Emotion Recognition From Facial Expressions: A Preliminary Report

Tanja Jaschkowitz* Leah Kawka* Mahdi Mohammadi* Jiawen Wang*
{Tanja.Jaschkowitz,Leah.Kawka,Mahdi.Mohammadi,Jiawen.Wang}@campus.lmu.de

1. Introduction

Facial emotion recognition (FER) [?] is a topic of both ongoing debate and significant relevance in the fields of artificial intelligence and computer vision. In this short proposal, we aim to leverage several deep neural networks to analyze and interpret different human facial emotions.

The structure of this report is arranged as follows. In **??**, we provide the datasets we used, the model architecture we implemented. The preliminary evaluation results of our models are given in **??**. In **??** describes the optimization strategies we have already used and plan to investigate in the coming weeks. Finally, an overview of our time schedule for the entire final project is given in **??**. Our code and supplementary material are available at https://github.com/werywjw/SEP-CVDL.

2. Approach

2.1. Dataset Acquisition and Processing

To initiate the training, we acquired the databases FER+, RAF-DB, CK+, TFEID and ISFA from public institutions and a GitHub-repository. Based on these databases we created a dataset by augmentation to increase variety, full details of augmentation is given in **??**. In terms of the illustrating content of the used pictures, we exclusively used human faces representing 6 emotions. We generalized a folder structure annotating the labels 1 (surprise), 2 (fear), 3 (disgust), 4 (happiness), 5 (sadness), and 6 (anger). Beside the original format we set a standard, resizing all images to 64x64 pixels and saved them as JPEGs.

The first test results in **??** are aggregated from the database FER+. The images are converted to greyscale with three channels as our original convolutional neural network (CNN) is designed to work with three-channel inputs and add random augmentation.

2.2. Model Architecture

We implemented from scratch an emotion classification model with four convolution layers at the very beginning. Following each convolutional layer, batch normalization is applied. This stabilizes learning by normalizing the input to each layer. Then three linear layers are applied to extract features to the final output. We also add a `dropout` layer to prevent overfitting. The activation function used after each layer is Rectified Linear Unit (ReLU), since it introduces the non-linearity into the model, allowing it to learn more complex patterns. In order to find the best hyperparameter configuration (see **??** for details) of the model, we utilize the parameter grid from sklearn [1].

Models	Accuracy (Train)	Accuracy (Test)	Accuracy (Vali)
CNN (Baseline)	66.3	75.2	52.6
CNN (SE)	74.3	79.9	59.6
CNN (SE-Aug)	84.6	79.5	62.8
CNN (SE+Residual)	71.5	78.9	56.4
ResNet18 [?]	76.8	79.8	60.3

Table 1. Accuracy (%) for different models in our experiments

Hyperparameter	Configuration
Learning rate	{0.1, 0.01, 0.001, 0.0001}
Batch size	{8, 16, 32, 64}
Dropout rate	{0.5}
Epoch	{10, 20, 30, 40}
Early stopping	{`True`, `False`}
Patience	{5}

Table 2. Explored hyperparameter space for our models

3. Preliminary Results

For evaluation, we use the metric accuracy. As seen in **??**, we report all the training, testing, and validation accuracy in % to improve the performance of our models. The loss function employed for all models is cross-entropy, which is typically for multi-class classification. The best results of the performance of the model with respect to loss and accuracy are depicted in **??**.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html

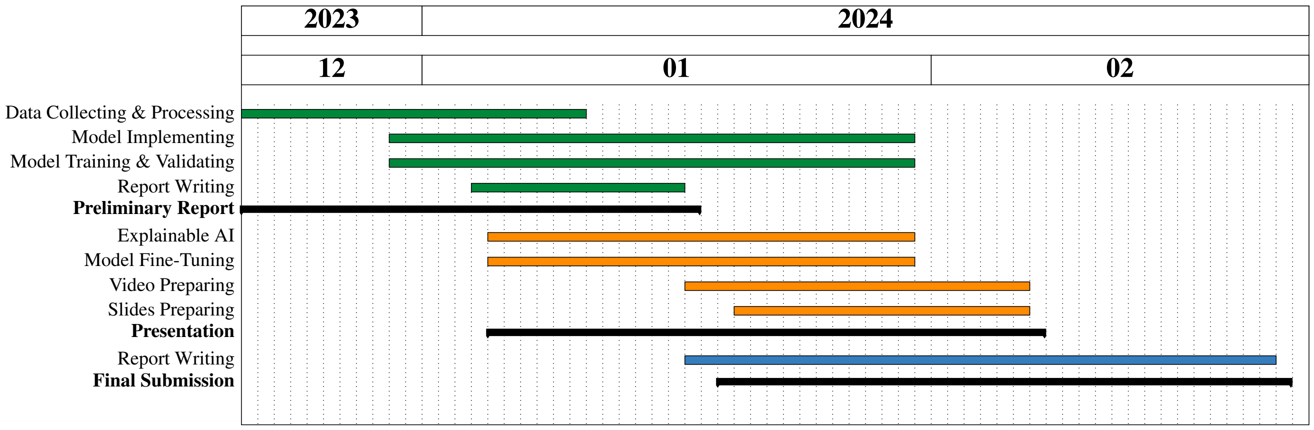

Figure 1. Overview of the schedule for the final project

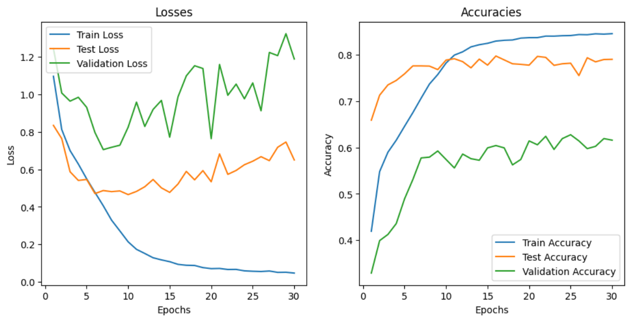

Figure 2. Empirical results in terms of the loss and accuracy on different training epochs

4. Optimization Strategies

We increase the depth of the network by adding some convolutional layers to learn more complex features. We also add the residual connections to help the training of deeper networks more efficiently, as they allow gradients to flow through the network more easily, improving the training for deep architectures. Moreover, we add squeeze and excitation (SE) blocks to apply channel-wise attention. In the coming weeks, we will focus on the following the tasks **??** and **??**.

4.1. Augmentation

In machine learning and artificial intelligence, augmentation stands as a transformative technique, empowering algorithms to learn from and adapt to a wider range of data. By introducing subtle modifications to existing data points, augmentation effectively expands the dataset, enabling models to generalize better and achieve enhanced performance. As models encounter slightly altered versions of familiar data, they are forced to make more nuanced and robust predictions. With this process we aim to prevent overfitting, a common pitfall in machine learning. Additionally we guide the training process to enhance recognition and handling of real-world variations. During the project we pursue various approaches. We are implementing different combinations of functions from the `pytorch.transforms` library and testing already established filters that have been developed, in other research contexts. We create various replications of existing photos by randomly altering different properties such as size, brightness, color channels, or perspectives.

4.2. Explainable AI Grad-CAM

Class Activation Mapping (Zitat: Zhou 2016) is a visualization technique designed to highlight the regions of an image or video that contribute the most to the prediction of a specific class by a neural network. Besides proposing a method to visualize the discriminative regions of a classification-trained convolutional neural network (CNN), the authors also use this method to localize objects without providing the model with any bounding box annotations. The model just learns the classification task with class labels and is then able to localize the object of a specific class in an image. CAM provides valuable insights into the decision-making process of deep learning models, like CNNs. We follow up Grad-CAM (Zitat: 2019), introduced as a technique that is easier to implement with different architectures. This task will be implemented by using the libraris of Pythorch and OpenCV. [2].

4.3.

We also wrote a script which takes a folder path as input and iterates through the images inside a subfolder. The output is a csv file representing the corresponding classification scores.

[2] https://opencv.org

Author Contributions

Equal contributions are listed by alphabetical order of surnames. Every author did the literature research and contributed to the writing of the paper.

- **Tanja Jaschkowitz** implemented the model architecture, training and testing infrastructure, and CSV file aggregations.
- **Leah Kawka** collected the training data, prepared data processing, implemented augmentation, and ran the results. She also takes part in the explainable AI, specificly in implementing the video-green squares.
- **Mahdi Mohammadi** implemented the augmentation, and did the research searching.
- **Jiawen Wang** implemented the model architecture, training and testing infrastructure, and optimization strategies. In the specific writing part, she also checked and aggeregated this report from other team members.

Acknowledgements

We are deeply grateful to our advisors **Johannes Fischer** and **Ming Gui** for their helpful and valuable support during the entire semester. We also thank **Prof. Dr. Björn Ommer** for providing this interesting practical course.