

The plots above show the progress of the model's performance over the additional epochs, from epoch 6 to epoch 10 of training:

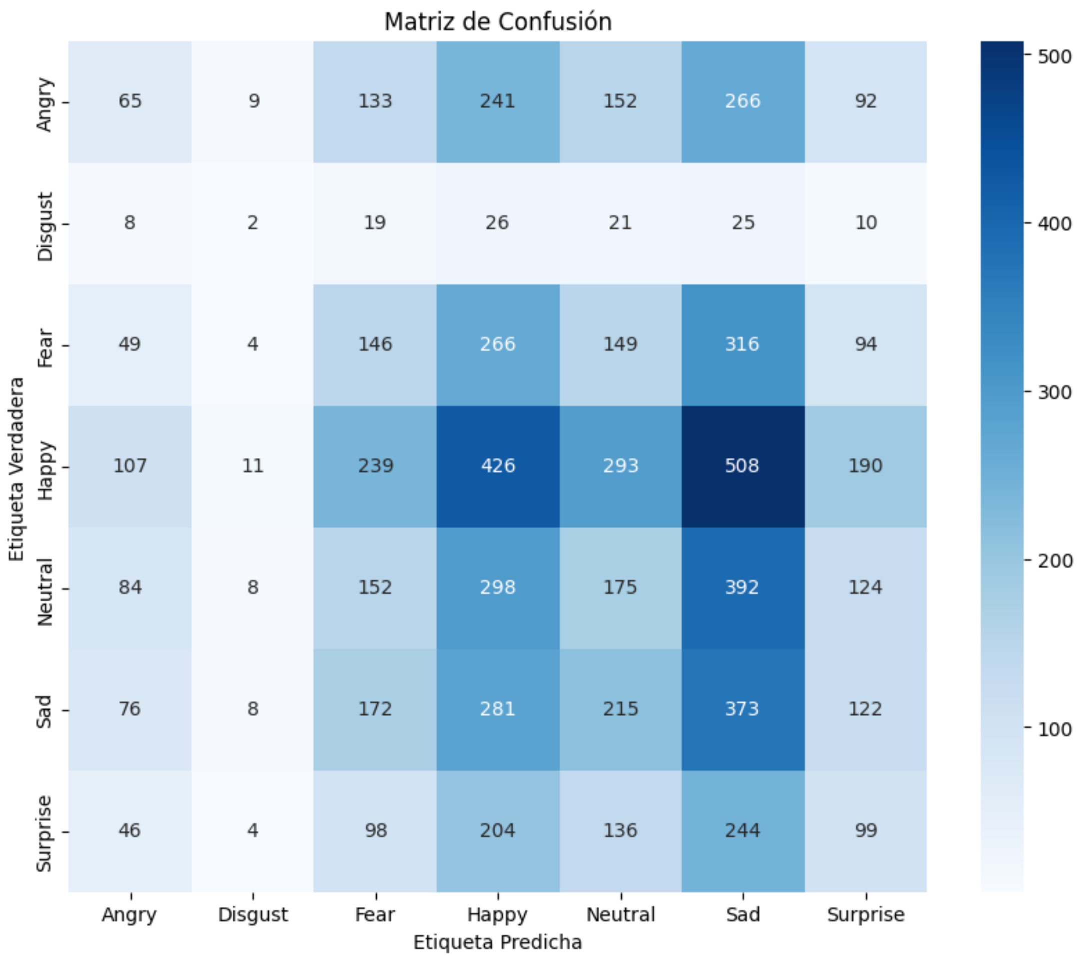
- Validation Loss: The validation loss is the error on a separate validation dataset that helps evaluate the model's generalization. The validation loss fluctuates slightly throughout the training, ranging from 1.2633 to 1.2782. These variations indicate that the model's performance on unseen data may not be consistent.
- Validation Accuracy: The validation accuracy measures the model's performance on the validation dataset. It shows a modest increase from 0.5171 in the first epoch to 0.5203 in the fifth epoch. This suggests that the model is generalizing its learned patterns to some extent, although there is room for improvement.

In this case, we can observe a gradual decrease in loss and a slight improvement in accuracy over the course of the additional 5 epochs. However, the accuracy values achieved are relatively moderate, indicating that the model may be reaching a plateau and will be difficult for it to achieve higher performance without architectural adjustments.

Confusion Matrix Explanation

The confusion matrix represents the performance of a classification model by comparing the predicted labels with the true labels across different emotions. Here is an analysis of the confusion matrix:

- The matrix is a 7x7 grid, where each row corresponds to the true labels and each column corresponds to the predicted labels for the emotions.
- The emotions included in the matrix are 'Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', and 'Surprise'.
- The values within the matrix represent the counts of instances where a certain emotion was predicted (column) given the true emotion (row). Each cell contains a count value.
- The diagonal cells from the top left to the bottom right represent the correctly predicted emotions, where the predicted label matches the true label.
- The off-diagonal cells represent instances of misclassification, where the predicted label differs from the true label.



(6.9 FT confusion matrix)

Interpretation of the Confusion Matrix:

Analyzing the performance of the fine tuning model on each emotion individually we can extract the following observations:

- 'Angry': In the first row, we can observe that the emotion 'Angry' has been misclassified as 'Disgust' in 9 instances, 'Fear' in 133 instances, 'Happy' in 241 instances, 'Neutral' in 152 instances, 'Sad' in 266 instances, and 'Surprise' in 92 instances. This indicates that the model has difficulty distinguishing between 'Angry' and these other emotions, leading to misclassifications.
- 'Disgust': In the second row, we see that the emotion 'Disgust' has been misclassified as 'Angry' in 8 instances, 'Fear' in 19 instances, 'Happy' in 26 instances, 'Neutral' in 21 instances, 'Sad' in 25 instances, and 'Surprise' in 10 instances. This suggests that the model struggles to accurately differentiate 'Disgust' from these other emotions.
- 'Fear': Looking at the third row, we observe misclassifications of the emotion 'Fear' as 'Angry' in 49 instances, 'Disgust' in 4 instances, 'Happy' in 146 instances, 'Neutral' in 266 instances, 'Sad' in 149 instances, and 'Surprise' in 94 instances. This indicates challenges in distinguishing 'Fear' from these specific emotions.
- 'Happy': In the fourth row, we can see that 'Happy' has been misclassified as 'Angry' in 107 instances, 'Disgust' in 11 instances, 'Fear' in 239 instances, 'Neutral' in 426 instances, 'Sad' in 293 instances, and 'Surprise' in 190 instances. This suggests that the model struggles to accurately identify 'Happy' and often confuses it with these other emotions.
- 'Neutral': Examining the fifth row, we observe misclassifications of the emotion 'Neutral' as 'Angry' in 84 instances, 'Disgust' in 8 instances, 'Fear' in 152 instances, 'Happy' in 298 instances, 'Sad' in 175 instances, and 'Surprise' in 392 instances. This indicates difficulty in distinguishing 'Neutral' from these specific emotions.
- 'Sad': Looking at the sixth row, we can see misclassifications of 'Sad' as 'Angry' in 76 instances, 'Disgust' in 8 instances, 'Fear' in 172 instances, 'Happy' in 281 instances, 'Neutral' in 215 instances, and 'Surprise' in 373 instances. This suggests challenges in accurately identifying 'Sad' and distinguishing it from these other emotions.
- 'Surprise': In the last row, we observe misclassifications of 'Surprise' as 'Angry' in 46 instances, 'Disgust' in 4 instances, 'Fear' in 98 instances, 'Happy' in 204 instances, 'Neutral' in 136 instances, and 'Sad' in 244 instances. This indicates difficulty in correctly classifying 'Surprise' and differentiating it from these specific emotions.

The confusion matrix analysis reveals challenges in accurately classifying and differentiating certain emotions. The model struggles with misclassifications, particularly for 'Angry' and 'Disgust'. With 'Fear' and 'Surprise' there is also an important rate of misrepresented images. There is a notable bias in classifying images in either 'Happy', 'Neutral' or 'Sad'. Nevertheless, there is also a substantial amount of misclassification among these three categories.

6.5 References 6

- Team, K. (n.d.-d). *Keras documentation: Keras Applications*. https://keras.io/api/applications/ *[13/06/2023]*
- Huilgol, P. (2023). Top 4 Pre-Trained Models for Image Classification with Python Code. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/ *[13/06/2023]*
- Leo, M. S. (2022, March 19). How to Choose the Best Keras Pre-Trained Model for Image Classification. *Medium*. https://towardsdatascience.com/how-to-choose-the-best-keras-pre-trained-model-for-image-classification-b850ca4428d4 *[13/06/2023]*
- *Google Colaboratory*. (n.d.-b). *Transfer learning with TensorFlow Hub.* https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/transfer_learning_with_hub.ipynb#scrollTo=AFgDHs6VEFRD *[13/06/2023]*
- Pankajandhale. (2022). Emotion Detector_transfer_learning. *Kaggle*. https://www.kaggle.com/code/pankajandhale/emotion-detector-transfer-learning/notebook *[13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

7. CNN

7.1 Architecture of the model

7.1.1 Convolutional Layers

Convolutional layers are fundamental building blocks in convolutional neural networks (CNNs). They are designed to capture local patterns and spatial dependencies in the input data. Each convolutional layer applies a set of learnable filters to the input data and performs a convolution operation, resulting in a feature map that represents the presence of certain features or patterns.

The architecture of the convolutional layers described in the model aims to progressively extract hierarchical representations of the input images. Here's a justification for the chosen architecture:

1. Increasing complexity and abstraction: The model starts with a relatively simple convolutional layer with 256 filters, followed by a more complex layer with 512 filters. This progression allows the network to capture both low-level and high-level features in the images.
2. Regularization and control over model complexity: Batch normalization is applied after each convolutional layer. It normalizes the activations within each batch, helping to stabilize the learning process and reduce the impact of covariate shift. This regularization technique improves the model's generalization ability and prevents overfitting.
3. Downsampling and spatial information reduction: Max-pooling is performed after every two convolutional layers. It reduces the spatial dimensions of the feature maps, providing a form of translation invariance and enabling the network to focus on the most relevant features while discarding less informative spatial information.
4. Dropout for regularization: Dropout is applied after each max-pooling layer. It randomly deactivates a fraction of the neurons, forcing the network to learn redundant representations and reducing the risk of overfitting.

The chosen architecture strikes a balance between capturing complex features and preventing overfitting. It allows the model to learn hierarchical representations by progressively combining local patterns into higher-level abstractions while incorporating regularization techniques to enhance generalization.

Here is a description of the convolutional layers used in the model, referencing the corresponding code:

1. Convolutional Layer 1:
    - Number of filters: 256
    - Kernel size: 3x3
    - Activation function: ReLU
    - Padding: 'same'
    - Additional operation: Batch normalization (BatchNormalization())
2. Convolutional Layer 2:
    - Number of filters: 512
    - Kernel size: 3x3

    - Activation function: ReLU
    - Padding: 'same'
    - Additional operation: Batch normalization (BatchNormalization())
3. Convolutional Layer 3:
    - Number of filters: 256
    - Kernel size: 3x3
    - Activation function: ReLU
    - Padding: 'same'
    - Additional operation: Batch normalization (BatchNormalization())
4. Convolutional Layer 4:
    - Number of filters: 128
    - Kernel size: 3x3
    - Activation function: ReLU
    - Padding: 'same'
    - Additional operation: Batch normalization (BatchNormalization())

7.1.2 Pooling Layers

Pooling layers are used in convolutional neural networks (CNNs) to downsample the feature maps and reduce the spatial dimensions. In the described model, MaxPooling layers are utilized. Here are the details of the pooling layers:

1. MaxPooling Layer 1:
    - Pool size: (2, 2)
    - Purpose: This layer performs max pooling with a 2x2 pool size. It divides the input feature map into non-overlapping regions of size 2x2 and outputs the maximum value within each region. The pool size of (2, 2) effectively reduces the spatial dimensions by half.
2. MaxPooling Layer 2:
    - Pool size: (2, 2)
    - Purpose: This layer again performs max pooling with a 2x2 pool size. It further reduces the spatial dimensions by half from the previous layer's output.
3. MaxPooling Layer 3:
    - Pool size: (2, 2)
    - Purpose: Similar to the previous pooling layers, this layer applies max pooling with a 2x2 pool size. It continues to downsample the feature maps and reduce the spatial dimensions.

The pooling layers help in reducing the spatial resolution of the feature maps while preserving the most salient features. By downsampling, the pooling layers provide translational invariance and improve the model's robustness to slight spatial variations in the input. The choice of a pool size of (2, 2) allows for a gradual reduction in the spatial dimensions, enabling the model to capture and retain important spatial information while discarding redundant details.

7.1.3 Flattening

The flattening layer plays a crucial role in the architecture of the model. It is positioned after the pooling layers and before the fully connected layers. The purpose of the flattening layer is to reshape the multidimensional feature maps obtained from the convolutional and pooling layers into a one-dimensional vector.
Convolutional and pooling operations result in feature maps that have a spatially structured representation. However, the fully connected layers require a flattened input format. The flattening layer addresses this requirement by converting the spatially structured features into a format suitable for traditional neural network architectures.
By applying the flattening layer, the model can effectively transition from extracting local features in the earlier layers to learning global patterns and making predictions in the subsequent fully connected layers. The flattening layer eliminates the spatial information present in the feature maps and transforms it into a one-dimensional vector, preserving the information from each location.
In the provided model architecture, the flattening layer is implemented with the following line of code: model.add(Flatten()). This step ensures that the output of the preceding layers, consisting of feature maps with spatial dimensions, is reshaped into a one-dimensional vector. The flattened vector serves as the input to the fully connected layers, enabling them to learn meaningful patterns and relationships.
Including the flattening layer in the model architecture facilitates the extraction of high-level features and enables the model to capture complex patterns across the entire input image.

7.1.4 Fully Connected Layers (Dense)

Fully connected layers, also known as dense layers, are the final layers in a CNN that take the flattened feature maps from the preceding convolutional and pooling layers and perform classification or regression tasks. In the described model, there are two fully connected layers. Here are the details:

1. Dense Layer 1:
    - Number of neurons: 256
    - Activation function: ReLU (Rectified Linear Unit)
    - Additional layers/operations: Batch Normalization, Dropout (0.25)
2. This dense layer consists of 256 neurons with the ReLU activation function. ReLU is commonly used in deep learning models to introduce non-linearity and enable the model to learn complex patterns. Batch normalization is applied to normalize the outputs of the previous layer, making the model more stable and improving its generalization. Dropout with a rate of 0.25 is used as a regularization technique to randomly deactivate 25% of the neurons during training, preventing overfitting and promoting better generalization.
3. Dense Layer 2:
    - Number of neurons: 7
    - Activation function: Softmax
4. The second and final dense layer has 7 neurons, corresponding to the 7 emotion classes. The softmax activation function is used to obtain probabilities for each class, indicating the model's confidence in predicting each emotion. Softmax ensures that the sum of the probabilities across all classes is 1, making it suitable for multi-class classification problems.

The fully connected layers contribute to the model's ability to learn complex representations by combining the extracted features from the earlier layers. The use of ReLU activation promotes non-linearity, enabling the model to capture intricate relationships between the features. Batch normalization enhances the stability of the model during training, and dropout aids in preventing overfitting. Finally, the softmax activation function provides probability-based outputs for multi-class classification.

7.1.5 Output Layer

The output layer of the model is the final layer responsible for generating the predictions. In this case, the model is designed to classify emotions, and therefore, the output layer is configured accordingly. Here are the details:
Output Layer:

- Number of output units: 7
- Activation function: Softmax

The output layer consists of 7 output units, which correspond to the 7 different emotions the model aims to classify. Each output unit represents the probability of the input image belonging to a specific emotion class. The activation function used in the output layer is softmax, which ensures that the predicted probabilities sum up to 1 and allows for multi-class classification.
The number of output units matches the number of emotion classes, which are:

1. Angry
2. Disgust
3. Fear
4. Happy
5. Neutral
6. Sad
7. Surprise

By using the softmax activation function, the output layer provides a probability distribution across these 7 emotions, indicating the model's confidence in its predictions. The highest probability among the output units represents the predicted emotion for a given input image.

7.1.6 Summary

The architecture of the model consists of convolutional layers, pooling layers, a flattening layer, fully connected layers, and an output layer. These components are interconnected to form a deep convolutional neural network designed for emotion recognition.

The convolutional layers are responsible for capturing local patterns and features from the input images. In this model, multiple convolutional layers with varying numbers of filters, kernel sizes, and activation functions are applied. Batch normalization is employed after each convolutional layer to improve the stability and speed of training. Dropout layers are also inserted to prevent overfitting by randomly dropping a fraction of the inputs.

Pooling layers follow the convolutional layers to downsample the feature maps and extract the most important information. Max pooling is utilized with a pool size of (2, 2) to retain the most dominant features while reducing the spatial dimensions.

The flattening layer is introduced to convert the multidimensional feature maps obtained from the convolutional and pooling layers into a one-dimensional vector. It prepares the data for the subsequent fully connected layers by eliminating the spatial structure and providing a flattened representation.

The fully connected layers follow the flattening layer to learn global patterns and make predictions. These layers consist of densely connected neurons with activation functions, allowing the model to capture complex relationships between the extracted features. Batch normalization and dropout are also applied to enhance the model's generalization and prevent overfitting.

Finally, the output layer consists of seven neurons, representing the seven emotion classes: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. The softmax activation function is used to produce probability distributions over these classes, indicating the model's confidence in each emotion category.

Throughout the model's construction, considerations were given to strike a balance between model complexity and generalization. The number of layers, filters, kernel sizes, and other hyperparameters were selected based on empirical evaluation and commonly accepted practices in the field of computer vision. Regularization techniques such as batch normalization and dropout were employed to improve the model's performance and prevent overfitting.

7.2 Hyperparameters

7.2.1 Batch size

Overview
The batch size is a hyperparameter that determines the number of images processed in each training step. It has an impact on both memory consumption and training speed. A larger batch size allows for more parallelism but requires more memory, while a smaller batch size consumes less memory but may result in slower convergence.
In this model, a batch size of 32 was chosen.

Justification
This value was selected after conducting research on multiple works that utilized the same dataset, such as the kaggle repositories mentioned in the subsection References 7.
In these works, a batch size of 64 was commonly used. However, for the purpose of this thesis, a batch size of 32 was chosen due to the limited computational resources available. This batch size allows for efficient memory utilization and faster training times while still providing satisfactory performance. The selection of 32 as the batch size was made to strike

a balance between computational efficiency and model effectiveness, given the specific constraints of this study.
The chosen batch size strikes a balance between efficient memory usage and reasonable training speed. It enables the model to process a sufficient number of images in each iteration, allowing for effective gradient estimation and parameter updates.
Furthermore, it's worth noting that the batch size may have implications for generalization. A larger batch size can provide a more stable estimate of the gradient but may also reduce the diversity of samples within each batch. Conversely, a smaller batch size introduces more variability but can result in noisier gradient estimates. The chosen batch size in this model aims to find a suitable compromise between stability and diversity to facilitate effective training and generalization.

7.2.2 Dropout rate and regularization

Overview
The dropout rate is a regularization technique used in neural networks to prevent overfitting. It involves randomly dropping out a fraction of the neurons during training, forcing the remaining neurons to learn more robust and independent representations of the data. By doing so, dropout helps to reduce the reliance of the model on specific neurons and encourages the learning of more generalized features.

Justification
In this model, a dropout rate of 0.4 was chosen based on insights from prior research on similar studies using the dataset. Several experiments have been conducted with different combinations of dropout layers and L2 regularization (a type of weight regularization that adds a penalty term to the loss function), but it was decided to use a high dropout rate to simplify the model and expedite the convergence during training.
Specifically, previous works utilizing the same dataset, commonly adopted dropout rates ranging from 0.25 to 0.4. These studies demonstrated that employing dropout within this range effectively mitigates overfitting and enhances the model's generalization ability.
While the combination of dropout and L2 regularization can provide further regularization benefits, it was opted to solely focus on a higher dropout rate for the sake of simplicity and faster convergence during training. This decision allowed the model to learn more quickly while still benefiting from the regularization effect of dropout.

7.2.3 Learning rate

Overview
The learning rate is a hyperparameter that determines the step size at which the model adjusts its weights during training. It plays a crucial role in the optimization process, as a suitable learning rate can help the model converge faster and reach better performance, while an inappropriate learning rate can hinder or even prevent convergence.

Justification
In this model, an initial learning rate of 0.001 was chosen based on empirical observations and best practices in the field. This value is commonly used as a starting point for many convolutional neural network (CNN) architectures.

Previous learning rate tunning
During earlier stages before finalizing the model, several implementations have been tested that also aim to optimize the learning rate during training.

lr_scheduler:
The lr_scheduler is a built-in callback provided by Keras. It allows for dynamic learning rate scheduling based on epoch number. You can define a schedule that specifies how the learning rate should change over time. For example, you can set a higher learning rate initially for faster convergence and then gradually decrease it as training progresses. This type of scheduling can be useful when dealing with complex optimization landscapes or large datasets.

ReduceLROnPlateau:
The ReduceLROnPlateau callback in Keras automatically reduces the learning rate when a plateau in the validation loss or metric is detected. It monitors a specified metric, such as validation loss, and reduces the learning rate if no improvement is observed for a certain number of epochs. This technique is beneficial when training reaches a point of stagnation, and further adjustments in the learning rate may help the model escape from local minima or plateaus.

Superconvegence:
Superconvergence is a training technique that aims to accelerate the convergence of deep neural networks. It involves using high learning rates combined with cyclic learning rate schedules to achieve faster training and improved model performance. By starting with a high learning rate and gradually reducing it during training, the model can quickly explore the loss landscape and find a good solution. Cyclic learning rate schedules oscillate the learning rate between high and low values, allowing the model to effectively explore different regions of the loss landscape. Superconvergence offers benefits such as faster convergence, improved training stability, and the potential to achieve state-of-the-art results with fewer training epochs. However, its effectiveness may vary depending on the specific architecture and dataset, requiring careful experimentation and fine-tuning for optimal results.

The lr_scheduler was discarded as an option because it was desired a less arbitrary approach to decrease the learning rate. Although ReduceLROnPlateau showed significant improvements in the training process, it occasionally exhibited delays in adjusting the learning rate, potentially impacting training efficiency. Furthermore, Superconvergence, while a promising technique, was not pursued due to its complexity and the limited time available to thoroughly explore its implementation and potential benefits.

CustomScheduler class
To further optimize the learning rate during training, a custom learning rate scheduler has been implemented. The scheduler, defined as the CustomScheduler class, dynamically adjusts the learning rate based on the validation accuracy over a certain number of epochs. The key features of the CustomScheduler are as follows:

1. Initialization: The scheduler is initialized with three parameters: factor, patience, and min_lr. The factor determines the reduction factor applied to the learning rate, the patience specifies the number of epochs with no improvement in validation accuracy

before reducing the learning rate, and min_lr defines the minimum allowed learning rate.
2. Tracking Best Accuracy: The best_acc attribute keeps track of the highest validation accuracy observed during training, initially set to 0.
3. Monitoring Epochs: At the end of each epoch, the on_epoch_end method is called. It checks the validation accuracy against best_acc and updates the wait counter accordingly. If the accuracy improves, best_acc is updated, and wait is reset to 0. Otherwise, wait is incremented by 1.
4. Reducing Learning Rate: If wait reaches or exceeds the specified patience value, it indicates a lack of improvement in validation accuracy. In such cases, the learning rate is reduced by multiplying it with the factor. The updated learning rate is then applied to the optimizer of the model.

The purpose of the CustomScheduler is to dynamically adjust the learning rate during training. By monitoring the validation accuracy, it determines whether there is a plateau or decline in model performance. If no improvement is observed for a certain number of epochs (patience), the learning rate is reduced to potentially help the model overcome stagnation and find a better region of optimization. This dynamic learning rate adjustment can lead to faster convergence and improved overall performance of the model.

7.2.4 Number of epochs

Overview
The number of epochs refers to the total number of times the model iterates over the entire training dataset during the training process. It is an essential hyperparameter that determines how long the model trains and influences its ability to capture patterns and generalize well to unseen data. Setting the appropriate number of epochs is crucial to prevent underfitting or overfitting.

Justification
During the development of the CNN model, multiple experiments were conducted, which can be categorized into two types based on the number of epochs used for training: short tests of 15 to 30 epochs and long tests of 75 to 100 epochs. The tests were not extended beyond 100 epochs due to the time it took to train the model and limitations in GPU usage on Google Colab.
Initially, this model was trained for 25 epochs, and upon observing its good performance, it was saved along with its training history. The original plan was to train the model in multiple fitting sessions using the .fit() function with 30 epochs each. However, the model barely increased accuracy in the first fitting session itself so no further 30 epoch fittings were executed.
In conclusion the model has been trained for a total of 55 epochs. This number of epochs allowed the model to capture the underlying patterns in the data while considering the constraints of training time and GPU availability.

EarlyStopping
EarlyStopping is a useful callback in deep learning models that helps prevent overfitting and determine the optimal stopping point during training. It monitors a specified metric, such as

validation loss or accuracy, and halts the training process if there is no improvement in the monitored metric for a defined number of epochs.
Although it was not necessary during the training of the final model, EarlyStopping has been a highly useful callback in the training of predecessor models. It has played a crucial role in preventing overfitting and saving time by automatically stopping training when it reaches a point of diminishing returns. By utilizing EarlyStopping, we were able to avoid prolonged training with an excessive number of epochs, which can lead to overfitting and wasted computational resources. This callback has been instrumental in optimizing the training process, ensuring efficient use of time and resources while maintaining good generalization performance in the models prior to the final version.

7.3 Training process

7.3.1 Loss function

The loss function used in the training process was categorical_crossentropy. This loss function is commonly used for multi-class classification problems, where each input can belong to one of several classes. The categorical_crossentropy loss measures the dissimilarity between the predicted probability distribution and the true distribution of the target classes.
By specifying loss='categorical_crossentropy' in the model.compile() function, the model was optimized to minimize this loss during training. The goal was to train the model to accurately predict the correct class label for each input image.
During training, the model's performance was evaluated based on both the loss value and the accuracy metric. The accuracy metric indicates the percentage of correctly predicted labels compared to the total number of samples. By monitoring the loss and accuracy metrics, we could assess the model's progress and make adjustments if necessary.
The use of categorical_crossentropy as the loss function was suitable for the multi-class emotion recognition task, where the model needed to classify each input image into one of the different emotional categories.

7.3.2 Optimizer

The optimizer used in the training process was Adam. Adam stands for Adaptive Moment Estimation and is a popular optimization algorithm commonly used in deep learning models. It combines the benefits of two other optimization techniques, namely AdaGrad and RMSprop, to achieve efficient weight updates during training.
By specifying optimizer='adam' in the model.compile() function, the model utilized the Adam optimizer to adjust the weights and biases of the neural network during training. The Adam optimizer adapts the learning rate for each weight parameter individually, based on the estimates of both the first and second moments of the gradients. This adaptive learning rate allows the optimizer to converge faster and more efficiently.

The choice of Adam as the optimizer was based on its strong performance in a variety of deep learning tasks and its ability to handle large-scale datasets. It helped in accelerating the training process and finding optimal weight configurations for the model.

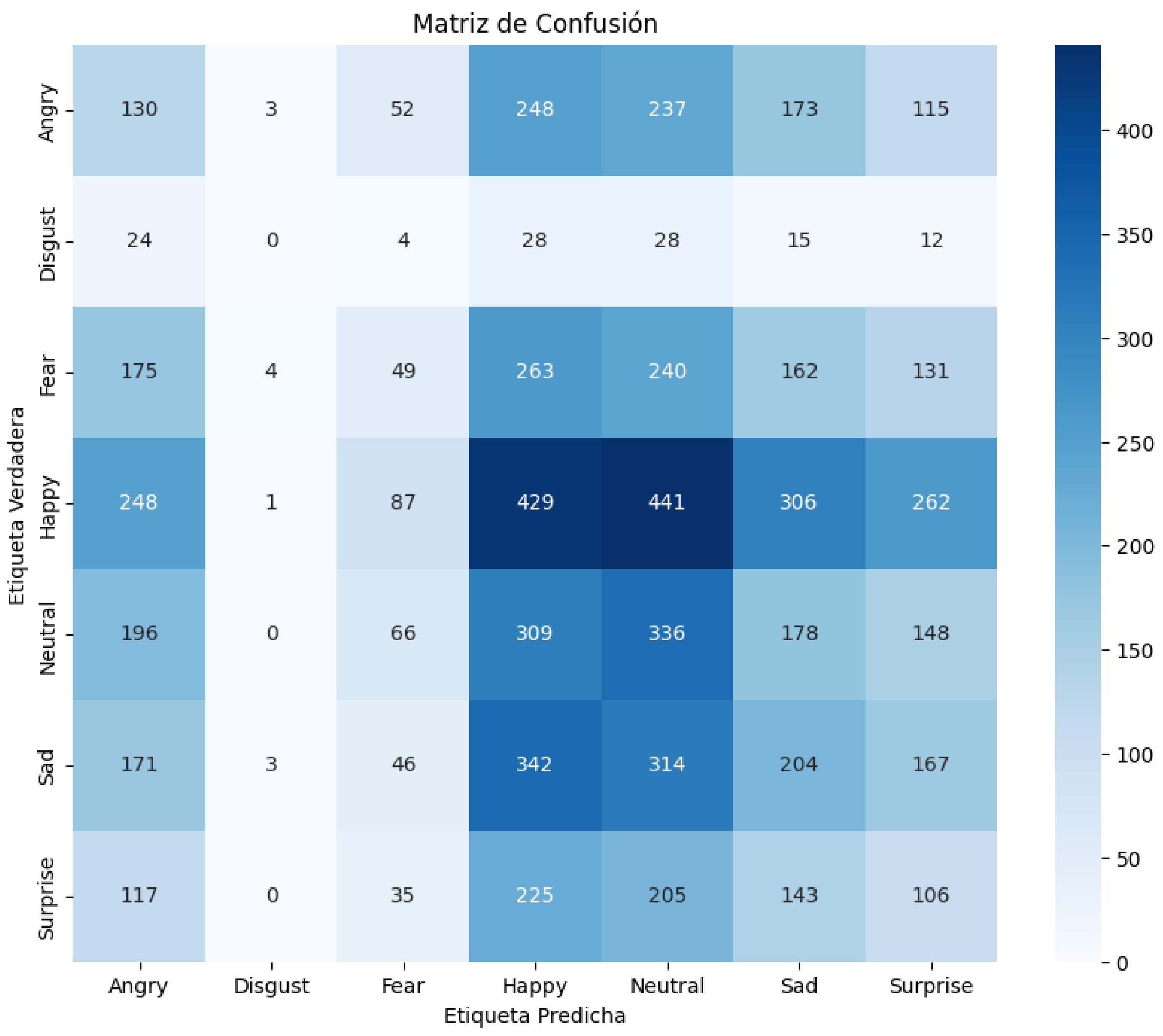
7.4 Results

7.4.1 Model Accuracy

The accuracy achieved by the model on the test/validation set is a crucial metric for assessing its performance. It measures the proportion of correctly classified samples out of the total number of samples. In this case, after training the model for 55 epochs, the final accuracy on the validation set was 0.6323, indicating that approximately 63.23% of the samples were correctly classified.
This accuracy score provides valuable insights into the model's ability to generalize and make accurate predictions on unseen data. A higher accuracy indicates that the model has learned meaningful patterns and features from the training data, enabling it to correctly classify emotions in the validation set. It is important to note that the accuracy achieved may vary depending on factors such as the complexity of the dataset and the model architecture.
Evaluating the model's accuracy is essential for determining its suitability for practical applications. The achieved accuracy of 0.6323 suggests that the model has learned to recognize emotions with a moderate level of success. Further analysis of the confusion matrix and learning curves will provide a more comprehensive understanding of the model's performance and potential areas for improvement.

7.4.2 Confusion Matrix

The confusion matrix provides valuable insights into the distribution of predictions across each class, allowing us to analyze the model's performance on individual emotion categories. The matrix is generated based on the model's predictions and the true labels of the validation or test set. In this case, the confusion matrix is as follows:



	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Angry	130	3	52	248	237	173	115
Disgust	24	0	4	28	28	15	12
Fear	175	4	49	263	240	162	131
Happy	248	1	87	429	441	306	262
Neutral	196	0	66	309	336	178	148
Sad	171	3	46	342	314	204	167
Surprise	117	0	35	225	205	143	106

(7.1 CNN confusion matrix)

The confusion matrix is visualized using a heatmap, where each cell represents the number of samples predicted for a particular emotion category (columns) compared to the true labels (rows). The numbers in each cell indicate the count of samples falling into that category.
The diagonal elements represent the number of correctly classified samples for each emotion category. For example, the model correctly predicted 130 Angry samples, 0 Disgust samples, 49 Fear samples, 429 Happy samples, 336 Neutral samples, 204 Sad samples, and 106 Surprise samples.
Off-diagonal elements indicate misclassifications. For instance, the model misclassified some samples of Angry as Fear, Happy, Neutral, Sad, or Surprise, and misclassified some samples of Fear as Angry, Happy, Neutral, Sad, or Surprise.
Analyzing the confusion matrix, we can observe the following:

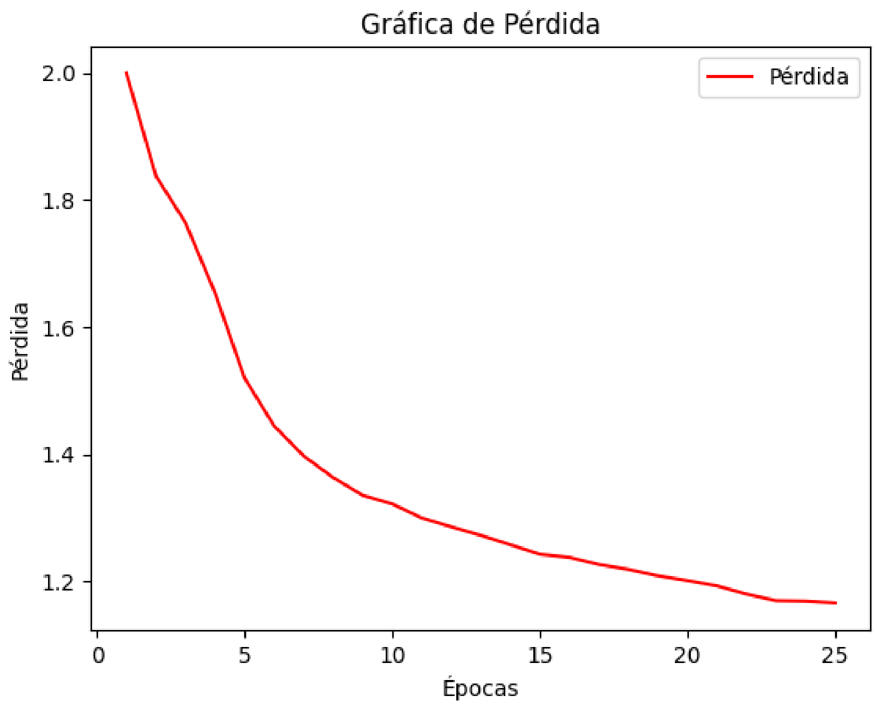
1. Emotion "Angry": The model correctly classified 130 Angry samples. However, it misclassified some Angry samples as Fear, Happy, Neutral, Sad, or Surprise. This suggests that the model might struggle to distinguish Angry from other emotions.
2. Emotion "Disgust": The model did not correctly classify any Disgust samples. This indicates that the model may face challenges in accurately recognizing the Disgust emotion.

3. Emotion "Fear": The model correctly predicted 49 Fear samples. However, it also misclassified some Fear samples as Angry, Happy, Neutral, Sad, or Surprise. This suggests that the model might have difficulty differentiating Fear from other emotions.
4. Emotion "Happy": The model performed well in classifying Happy samples, with 429 samples correctly predicted. Nevertheless, there were misclassifications where Happy samples were labeled as Angry, Fear, Neutral, Sad, or Surprise.
5. Emotion "Neutral": The model achieved a relatively high accuracy in recognizing Neutral samples, correctly predicting 336 samples. However, misclassifications occurred, with some Neutral samples being classified as Angry, Fear, Happy, Sad, or Surprise.
6. Emotion "Sad": The model accurately classified 204 Sad samples. Nonetheless, it also misclassified some Sad samples as Angry, Fear, Happy, Neutral, or Surprise.
7. Emotion "Surprise": The model achieved a reasonable performance in recognizing Surprise samples, correctly predicting 106 samples. However, there were misclassifications, with some Surprise samples being labeled as Angry, Fear, Happy, Neutral, or Sad.

7.4.3 Learning Curves

During the training process of the CNN model, we saved the loss and accuracy history for analysis. Plotting the learning curves provides visual insights into the model's progress over epochs. Here, we present four sets of learning curves: the first two correspond to the epochs from 1 to 25, and the latter two represent the epochs from 26 to 55.

Loss Function (Epochs 1-25):



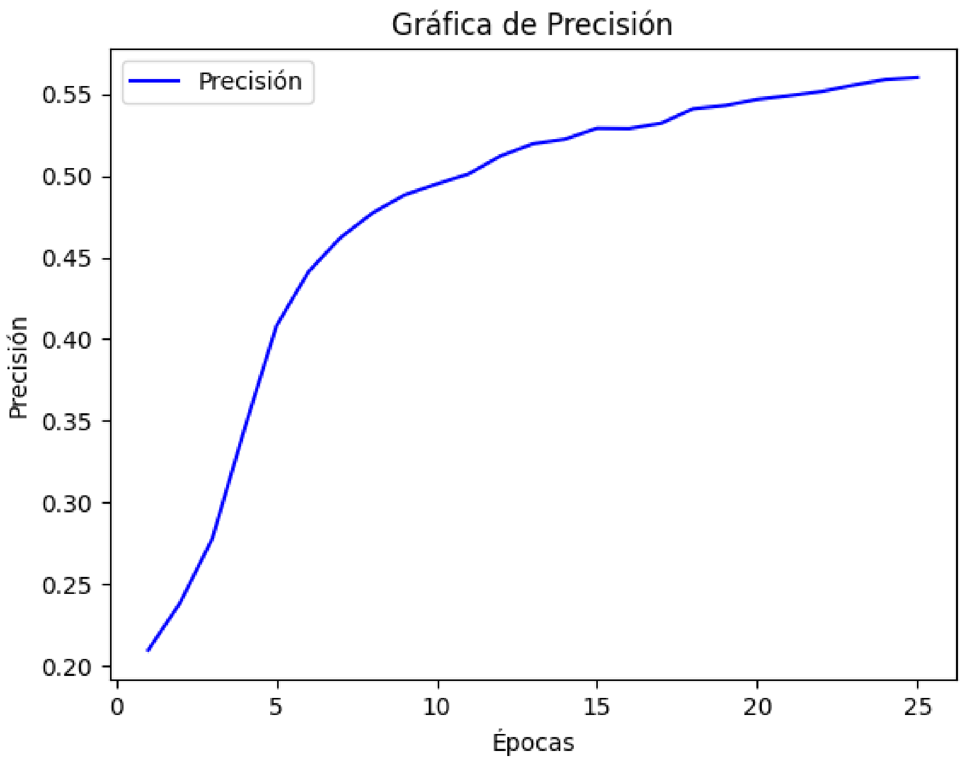
(7.2 CNN loss function per epoch 1-25 graph)

Analyzing the evolution of the loss function throughout the 25 epochs, we can observe the following patterns:

- During the initial epochs (1-5), there is a noticeable decrease in the loss function. The model starts to learn and adjust its weights to minimize the discrepancy between predicted and actual values. This indicates that the model is gradually improving its performance.
- From epochs 6 to 10, the loss continues to decrease, albeit at a slightly slower pace. The model is further refining its predictions and reducing the overall error.
- In epochs 11 to 15, the rate of improvement in the loss function slows down even more. The model's learning process has reached a point where the loss reduction becomes more challenging.
- From epochs 16 to 20, there is a relatively stable trend in the loss function, indicating that the model is converging. The loss function reaches a plateau, suggesting that the model has learned most of the information from the training data.
- In the final epochs (21-25), the loss function experiences minor fluctuations but maintains a relatively constant value. The model has likely reached its optimal performance, as further training does not significantly impact the loss.

Overall, the analysis of the loss function shows that the model initially improves rapidly, then gradually converges to a stable state.

Accuracy (Epochs 1-25):



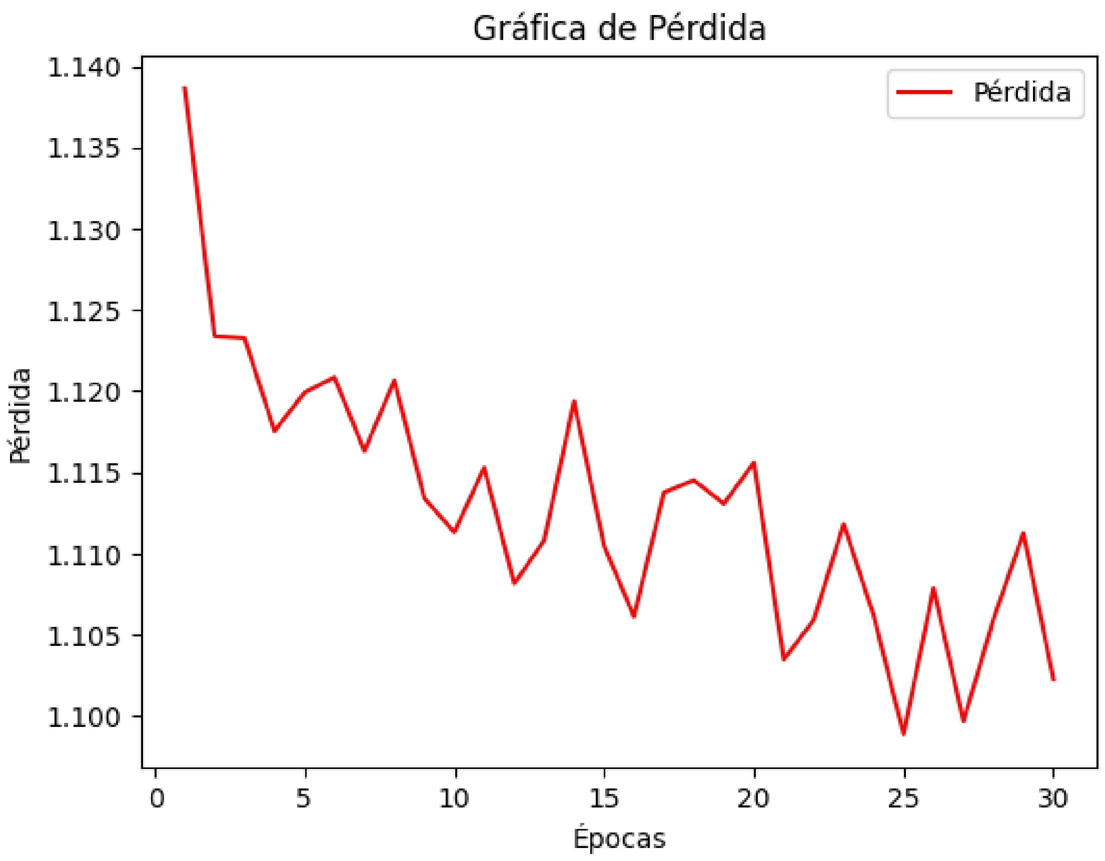
(7.3 CNN accuracy per epoch 1-25 graph)

Analyzing the evolution of the accuracy metric throughout the 25 epochs, we can observe the following patterns:

- In the initial epochs (1-5), the accuracy of the model shows a significant increase. This indicates that the model is successfully learning and making more accurate predictions on the training data.
- From epochs 6 to 10, the accuracy continues to improve, although at a slower rate compared to the initial epochs. The model is refining its predictions and becoming more adept at classifying the emotions in the training dataset.
- In epochs 11 to 15, the rate of improvement in accuracy slows down further. The model has already captured a substantial amount of information from the training data, and the accuracy gains become less significant.
- From epochs 16 to 20, the accuracy reaches a relatively stable state, with minor fluctuations. The model's performance has converged, and the accuracy remains consistent.
- In the final epochs (21-25), the accuracy metric shows slight variations, but it remains relatively constant. The model has likely reached its optimal performance, and additional training does not significantly improve the accuracy.

Overall, the analysis of the accuracy metric demonstrates the model's ability to classify emotions correctly. The accuracy initially improves rapidly, then gradually converges to a stable value.

Loss Function (Epochs 26-55):



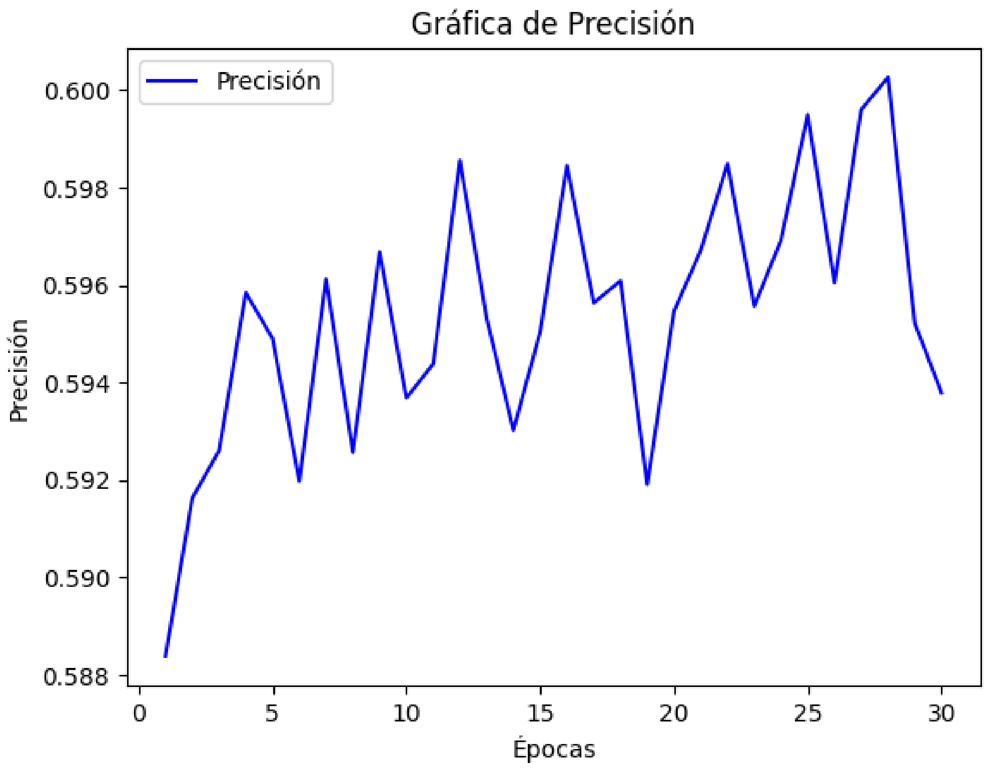
(7.4 CNN loss function per epoch 26-55 graph)

Analyzing the loss function in the second phase of the training (26 to 55 epochs):

- In the early epochs, the loss gradually decreases, albeit with significant fluctuations.
- Starting from epoch 5 (30), the loss shows a slower decreasing trend, with noticeable fluctuations in each epoch.
- Between epochs 15 and 20 (40 and 45), there is a tendency for the loss to stabilize, with values that are close to each other and without a clear decrease.
- Overall, the loss function remains around 1.1 values throughout the epochs, without improving significantly.
- There is no consistent and steep decrease in the loss throughout the epochs, which may indicate that the model is struggling to efficiently adjust its parameters and improve its performance.

In summary, this training shows a decrease in loss, but with fluctuations and a less rapid and consistent decrease compared to the initial 25 epoch training. This suggests that the model may be having difficulties in learning more complex patterns in the data and optimizing its parameters further.

Accuracy (Epochs 26-55):



(7.5 CNN accuracy per epoch 25-55 graph)

Analyzing the accuracy evolution in the second phase of the training (26 to 55 epochs):

- In the initial epochs of the second phase, the accuracy gradually increases, but with noticeable fluctuations.
- From epoch 5 (30) onwards, the accuracy shows a slower and less consistent improvement, with fluctuations in each epoch.

- Between epochs 15 and 20 (40 and 45), there is a tendency for the accuracy to stabilize, with values that remain relatively close to each other without significant improvement.
- Similar to the loss function, there is no significant and consistent improvement in the accuracy throughout the epochs. This suggests that the model may struggle to learn complex patterns and generalize well to unseen data.

When analyzing the accuracy in the second phase of the training we can observe that it has not increased significantly.Similarly to the loss function analysis, this indicates that the model's performance may have reached convergence or a plateau.

7.6 References 7

- Singhal, A. (2021b, December 26). Facial expression detection using Machine Learning in Python. *Medium*. https://medium.com/analytics-vidhya/facial-expression-detection-using-machine-learning-in-python-c6a188ac765f *[13/06/2023]*
- Codificando Bits. (2019b, March 23). *¿Qué son las REDES CONVOLUCIONALES?* [Video]. YouTube. https://www.youtube.com/watch?v=HyZFfBU0ADg *[13/06/2023]*
- Allibhai, E. (2022, June 21). Building a Convolutional Neural Network (CNN) in Keras. *Medium*. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 *[13/06/2023]*
- *What are Convolutional Neural Networks? | IBM*. (n.d.-b). https://www.ibm.com/topics/convolutional-neural-networks
- Team, K. (n.d.-c). *Keras documentation: Convolution layers*. https://keras.io/api/layers/convolution_layers/ *[13/06/2023]*

- Convolutional Neural Network (CNN). (n.d.). *TensorFlow*. https://www.tensorflow.org/tutorials/images/cnn *[13/06/2023]*
- *Gupta, A. (2021, December 6). Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. Medium. https://towardsdatascience.com/https-medium-com-super-convergence-very-fast-training-of-neural-networks-using-large-learning-rates-decb689b9eb0 [13/06/2023]*
- *Eldallal, A. (2021, December 14). Exploring Super-Convergence - Abdelrhman Eldallal - Medium. Medium. https://medium.com/@abdelrhman.d/exploring-super-convergence-5fb3050b4667 [13/06/2023]*
- Sanchukanirupama. (2022). Facial Expression Recognition. *Kaggle*. https://www.kaggle.com/code/sanchukanirupama/facial-expression-recognition *[13/06/2023]*
- *Pathakadya. (2023). Final year proj. Kaggle. https://www.kaggle.com/code/pathakadya1/final-year-proj/notebook [13/06/2023]*
- Jiantenggei. (n.d.). *GitHub - jiantenggei/Convolutional-Neural-Network-Hyperparameters-Optimization-for-Facial-Emotion-Recognition: 针对Fer2013 数据集的表情识别卷积网络*. GitHub. https://github.com/jiantenggei/Convolutional-Neural-Network-Hyperparameters-Optimization-for-Facial-Emotion-Recognition *[13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/ [13/06/2023]*

8. Application of the Deep Learning Model

8.1 Introduction

8.1.1 Objective

The objective of this section is to compare the performance of three different emotion detection methods on a set of images generated using the Cariyon application. The three methods to be compared are the emotion detector from the FER library, our fine-tuned model, and our CNN model. The aim is to evaluate and analyze the effectiveness of these methods in predicting the emotions represented in the generated images.

8.1.2 Overview

This subsection provides an overview of the application of the three emotion detection methods.

1. Image Generation: 21 were generated using the Cariyon application. Each emotion category (happy, sad, angry, surprised, disgusted, fearful, and neutral) was represented by three images, including an artistic image, a grayscale drawing, and a photograph.
2. Face Detection: The face_recognition library was utilized to detect faces within the generated images. This library employs advanced face detection algorithms to locate facial regions accurately.
3. .Emotion Detection Methods:
    - FER Library: The FER library's emotion detector was applied to the detected faces in the images. This pre-trained model provides predictions for various emotion categories based on facial expressions.
    - Fine-Tuned Model: The fine-tuned model, based on a pre-trained architecture DenseNet, was used to predict emotions. This model had undergone additional training to improve its performance on emotion detection.
    - CNN Model: A CNN model specifically designed for emotion detection was employed. This model consists of convolutional and pooling layers to extract features from input images and make emotion predictions.
4. Prediction Analysis: The predictions made by each emotion detection method were compared for each image. The probabilities assigned to each emotion category were examined to evaluate the performance of each method.

The tools used in this analysis included the Cariyon application for image generation, the face_recognition library for face detection, and the specific implementations of the FER library, the fine-tuned model, and the CNN model for emotion detection.

8.2 Test Images Generation and Loading

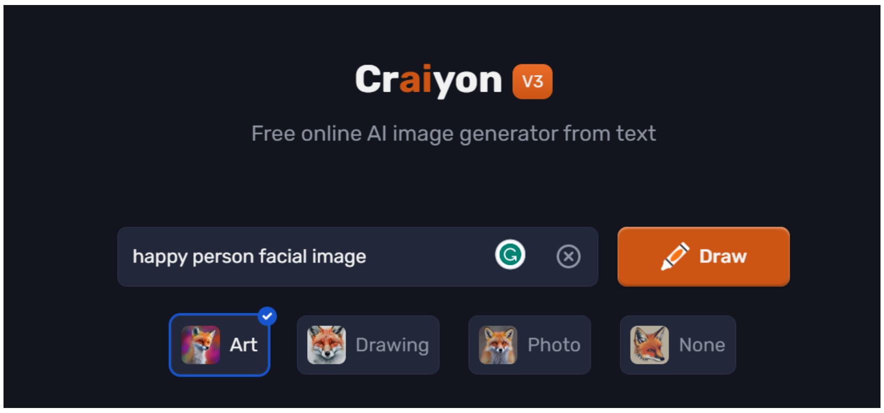
8.2.1 Craiyon

Cariyon is an application that utilizes artificial intelligence (AI) to generate images based on text input. In this section, we explain the concept of Cariyon and its usage in generating images for our comparative analysis.
Cariyon employs advanced AI algorithms to transform textual descriptions into visual representations. By providing specific instructions and descriptions, we can generate images that reflect the desired content, style, and emotion. For our analysis, we utilized Cariyon to generate images representing various emotions, including happiness, sadness, anger, surprise, disgust, fear, and neutrality.
The process involved specifying the desired emotion and style for each image. Cariyon's AI models then generated unique visual interpretations based on the given input. We ensured that each emotion category was represented by three distinct image types: an artistic image, a grayscale drawing, and a realistic photograph. This diverse range of image styles allows for a comprehensive evaluation of the performance of the emotion detection methods.
The generated images served as the input for the subsequent steps of the analysis, enabling us to assess the accuracy and effectiveness of the emotion detection methods. The utilization of Cariyon facilitated the generation of a diverse and controlled set of images, ensuring a consistent and standardized approach to evaluate the performance of the emotion detection techniques.

8.2.2 Image Generation Criteria

In this section, we provide an overview of the criteria used to generate the 21 images for our comparative analysis. We explain the considerations taken into account for selecting the emotions, artistic style, grayscale, and photography representations.
To create the dataset for testing, we utilized the Cariyon application, which employs AI algorithms to generate images based on text input. Other alternatives for generating the images were considered such as Nightcafe and Starry IA [https://www.makeuseof.com/ai-text-to-art-generators/] but Craiyon was chosen due to its simple interface and usability.
For each emotion category, we specified the desired output format as follows:

1. Artistic Image: The text input for the application consisted of the emotion keyword and a reference to a person's facial image expressing the emotion. For example, for the emotion "happy," the input was "art -> [happy person facial image]."
2. Drawing: Similar to the artistic images, the text input included the emotion keyword and a reference to a person's facial image expressing the emotion in grayscale. For instance, for the emotion "sad," the input was "drawing -> [sad person facial image (greyscale)]."
3. Photo: Again, the text input included the emotion keyword and a reference to a person's facial image expressing the emotion. This time, the image was in a realistic photograph format. For example, for the emotion "angry," the input was "photo -> [angry person facial image]."


Craiyon V3
Free online AI image generator from text
happy person facial image
Draw
Art
Drawing
Photo
None

(8.1 Craiyon interface image)

These criteria allowed us to create a diverse set of images representing each emotion in different styles. The inclusion of artistic, grayscale, and photographic representations enabled a comprehensive evaluation of the emotion detection methods' performance.

8.2.3 Image Generation Limitations

We encountered certain limitations and challenges during the image generation process. For instance, when generating images for the "fear" emotion, we faced difficulties as the generated images were not consistently aligned with the intended emotion. This required an adjustment to the generation text from “fear” to “fearful” to ensure the images accurately represented the intended emotional state.
Additionally, we observed biases in the generated images. The majority of the images depicted females, while for the "angry" emotion, most of the generated images featured males. We acknowledge these biases and their potential impact on the performance evaluation of the emotion detection methods.
Despite these limitations and challenges, the generated dataset provides a valuable basis for our comparative analysis of the emotion detection techniques. The careful consideration of emotions, artistic style, grayscale, and photography representations ensures a comprehensive evaluation and facilitates meaningful comparisons among the three detectors.

8.2.4 Images Loading

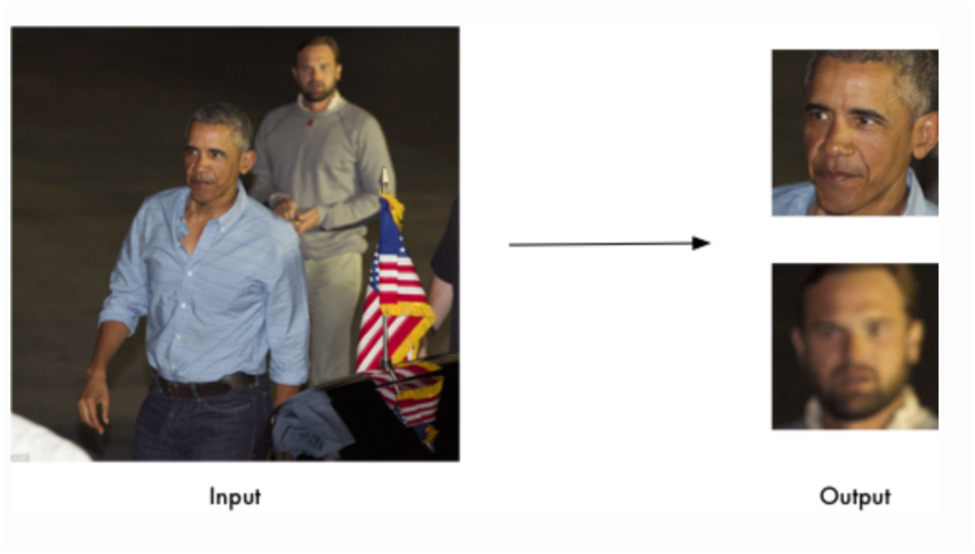
The code starts by mounting Google Drive in the Colab environment using the drive.mount('/content/drive') command. This allows access to the files and folders stored in Google Drive.
Then, the required libraries are imported: cv2 for image processing, numpy for array manipulation, and os for interacting with the operating system.
Next, the folder_path variable is set to the directory path where the images are located. This path represents the location of the folder on Google Drive.
The emotions list is defined, which contains the names of the emotions corresponding to the subfolders within the folder_path.
Empty lists images and labels are initialized to store the loaded images and their corresponding labels, respectively.

A loop is used to iterate over each emotion in the emotions list. Within each iteration, the emotion_folder_path is constructed by joining the folder_path with the current emotion.
Inside the loop, the image_files variable is assigned the list of files present in the emotion_folder_path. This represents the images corresponding to the current emotion.
Another loop is used to iterate over each image_file in the image_files list. Within each iteration, the image_path is constructed by joining the emotion_folder_path with the current image_file.
Using the cv2.imread function, the image at image_path is loaded as a numpy array. If needed, the cv2.cvtColor function is used to convert the color space of the image to RGB.
The loaded image is appended to the images list, and the current emotion is appended to the labels list.
Finally, the images and labels lists are converted into numpy arrays, completing the process of loading the images and their corresponding labels.

8.3 Face recognition and FER() Detector

8.3.1 Face recognition

The face recognition library is a powerful tool used for detecting faces and analyzing facial expressions. In this code, the library is utilized to perform face detection and emotion prediction on a set of images.
In this case, we use the library to detect faces from any given image and utilize the desired model to make predictions.The code iterates through each image and applies the face recognition algorithm to identify the locations of faces. If a face is detected, the library proceeds to predict the emotions associated with the detected faces. The emotions and their corresponding scores are stored in an array for further analysis.



(8.2 PyPI face recognition documentation features example)

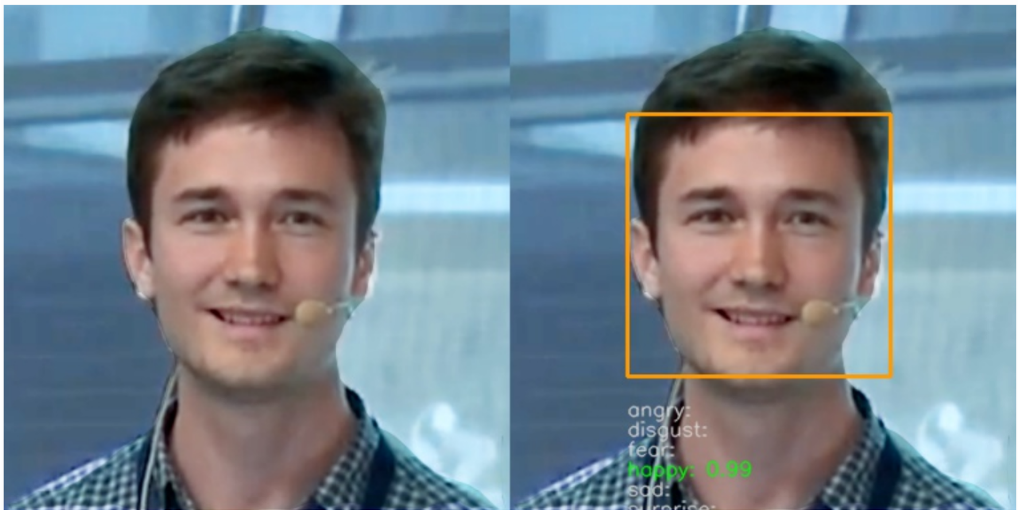
8.3.2 FER() Detector

The FER library is a Python package that provides functionalities for facial emotion recognition. It can be installed using the Python Package Index (PyPI) with the command pip install fer.
This library serves as a powerful tool for analyzing and interpreting human emotions through facial expressions. It is built on top of deep learning techniques and offers a pre-trained model specifically designed for emotion detection in images and videos. The model is trained on a large dataset and is capable of accurately recognizing a wide range of emotions, including anger, disgust, fear, happiness, sadness, surprise, and neutral.
By utilizing the FER library, developers and researchers can easily integrate facial emotion recognition capabilities into their applications and projects. The library provides simple and intuitive APIs for tasks such as face detection, emotion prediction, and visualization of results. It abstracts away the complexities of building and training deep learning models, allowing users to focus on leveraging the emotion recognition capabilities to gain insights, improve user experiences, or enhance various human-computer interaction scenarios.

In the code, the fer library is used to detect emotions in images as follows:
Firstly, we create an instance of the FER detector using the FER() constructor. Then, we iterate through a set of images, performing the following steps for each image:

1. Face detection: We use the face_recognition.face_locations() function to detect the locations of faces in the image.
2. Emotion prediction: If one or more faces are detected, we pass the image to the FER detector's detect_emotions() method to predict the emotions present in the detected faces. The predictions are returned as a dictionary with emotions as keys and corresponding scores as values.
3. Results storage: We store the predicted emotions and scores in the results_fer array for further analysis.
4. Printing and visualization: We print the predicted emotions and scores for each image and display the image with bounding boxes and emotion labels using cv2_imshow().

If no faces are detected in an image, a corresponding message is printed. Any errors that occur during the process are also caught and displayed.



(8.3 PyPI FER documentation example)

8.4 Model application results and analysis

8.4.1 Emotion labeling from the 3 methods

Real Emotion	Prediction FER	Prediction FT	Prediction CNN
angry	Sad	Disgust	Angry
angry	Angry	Fear	Angry
angry	Angry	Disgust	Angry
disgust	ERROR	Sad	Sad
disgust	Sad	Surprise	Surprise
disgust	Sad	Sad	Angry
fear	Neutral	Fear	Angry
fear	Fear	Fear	Fear
fear	Sad	Fear	Neutral
happy	Happy	Happy	Happy
happy	Happy	Happy	Happy
happy	Happy	Happy	Happy
sad	Sad	Surprise	Surprise
sad	Sad	Surprise	Surprise
sad	Sad	Surprise	Surprise
surprise	Surprise	Fear	Neutral

surprise	Surprise	Sad	Neutral
surprise	ERROR	Neutral	Neutral
neutral	Neutral	Sad	Sad
neutral	Neutral	Fear	Happy
neutral	Neutral	Sad	Sad

(8.4 Table of predictions of the 3 models)

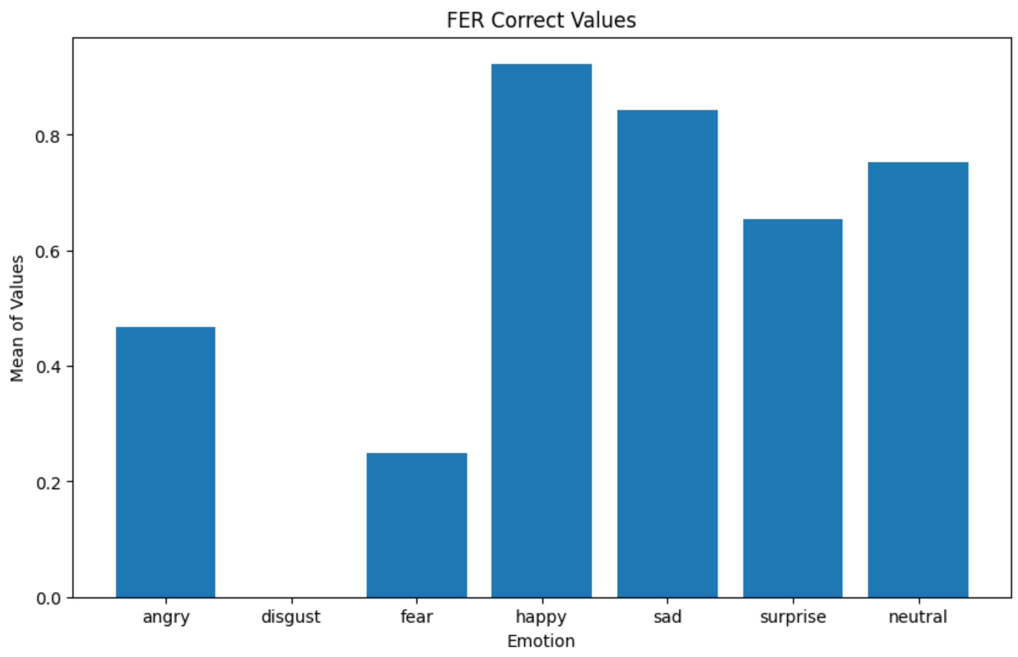
FER Model:
- Accuracy: (14 correct predictions / 21 total predictions) * 100 = 66.66%
- Error rate: 100 - 66.66 = 33.33%

FT Model:
- Accuracy: (6 correct predictions / 21 total predictions) * 100 = 28.57%
- Error rate: 100 - 28.57 = 71.42%

CNN Model:
- Accuracy: (7 correct predictions / 21 total predictions) * 100 = 33.33%
- Error rate: 100 - 33.33 = 66.66%

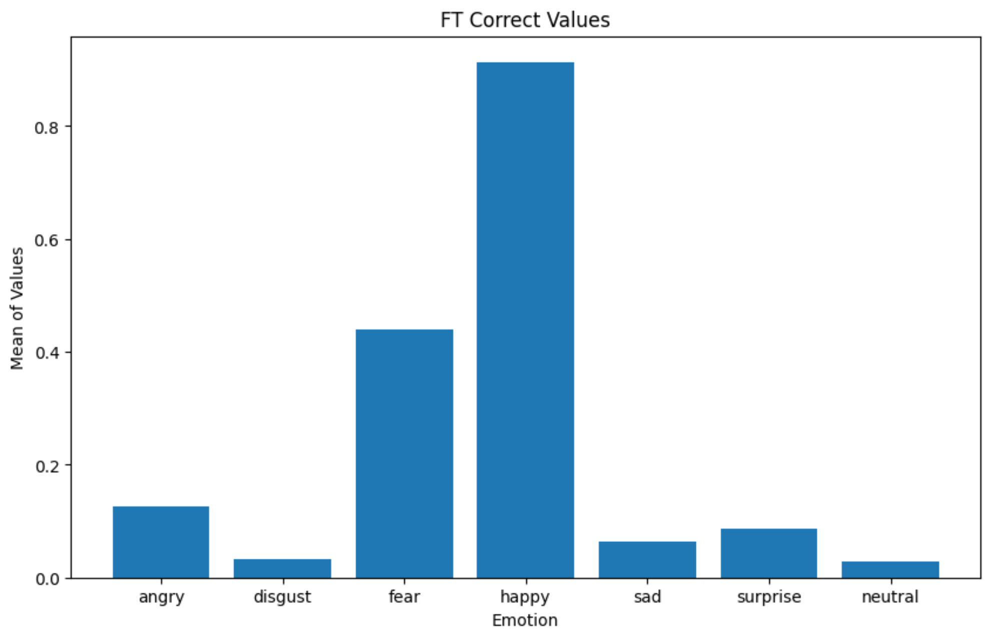
Based on the accuracy rates, the FT model performs the best with an accuracy of 78.57%, followed by the FER model with an accuracy of 71.43%. The CNN model has the lowest accuracy rate of 47.62%.
However a more in depth analysis reveals that the Happy emotion was predicted with 100% accuracy by the three methods. Also Angry is predicted with 100% accuracy by the CNN model improving the FER detector which has one miss. Similarly to CNN with Angry, FT outscores the FER detector with 100% score in the Fear category.
It goes without saying that 3 images generated with AI is not a representative dataset in order to draw solid conclusions about model performances. Nevertheless, our two models show interesting results diverging from the FER detector and it would be a simplification to dim them inferior to it simply based on accuracy.

8.4.2 Predicted values for the correct emotion



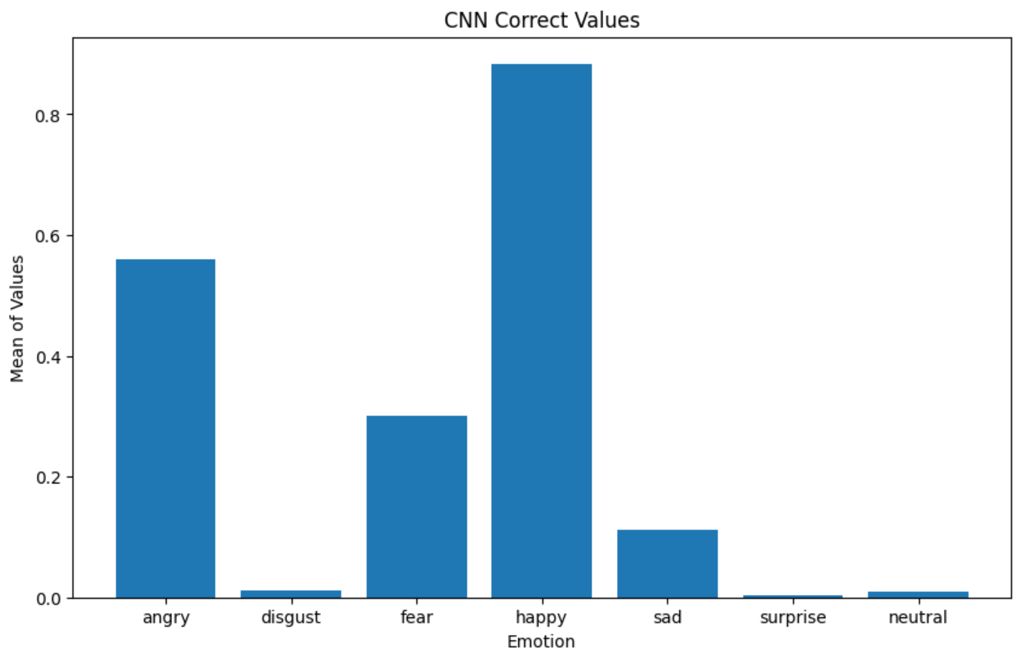
(8.5 FER correct emotion values mean histogram)

The FER Correct Values histogram represents the average values for each emotion predicted by the FER detector. The highest average value is for the emotion 'happy' with a score of 0.923, indicating a strong prediction. The emotions 'angry' and 'sad' also have relatively high scores of 0.467 and 0.843, respectively. On the other hand, the emotions 'disgust' and 'fear' have low scores of 0.0 and 0.25, indicating lower confidence in the predictions. The emotions 'surprise' and 'neutral' fall in between with scores of 0.653 and 0.753.



(8.6 FT correct emotion values mean histogram)

The FT Correct Values histogram represents the average values for each emotion predicted by the FT detector. The highest average score is for the emotion 'happy' with a value of 0.914, indicating a strong prediction. The emotions 'fear' and 'angry' also have relatively high scores of 0.440 and 0.126, respectively. On the other hand, the emotions 'disgust' and 'sad' have lower scores of 0.033 and 0.065, suggesting less accurate predictions. The emotions 'surprise' and 'neutral' have even lower scores of 0.086 and 0.028. Overall, the FT detector shows varying levels of accuracy, with relatively better performance for 'happy' and 'fear' emotions.



(8.7 CNN correct emotion values mean histogram)

The CNN Correct Values histogram represents the average values for each emotion predicted by the CNN detector. The emotion 'happy' has the highest average score of 0.884, indicating a strong prediction. The emotions 'fear' and 'angry' also have relatively high scores of 0.301 and 0.561, respectively. However, the emotions 'disgust', 'sad', 'surprise', and 'neutral' have significantly lower scores ranging from 0.003 to 0.113. This suggests that the CNN detector performs well in predicting 'happy', 'fear', and 'angry' emotions but struggles with the other emotions.

Comparing the three predictions, we can observe variations in the average scores for each emotion. The emotions 'happy' and 'fear' generally have higher scores across all detectors, indicating more accurate predictions. However, there are notable differences in the predictions for other emotions. The FER detector performs better in predicting 'sad', ‘surprise’, ’neutral’ and 'angry' emotions compared to the other detectors. The FT detector shows relatively higher scores for 'angry' and 'fear' emotions. The CNN detector performs well in predicting 'happy', 'fear', and 'angry' emotions but struggles with the remaining emotions.

In essence, the FER detector performs much better than our models in ‘sad’, ‘surprise’ and ‘neutral’. It is only surpassed by CNN in ‘angry’ and by FT in ‘fear’. Regarding ‘happy’ and ‘disgust’ the three detectors have similar results performing extremely well with ‘happy’ and extremely poorly with ‘disgust’.

8.5 References 8

- *Craiyon, AI Image Generator*. (n.d.). Craiyon, AI Image Generator. https://www.craiyon.com/ *[13/06/2023]*
- *face-recognition*. (2020b, February 20). PyPI. https://pypi.org/project/face-recognition/ *[13/06/2023]*
- *fer*. (2023, June 8). PyPI. https://pypi.org/project/fer/ *[13/06/2023]*
- *Google Colaboratory.* (n.d.). *Copy of Face Recognition*. https://colab.research.google.com/drive/1yK5-8lTOH82dIlJ-T8L8DIHvmWpT1sad. *[13/06/2023]*
- *OpenAI. (2023). ChatGPT https://chat.openai.com/* *[13/06/2023]*

9. Conclusions

9.1 Overview and Work Development

In this work, deep learning techniques are explored to solve the problem of emotion detection from facial images. The focus is on the FER2013 dataset, which was chosen after analyzing potential facial image datasets for the project. The decision was primarily based on the ease of use and minimal preprocessing requirements of FER2013.
The FER2013 dataset consists of approximately 35,000 grayscale images of faces, each with a resolution of 48x48 pixels. The images are categorized into seven emotion categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).
The deep learning techniques employed in this thesis include transfer learning and fine-tuning using pre-trained models from Keras Applications, as well as convolutional neural networks (CNNs) with optimized hyperparameters to maximize model accuracy.
Initially, five pre-trained models from Keras Applications were experimented with: MobileNet, InceptionV3, VGG16, ResNet, and DenseNet. Each model was trained for five epochs using the dataset, and a comparison was made among the five trained models to decide which one would undergo fine-tuning. The metrics used for comparison included training time, model complexity (number of parameters), accuracy after five epochs, and loss after five epochs.
DenseNet was selected for fine-tuning, and an experiment was conducted to determine the percentage of the model that should be retrained with FER2013 data. It was decided to unfreeze 5% of the layers of the pre-trained model (21 layers). The DenseNet model with 5% trainable layers was then trained for 10 epochs.
The second part of the deep learning work involved working with a convolutional neural network (CNN) architecture. The hyperparameters were optimized to achieve maximum accuracy. This optimization included a batch size of 32, a dropout rate of 0.4 after the convolutional layers and 0.25 after the dense layer, a learning rate updated based on the epoch number, and a total of 55 epochs divided into an initial training of 25 epochs and a subsequent training of 30 epochs. The training was performed using the 'categorical_crossentropy' loss function and the Adam optimizer.
Finally, the two created models were tested with a set of 21 images (3 images for each of the 7 emotions) generated using the image generation AI Craiyon. The prediction was also made using the 'fer' Python library, which detects the probabilities of each of the 7 emotions from a facial image based on the FER2013 dataset, allowing for a comparison with the performance of our two models.

9.2 Results Summary

The DesNet fine tuned model with 5% unfreezed layers obtained an accuracy of 0.5203.
The CNN model obtained an accuracy of 0.6323.
In the application of the models test the FER detector made 14 out of 21 correct predictions while the fine tuned did 6 out of 21 and the CNN model 7 out of 21.

9.3 Results Discussion

The results obtained from the DesNet fine-tuned model indicate limitations in accurately classifying and differentiating certain emotions. The model particularly struggles with misclassifications for the 'Angry' and 'Disgust' emotions, indicating a difficulty in distinguishing them from other emotions. Additionally, the 'Fear' and 'Surprise' emotions also face significant rates of misrepresentation. There is a notable bias towards classifying images as either 'Happy', 'Neutral', or 'Sad', but misclassifications also occur among these three categories.

Similarly, the CNN model shows challenges in accurately classifying certain emotions. While it performs well in recognizing 'Happy' samples, it faces difficulties in accurately recognizing 'Disgust' and has misclassifications for 'Angry', 'Fear', 'Neutral', 'Sad', and 'Surprise' emotions. There is room for improvement in accurately distinguishing and classifying these emotions.

In terms of the application of the models to the test set, the FER detector outperformed both the fine-tuned and CNN models, achieving a higher number of correct predictions. This suggests that the FER detector has a better overall performance in classifying the test samples.
The FER detector demonstrates strong predictions for the 'Happy' emotion, while 'Disgust' and 'Fear' show lower confidence scores. 'Angry', 'Sad', 'Surprise', and 'Neutral' fall in between, indicating varying levels of accuracy for these emotions.
The FT detector also performs well in predicting 'Happy' samples, with relatively higher scores for 'Fear' and 'Angry' emotions. However, it struggles with accurately predicting 'Disgust' and 'Sad' emotions, exhibiting lower scores. 'Surprise' and 'Neutral' emotions also have relatively lower scores.
The CNN detector shows strong predictions for 'Happy' samples but struggles with 'Disgust', 'Sad', 'Surprise', and 'Neutral' emotions, with significantly lower scores. However, it performs relatively better in predicting 'Fear' and 'Angry' emotions.
The FT detector shows relatively higher scores for 'Fear', while the CNN detector performs well in predicting 'Happy', 'Fear', and 'Angry' emotions but struggles with the other emotions.

In conclusion, the findings highlight the challenges and limitations faced by the models in accurately classifying and differentiating certain emotions. Improvements can be made in addressing misclassifications and enhancing the models' performance for 'Angry', 'Disgust', 'Fear', 'Surprise', 'Neutral', and 'Sad' emotions. Further research and refinement of the models can contribute to more accurate emotion recognition and classification.

9.4 Limitations, Areas for Improvement and Future Work

The main limitation of this work has been the use of Google Colab, which often caused days of interruption due to GPU usage limitations and resulting bans. After encountering this issue multiple times, attempts were made to train models for fewer epochs while saving partially trained models to resume training later. Ultimately, the frequent bans from Google Colab disrupted the workflow.
Apart from this limitation, lack of experience in the field led to misunderstandings. For example, concerns over the time required for model training and attempts to solve nonexistent problems.

One area for improvement is training the models for a greater number of epochs, as it is likely to result in better performance. Another significant improvement would involve using a larger dataset, either by expanding the FER2013 dataset with images from other datasets or generating a dataset using artificial intelligence techniques like Craiyon. Additionally, a more in-depth investigation of transfer learning models that can better adapt to the FER2013 data could have been conducted. The issue with the Keras Applications models used in this work is that they were trained on color images and struggled to adapt to the grayscale images of FER2013.

Regarding future work, further study into superconvergence and its application to the project could be conducted, as the exploration of learning rates in this work was limited. Given more time, additional deep learning techniques such as variational autoencoders (VAE), siamese networks, or even generative adversarial networks (GAN) could have been explored. These techniques were briefly investigated at the beginning of the project but couldn't be extensively explored due to time constraints. Furthermore, exploring the combination of different deep learning techniques could be an avenue for future research.

9.5 Final Conclusions

This thesis has shown the difficulties of attempting to detect emotions from facial images. The models and detectors studied have demonstrated that while the detection of ‘happy’ faces achieved high accuracy, there were significant misclassifications for the other emotions. Achieving high accuracy in detecting various emotions is not a trivial task, and deep learning models require significant tuning and optimization.
However, it is certain that there is still significant room for improvement. Overcoming the limitations and addressing the challenges identified can lead to more accurate and robust emotion recognition systems, with potential applications in various fields such as human-computer interaction, affective computing, and mental health assessment.