Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Bachelor's diploma thesis

in the field of study Computer Science and Information Systems

Project of an application implementing deep learning models for emotion recognition based on facial expression images

Andrey Maximenko

student record book number 323864

Ivan Stankevich

student record book number 317134

thesis supervisor
dr hab. inż. Jerzy Balicki, prof. ucz.

WARSAW 2025

Abstract

Project of an application implementing deep learning models for emotion recognition based on facial expression images

This study aims to design an application that uses self-developed deep learning models for emotion recognition based on human face images and videos. The models were trained to recognize up to 8 emotions: anger, contempt, disgust, fear, happiness, sadness, surprise, and neutrality. To develop these models, several datasets – FER2013, KDEF, Natural Human Face Images and their mix – were used. For each dataset, two models were implemented using different frameworks, TensorFlow and PyTorch, and their capabilities were comparatively analyzed in this work. The performance of each model is evaluated using different metrics like F1 score and accuracy, with all models achieving promising results, exceeding 60% accuracy. The application was developed using ReactJS framework with Electron and Python with FastAPI library.

Keywords: Convolution Neural Network, Emotion Recognition, Transfer Learning, Machine Learning, Facial Expression Images

Streszczenie

Projekt aplikacji implementującej modeleuczenia głębokiego do rozpoznawania emocjina podstawie zdjęć ekspresji twarzy

Przedstawiona analiza ma na celu zaprojektowanie aplikacji wykorzystującej autorskie modele głębokiego uczenia do rozpoznawania emocji na podstawie obrazów i filmów przedstawiających ludzkie twarze. Modele zostały wytrenowane do rozpoznawania 8 emocji: złości, pogardy, obrzydzenia, strachu, radości, smutku, zaskoczenia i neutralności. Do opracowania tych modeli wykorzystano kilka zbiorów danych – FER2013, KDEF, Natural Human Face Images oraz ich mieszanki. Dla każdego zbioru danych zaimplementowano dwa modele w różnych frameworkach – TensorFlow i PyTorch – a ich możliwości zostały porównane w niniejszej pracy. Wydajność każdego modelu została oceniona przy użyciu różnych metryk, takich jak wynik F1 i dokładność, a wszystkie modele osiągnęły obiecujące wyniki, przekraczając dokładność na poziomie 60%. Aplikacja została opracowana przy użyciu frameworka ReactJS z Electronem oraz Pythona z biblioteką FastAPI.

Słowa kluczowe: Konwolucyjna Sieć Neuronowa, Rozpoznawanie Emocji, Transfer Learning, Uczenie Maszynowe, Obrazy Mimiki Twarzy

Zał. nr 1B do protokołu z posiedzenia Komisji Programowej kierunku Inżynieria i Analiza Danych Wydziału Matematyki i Nauk Informacyjnych Politechniki Warszawskiej z dnia 12.06.2024

Zapisy związane z użyciem narzędzi informatycznych do przygotowania tekstu pracy i oprogramowania – obligatoryjne w przypadku wszystkich prac dyplomowych inżynierskich i magisterskich kierunku IAD

Each thesis must feature the following declaration: / *Każdy dokument pracy dyplomowej musi zawierać następującą deklarację:*

We hereby declare that: / *Niniejszym oświadczamy, że:*

1) We have not used IT tools to generate the content of the manuscript of this thesis[1]. / *Nie używaliśmy narzędzi informatycznych do generowania treści niniejszej pracy.*
2) We have not IT tools to generate the code of the software developed for this thesis. / *Nie używaliśmy narzędzi informatycznych do generowania kodu programów stworzonych na potrzeby niniejszej pracy.*
3) We take full responsibility for all content in the thesis, including both the manuscript and the software developed for it. */ Bierzemy pełną odpowiedzialność za zawartość niniejszej pracy, która składa się zarówno z jej tekstu, jak i stworzonego na jej potrzeby oprogramowania.*

If any IT tools were used to generate the manuscript and/or the software of the thesis, please fill in the table below: */ W przypadku wykorzystania narzędzi informatycznych do generowania tekstu lub kodu, należy wypełnić poniższą tabelę:*

The scope of the use of IT tools to generate manuscript[2] *(zakres użycia narzędzi informatycznych do generowania tekstu pracy)*	The scope of the use of IT tools to generate software code *(zakres użycia narzędzi informatycznych do generowania kodu)*
… see below */ zob. niżej ...*	… see below */ zob. niżej ...*

Please consider at least the following categories / *Należy uwzględnić co najmniej następujące kategorie*:

[1] By IT tools, tools generating text, multimedia content and software code are meant, including but not limited to the tools relying on Large Language Models, such as ChatGPT, Gemini, Copilot, etc.. In statements 1 and 2, please mark the option matching the way your diploma project has been prepared. / *Przez narzędzia informatyczne rozumiane są narzędzia generujące tekst, treści multimedialne i kod oprogramowania, w tym między innymi narzędzia bazujące na modelach językowych, takie jak ChatGPT, Gemini, Copilot itp. W stwierdzeniach 1 i 2 proszę zaznaczyć opcję odpowiadającą sposobowi przygotowania pracy dyplomowej.* [2] Please note that authors of a group thesis are jointly responsible for the outcomes of their works. */ Wszyscy autorzy prac wieloosobowych są współodpowiedzialni za wyniki umieszczone w pracy.*

- Creating or modifying images, sound, and video used in the thesis (not allowed, unless mandatory to fulfil the objectives of the thesis, if used it should be explicitly declared for each generated content)[3] */ Tworzenie lub modyfikacja obrazów i innych danych audiowizualnych użytych w pracy (niedozwolone, chyba że konieczne ze względu na zakres pracy – w takim przypadku, każdy wygenerowany element musi być wyraźnie oznaczony i opisany)*
- Generating novel algorithms and methods (not allowed, unless mandatory to fulfil the objectives of the thesis)[4] */ Generowanie nowatorskich algorytmów lub innych metod (niedozwolone, chyba że konieczne ze względu na zakres pracy)*
- Generating text of the manuscript (strongly discouraged)[5] */ Generowanie tekstu pracy (mocno odradzane)*
- Generating the code of the software (not recommended, except for generic tasks such as loading data from CSV files)[6] / *Generowanie kodu (niezalecane, za wyjątkiem prostych zadań typu ładowanie plików CSV)*

Editing text for language improvements, performing editorial tasks (such as generating LaTeX code for printing out tabular data), and grammar enhancement do not have to be reported and are acceptable. */ Edycja tekstu pod względem językowym, dokonywanie standardowych działań redakcyjnych (generowanie kodu LaTeX do wypisywania tabel) lub poprawa gramatyki nie muszą być deklarowane – uznajemy je za akceptowalne.*

---

[3] Note that multimedia content generated with IT tools, including AI tools is not considered to be protected by copyright law and owned by the person who generated it. */ Treści multimedialne generowane przez narzędzia informatyczne nie są chronione prawem autorskim i nie uznaje się, że są one autorstwa generującego go użytkownika.*

[4] The objective of the thesis is to confirm the ability of the authors to generate novel solutions. */ Celem pracy jest wykazanie, że autorzy umieją stworzyć nowatorskie rozwiązania samodzielnie.*

[5] Note that there are multiple risks related to generating the content of the manuscript: / *Należy zwrócić uwagę na wiele zagrożeń związanych z generowaniem treści pracy przy użyciu narzędzi informatycznych:*

- The text generated by IT tools, including AI-based tools may infringe the copyrights of other parties. A possible overlap can be detected by the anti-plagiarism tools and may even cause the work to be rejected based on the recommendations of the anti-plagiarism system. */ Wygenerowany tekst może naruszać prawa autorskie innych stron. Programy antyplagiatowe mogą potem odrzucić tekst pracy jako zbyt podobny do innych źródeł.*
- Moreover, IT tools, e.g., relying on large language models may generate incorrect content and not reflect the state-of-the-art in the field of study that could be better addressed by analysing the most recent works published in the field. */ Narzędzia informatyczne oparte na modelach językowych mogą generować nieprawdziwą treść oraz nie odzwierciedlać aktualnego stanu wiedzy w dziedzinie, który można tylko poznać studiując samodzielnie literaturę przedmiotu.*
- Questions related to individual statements of the diploma thesis may be asked during the defence of the diploma thesis related to questionable statements. */ Pytania dotyczące zagadnień będących przedmiotem powyższej deklaracji mogą zostać zadane na obronie pracy.*

[6] The objective of the thesis is also to build and confirm programming skills. / *Innym celem pracy dyplomowej jest nabycie i wykazanie umiejętności programistycznych.*

Work Division

Author	**Project Contribution**	**Thesis Contribution**
Andrey Maximenko	Data collection and preprocessing, Application development, testing and deployment	Introduction, Datasets, Application
Ivan Stankevich	Models development, training and results evaluation	Deep Learning Models, Numerical Experiments, Summary

Contents

Introduction

Human emotions play a central role in communication, influencing everything from personal relationships to consumer behavior. Recent breakthroughs in deep learning architectures open the door for reliable emotion recognition models that can be used in such fields as healthcare [1], entertainment [2] and many more. Despite various existing solutions, the field still faces significant challenges due to model overfitting, data imbalance and the complexity of facial expressions[3].

In this thesis we aimed to develop custom deep learning models, each trained on a specific dataset that was chosen for this work: FER2013, KDEF, Natural Human Face Images, and a mixed dataset. For each dataset two Convolutional Neural Networks (CNNs) were trained, one in TensorFlow framework and another in PyTorch. The reason behind that is to compare two of the most popular Machine Learning frameworks in their computational capabilities, ease of use and training efficiency. The models performance is evaluated as well, using metrics such as accuracy and F1 score.

The final product and central contribution of this research is a unified application designed for real-time emotion recognition based on images or videos. All eight models are integrated in this application allowing the user to choose between them. The application is designed in a ReactJS wrapped in Electron making it a desktop interface. Behind the scenes for image pre-processing and communication with models, we used a Python FastAPI.

Following this introduction, the work is divided in four main chapters. Chapter 1 describes the above mentioned datasets, what challenges and strengths each of them present, as well as the augmentation techniques used to achieve uniform distribution of the classes in all of them. Chapter 2 discusses the architectural design of the deep learning models. Chapter 3 presents a in depth documentation of the final application. And Chapter 4 interprets the experimental results achieved by our models.

1 Datasets

This work involves three different publicly accessible and known datasets that are often used in the creation of emotion recognition models. These datasets provide a wide variety of facial expressions and emotions, captured under different conditions. Those sets are: Karolinska Directed Emotional Faces (KDEF), Natural Human Face Images for Emotion Recognition (NHFI), Facial Expression Recognition 2013 (FER2013).

1.1 Karolinska Directed Emotional Faces

The KDEF dataset consists of 4900 pictures of 70 individuals (35 males and 35 females) displaying 7 distinct emotional expressions: afraid, angry, disgusted, happy, neutral, sad, and surprised [4]. As indicated by the name, the pictures were taken in the controlled and directed studio environment.

All expressions were captured from 5 different angles. Each subject had specific instructions and practiced their poses for one hour. The photos were taken during 2 separate sessions, resulting in each individual being photographed twice for all 7 emotions. A sample from the dataset is presented in Figure 1.

The following criteria were applied in the selection process: age between 20 and 30 years, no facial hair, accessories, or makeup. During the shooting all wore an identical gray T-shirt.

While the KDEF dataset provides a well-structured and balanced collection of emotional expressions, some challenges arise:

- Controlled environment: models trained on KDEF might struggle when applied to unposed pictures. The specific lighting conditions and practiced posing deprive model from training on spontaneous and ambiguous emotions. This can result in model performing excellently on the test set but showing poor results on real-world data that is proved by many researches [5].

- Age limit: all participants are in their 20s, which limits the model's exposure to some facial features (like wrinkles) that are common to other demographics.

- Dataset size: dataset's design makes it harder to scale it while maintaining the same quality. It results in KDEF being only 4900 images

in size which can be insufficient fir training an effective model.

Figure 1: A sample from KDEF dataset

Despite these challenges, the nature of the dataset offers also some significant advantages that are rarely available in other datasets, for which we decided to use KDEF in our work. The most notable one being the queal representation of emotions with 700 images. This allows to avoid augmentation of the set to achieve uniform distribution.

Leveraging its balanced nature, we divided dataset into training and test subsets. 70 pictures per emotion were randomly selected to form a test set containing 490 images. The remaining 4410 images were used for training.

To optimize the dataset for training a deep learning model, the images were resized from 562x762 pixels to 224x224 format that is better suitable for convolutional neural networks.

1.2 The Facial Expression Recognition 2013

The FER2013 dataset was introduced by Pierre-Luc Carrier and Aaron Courville during the ICML 2013 Workshop on Challenges in Representation Learning. The dataset was created by collecting images from the internet. Face detection and cropping were applied to create standardized grayscale images of 48x48 pixels in size. FER2013 includes seven distinct emotions: angry, disgust, fear, happy, sad, surprise, neutral [6]. A sample of the dataset is shown in Figure 2.

Figure 2: FER2013 dataset sample

The set is divided into training and testing subsets. Two version of the test set exists, public and private. In this work only the second one was used.

The training set consists of 28,709 examples, while the test set contains 3589 images. This provides a substantial amount of data for training and evaluating models.

However, the FER2013 is unbalanced. Most classes are distributed between 11% and 17% of the dataset, except for "happy" emotion which takes 25% of the data, and "disguted", which is heavily underrepresented at only 1.5%. The detailed emotion distribution across both subsets can be seen in Figure 3.

Even though the FER2013 dataset introduces some challenges like data imbalance and low resolution of images it was chosen as one of the datasets in our work due to several crucial reasons:

- Known performance: FER2013 has become a widely used benchmark

for training top-notch emotion recognition models, often achieving excellent performance [7].

- Size: this dataset provides substantial amount of data, even though unbalanced. It is the largest dataset used in this research.
- Publicly accessible: the dataset is officially available for educational purposes, meaning there is no need to rely on the authors approval.

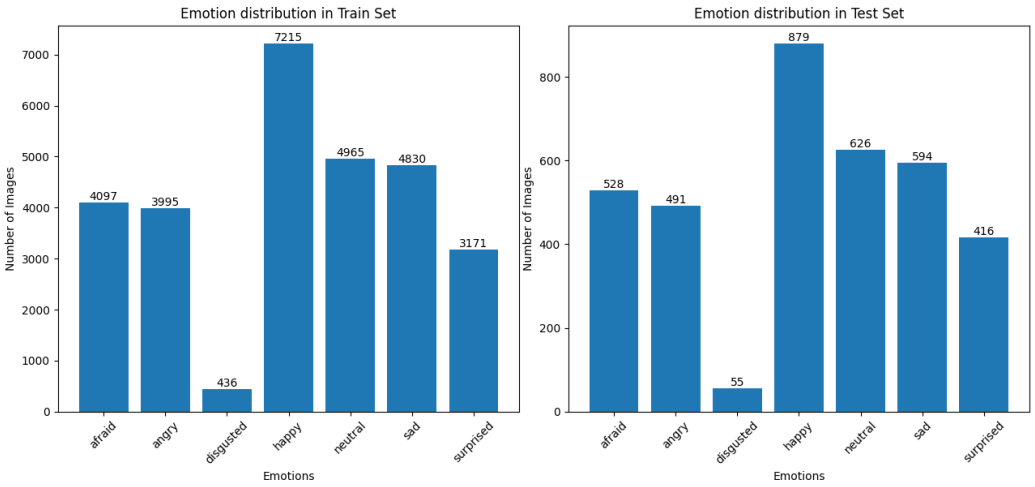


Figure 3: FER2013 emotion distribution

To address the imbalance, data augmentation techniques were applied and will be described in the later section. The goal was to increase all classes to match the ”happy” emotion’s sample size. In the result each emotion was represented by 7215 images in the training set and 879 images in the test set.

1.3 Natural Human Face Images

The Natural Human Face Images for Emotion Recognition (NHFI) dataset was collected as an attempt to create additional source of data for training of emotion recognition models along side other popular datasets such as FER2013 and KDEF. Limited information about the set is available, with primary source being the official Kaggle page posted by the user Sudarshan Vaidya in 2020 [8].

The images of the dataset were downloaded from different websites like Google and Unsplash. They are presented in 8 different emotion categories: anger, contempt, disgust, fear, happiness, neutrality, sadness and surprise. This makes NHFI dataset unique in our studies as the only data collection that has 8th emotion, contempt.

Despite its relatively recent release and the limited number of studies validating its standalone effectiveness, the NHFI dataset was chosen in this work due to the following key reasons:

- High image resolution: unlike FER2013 and many others, the images in this dataset are saved in 224x224 grayscale format that can help model learning more facial features and details. Combined with diverse image sources, the model has a huge potential to show promising results in real world scenario. The sample of the dataset is provided in Figure 4.

- Contribution to the research: by choosing this dataset, the study serves to fill a gap in the current literature by exploring set usability for emotion recognition.

Figure 4: NHFI dataset sample

The set consists of 5558 manually annotated images being highly unbalanced (see Figure 5). After later augmentation, dataset was expanded to 11200 images, with each emotion class represented by 1400 images. Following this augmentation, the set was split into train and test subsets, containing 10080 and 1120 images, respectively.

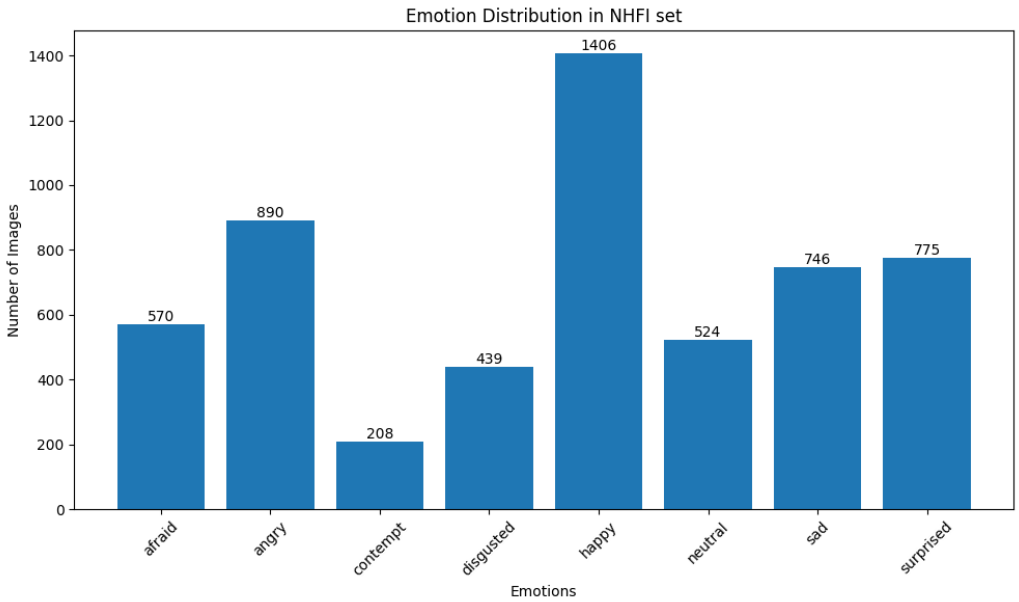


Figure 5: NHFI emotion distribution

1.4 Augmentation of datasets

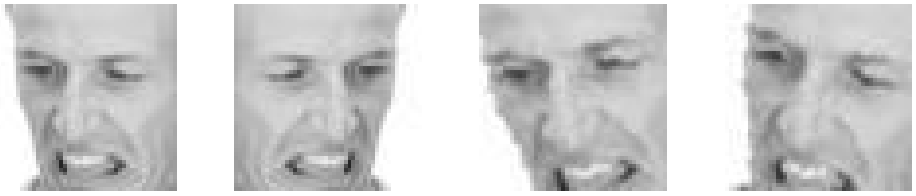
As established, two datasets, FER2013 and NHFI, required an augmentation to achieve uniform distribution among classes. The most common and useful techniques [9] were applied in this specific order:

- Mirroring: each image in the class was horizontally flipped to create a mirrored copy.
- Rotation: pictures were rotated by 10 degrees to the right and left. To avoid black pixels in the corners, pictures were slightly zoomed in.
- Zooming: if the class still lacked the required amount of data, unrotated images would also get a copy with slight zooming.
- Brightness adjustment: as the resort, the brightness of images would be increased or decreased by 10%.

Most classes reached the required number of images after the mirroring or rotation steps. However, ”contempt” in NHFI dataset required additional zooming augmentation. Similarly, the ”disgusted” class in FER2013, as the

most underrepresented class, after zooming still lacked a lot of images, that's why a step with brightness correction was added.

The specified order was design to minimize the risk of introducing problems like bias into the dataset. Mirroring is considered to be the most neutral and safe augmentation method [9]. While rotation is usually not a problem, the required zooming might remove some critical features. Brightness correction, if applied only to some classes, can lead the model to mistakenly associate lighting changes with specific emotions. However, in our case, this is higly unlikely, since FER2013 consists of images from different sources and different lighting conditions.



Original Mirrored Rotated left Rotated right

Zoomed Brightened Darkened

Figure 6: Augmentation examples

1.5 Mixed dataset

As a final step in the data collection and preprocessing, a new dataset was constructed by combining images from the three previously mentioned datasets. The goal was to create a diverse dataset that will benefit from integrating images from different sources that consists of images that are both captured under controlled conditions and spontaneous pictures sourced from the internet [10].

The set was designed with the following proportions: 20% of the images

were taken from the KDEF set, 40% each take NHFI and FER2013 sets. All of the images from KDEF and NHFI were included, with the exception of the 'contempt' class present only in NHFI and that was not added to the mixed set. Meanwhile, only a subset of FER2013 was taken to avoid introducing bias due to its larger size.

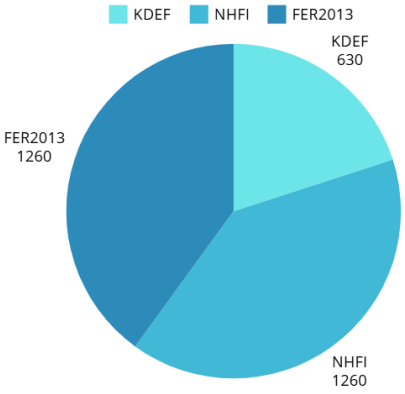


Figure 7: Mixed Set contribution to each class

In total, the set contains 22050 images in the training subset and 2450 images in the training subset. With all datasets being uniform after augmentation, we get 3150 and 350 images per class in the relative subsets.

1.6 Remarks and conclusions

Each dataset used in this work presents its own difficulties and advantages. All of them differ in some aspects from others; it can be the size of the set, the image resolution, the number of emotions, or the image sources.

These differences provide a solid foundation for a diverse and deep analysis. However, this diversity also introduces more challenges that are unique for each dataset.

Nevertheless, the variety is important in a work like ours, as it allows us test models across different scenarios and conditions. By leveraging the strengths of each dataset and overcoming all possible difficulties, we aim to achieve more comprehensive results in this research.

2 Deep Learning Models

In this chapter, we discuss the deep learning architectures and frameworks employed for emotion recognition. We begin by highlighting our choice of frameworks (PyTorch and TensorFlow) and explaining why VGG16 was selected as our backbone. After that, we detail the transfer learning strategy and data preprocessing techniques.

2.1 Using PyTorch and TensorFlow in Python Applications

We used convolutional neural networks (CNNs) for our models because they are well-suited for tasks such as emotion recognition from facial expressions. In this project, we adopted a unified approach to train CNN-based models on four different datasets mentioned earlier.

For each dataset, we build two models in parallel: one in **TensorFlow** [13] and one in **PyTorch** [12]. All eight models share the same backbone architecture (VGG16, see Section 2.2), making them comparable "apple-to-apple" in architecture, hyperparameters, and training regimes. With the help of that, we can evaluate how these frameworks differ in training speed, ease of implementation, and performance.

TensorFlow. TensorFlow tends to use a static computational graph by default (although Eager Mode is available), and it has a large ecosystem:

- **Deployment tools:** TensorFlow Lite, TensorFlow Serving
- **Visualization and monitoring:** TensorBoard
- **Serialization:** Easy model export to `.pb` or `.h5` formats

PyTorch. PyTorch relies on a dynamic computation graph, making it very "Pythonic" and flexible:

- **Ecosystem synergy:** Integrates well with the Hugging Face library [14]
- **Debugging features:** `torch.autograd` and an imperative style for simpler prototyping

- **Research usage:** Widely adopted for cutting-edge research projects

By using the same training pipeline in both frameworks, we can compare differences in runtime stability, resource usage, and final classification performance. Moreover, in our final application design, *all* models are accessible via a *single* API endpoint, with a parameter specifying which model/framework to use. This allows the user to switch easily between PyTorch or TensorFlow implementations as needed

Note on CNN History and Adam Optimization. Deep convolutional neural networks are widely used because of well known AlexNet [17], which achieved really good results on ImageNet. Subsequent architectures such as Inception [18] and ResNet [19] further improved classification benchmarks. In our project, we rely on the Adam optimizer [20] for its computational efficiency and ease of tuning. Although there is more advanced designs, such as the original VGG16 model by Simonyan and Zisserman [21], we found that a simplified VGG16-based approach gives us an ideal balance between performance and replicability for both PyTorch and TensorFlow frameworks.

2.2 Using the VGG16 Architecture

Among the many powerful CNN architectures available, we selected **VGG16** because it consistently demonstrates good performance, has widely available pretrained weights, and is relatively easy to replicate between frameworks. Before that we experimented with custom architectures or ResNet-based models, but they did not yield stable results under our constraints (limited GPU time, overfitting, etc.). Thus, we found that VGG16 shows the best balance between simplicity and performance. VGG16 is characterized by:

- A series of small 3×3 convolution filters stacked with increasing depth.
- Fully connected layers at the end (a typical feature of earlier CNN designs).
- Simplicity in architecture, which makes debugging straightforward and removes extraneous factors when comparing frameworks.

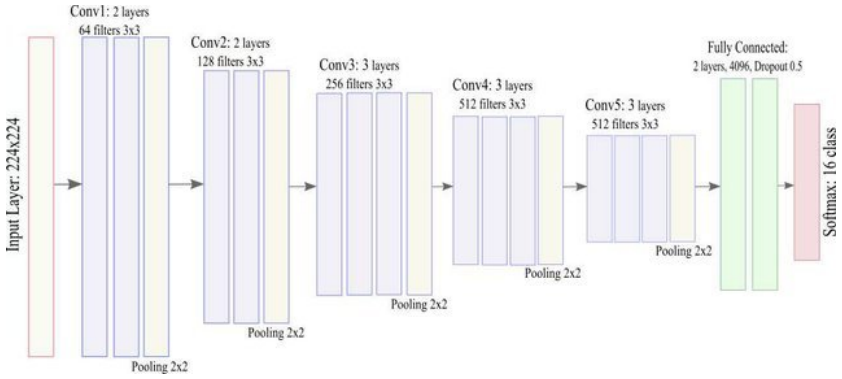


Figure 8: VGG16 architecture showing its layers and connections.

Moreover, VGG16's intermediate representations often capture essential facial features (edges, contours, eye and mouth shapes) crucial for emotion recognition tasks. By loading the pretrained ImageNet weights, we can fine-tune the network instead of training from scratch, saving time and reducing overfitting risks.

2.3 Two-Phase Transfer Learning

To speed up convergence and improve generalization, we adopted a **two-phase transfer learning** procedure for each dataset. Specifically, we split our VGG16 training into two main stages:

1. **Phase 1: Train Only the Classification Head**
    - Initialize VGG16 with **pretrained ImageNet weights**, but **freeze** all convolutional layers so they do not update during backpropagation.
    - Add a custom classification head, e.g.:
        - **GlobalAveragePooling2D** (reducing spatial dimensions)
        - One or more **Dense** layers (e.g., Dense(256) with ReLU)
        - **Dropout** for regularization (we used 0.4 or 0.5)
        - A final **Dense(num_classes)** with softmax
    - Train just this head for **5 epochs** at a learning rate of 1×10^{-3}, allowing only the newly added dense layers to be updated.

2. **Phase 2: Fine-Tune the Last Block of VGG16**
    - **Unfreeze** only the topmost convolutional block in VGG16 (often called Block 5). Earlier blocks remain frozen to preserve lower-level features.
    - With reduced learning rate (depends on the dataset) train for an additional **20 epochs**.
    - At this stage, not only the classification head but also the last convolutional block learns dataset-specific facial features, while earlier convolution blocks stay fixed.

Best Model Preservation. During training, we employed a checkpointing strategy to save the best-performing model based on validation loss. Even though the training process lasted for 20 epochs in the fine-tuning phase, the model with the lowest validation loss was preserved to ensure optimal performance on unseen data. This approach prevents overfitting and ensures that the final model deployed for testing represents the best iteration of the training process.

This approach typically yields faster convergence and better generalization than training only the classification head or unfreezing the entire network all at once. By freezing the initial feature-extraction layers in Phase 1, we preserve the robust feature representations learned from ImageNet, and in Phase 2, we fine-tune the highest-level convolutional filters to adapt to each dataset's unique distribution.

2.4 Data Preprocessing

Regardless of the dataset or framework, we apply a consistent preprocessing pipeline:

1. **Resize:** All images are resized to 224×224 pixels.
2. **Channel Mode:** Convert to 3-channel (RGB). If the source is grayscale, we replicate the single channel three times.
3. **Normalization:** Subtract the ImageNet mean $[0.485, 0.456, 0.406]$ and divide by the ImageNet standard deviation $[0.229, 0.224, 0.225]$.

While the specific code differs slightly between the TensorFlow `tf.data` pipeline and PyTorch `Dataset` transformations, parameters remain aligned.

2.5 General AI Pattern and Code Overview

The general AI pattern employed in this project serves as a unified framework for building, training, and evaluating convolutional neural networks (CNNs) for emotion recognition. This approach ensures consistency across datasets and frameworks (TensorFlow and PyTorch) and facilitates fair comparisons between the two. Below is the general training pipeline used in TensorFlow, which encapsulates the key steps discussed earlier in the chapter:

Listing 1: General Training Pipeline Code

```
import numpy as np
import tensorflow as tf
from keras.applications import VGG16
from keras import layers, models

# Load data
train_x, train_y = np.load("train_images.npy"), np.load("train_labels.npy")
test_x, test_y = np.load("test_images.npy"), np.load("test_labels.npy")

# Preprocessing
def preprocess(img, lbl):
    img = tf.image.resize(img, (224, 224))
    img = tf.repeat(img, 3, axis=-1) if img.shape[-1] == 1 else img
    img = (img - [0.485, 0.456, 0.406]) / [0.229, 0.224, 0.225]
    return img, lbl

train_ds = tf.data.Dataset.from_tensor_slices((train_x, train_y)).map(preprocess)
    .batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((test_x, test_y)).map(preprocess).
    batch(32)

# Model setup
base = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
x = layers.GlobalAvgPool2D()(base.output)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
out = layers.Dense(7, activation="softmax")(x)
model = models.Model(inputs=base.input, outputs=out)

# Training
base.trainable = False
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["
    accuracy"])
model.fit(train_ds, validation_data=test_ds, epochs=5)
base.trainable = True
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5), loss="
    categorical_crossentropy", metrics=["accuracy"])
model.fit(train_ds, validation_data=test_ds, epochs=20)

# Save & Evaluate
model.save("best_model.h5")
print("Accuracy:", model.evaluate(test_ds)[1])
```

2.6 Explanation of the Training Pipeline Code

The code above illustrates the complete pipeline for training a deep learning model for emotion recognition using TensorFlow. Key components include:

- **Data Loading and Preprocessing:** The dataset is loaded and preprocessed by resizing to 224×224, converting to RGB if necessary, and normalizing using ImageNet statistics.
- **Model Design:** A pretrained VGG16 network is augmented with a global pooling layer, dense layers for feature extraction, and a softmax output for classification.
- **Training Strategy:** Two-phase transfer learning is employed:
  - **Phase 1:** Train only the classification head with frozen VGG16 weights.
  - **Phase 2:** Fine-tune the last convolutional block along with the classification head for better adaptation.
- **Model Evaluation:** The best model is saved, and its accuracy on the test set is reported.

By streamlining the pipeline, the essential steps are clearly communicated without redundant details, maintaining uniformity across frameworks.

2.7 Remarks and Conclusions

In summary, we adopted a **VGG16**-based architecture and a **two-phase transfer learning** strategy to maintain a fair comparison across different datasets and frameworks. Each dataset undergoes identical resizing and normalization procedures, and each model trains for two distinct phases to maximize the benefits of transfer learning. The subsequent chapters describe our unified application (Chapter 3) and we evaluated models performance using a variety of metrics (Chapter 4).

3 Application

The goal of this project is to make an application that uses deep learning models to recognize emotions from facial images and videos. The user will be able to choose between multiple models that differ in the framework they were implemented in and on which dataset. The model will be able to recognize up to 8 different emotions. User can either upload an image or video from their computer or turn on their webcamera for real-time emotion recognition.

3.1 Functional requirements

Functional requirements are presented in a form of a User Stories and Use Case Diagram (Figure 9.

As a user, I would like to:

- Upload an image or video, or use a webcam to detect emotions in real-time from facial expressions. (Must Have)
- Display detected emotions clearly after media analysis. (Must Have)
- Provide confidence levels for each detected emotion. (Should Have)
- Track emotions over time, displaying historical emotion data. (Could Have)

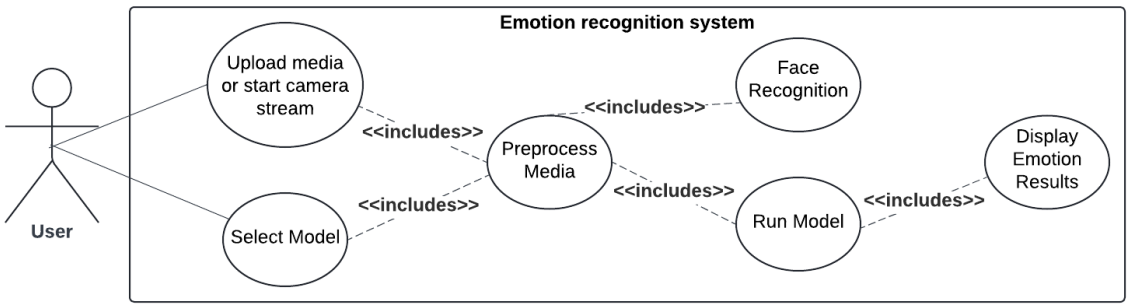


Figure 9: Use Case Diagram for Emotion Detection System

3.2 Non-functional requirements

Non-functional requirements are grouped under categories of URPS.

- **Utility** The system must have a user-friendly interface that fits on a single screen. It should include an easy way to choose between different models. Options to "Upload Video/Image" or to "Start/Stop Webcam" must be present. Finally, the media should be displayed in a dedicated section of the interface.
- **Reliability** The system should function consistently and correctly in a controlled test environment. Multiple tests will be implemented to ensure stable performance.
- **Performance** The system should process emotion detection within reasonable amount of time, ensuring flawless real-time recognition for videos.
- **Supportability** The system must be easily maintainable, allowing for model updates without breaking the existing application.

3.3 Risk Analysis

The following risk analysis has been performed using the SWOT technique to identify the Strengths, Weaknesses, Opportunities, and Threats of our project.

Strengths

- **Strong technical foundation:** The project is based on well-established deep learning methodologies and popular frameworks.
- **Experienced team members:** Both project members have experience in implementing and testing deep learning models.

Weaknesses

- **Limited access to computational resources:** Deep learning projects require significant computational power, which may be limited based on the available hardware.

Likelihood: High
Actions: Explore cloud-based solutions (e.g., AWS, Google Cloud) or optimize the model to run on available hardware.

- **Limited time for project completion:** With strict deadlines, there may be insufficient time to explore all techniques for model to achieve top-notch performance
  Likelihood: High
  Actions: Prioritize critical tasks and consider using backbones like VGG16 or ResNET50.

Opportunities

- **Integration with diverse applications:** Emotion recognition models have applications in various fields, including education, healthcare, entertainment, and human-computer interaction.
- **Potential for future research:** This project can act as a foundation for future research on integrating those models into a more specific application like mental health monitoring tools.

Threats

- **Unavailability of high-quality datasets:** Some of the best datasets for emotion recognition might only be accessible upon request.
  Likelihood: High
  Actions: Seek open-source alternatives.
- **Unexpected software problems:** Unforeseen bugs or compatibility issues may arise during development.
  Likelihood: Moderate
  Actions: Document the whole development process to identify compatibility issues and test the application on multiple platforms.

3.4 System architecture

The application is built on three main components: the Frontend module, the Backend module, and the Emotion Detection Module. Each playing a distinct role in the overall functionality of the application (Figure 10).

Frontend is designed to provide user an intuitive interface. After selecting one of the available emotion recognition models, user can upload a media or strat webcam stream for emotion recognition.

One more functionality of the Frontend module is face detection. Using available third party library, all faces on the media are detected and send to the Backend module in the cropped format.

Backend starts preprocessing received facial data in a way that suits the chose model. Prepared images are sent to the Emotion Detection module where pretrained on a specific dataset model gives the result and sends it to the Backend. Obtained result is transmitted back to the User Interface.

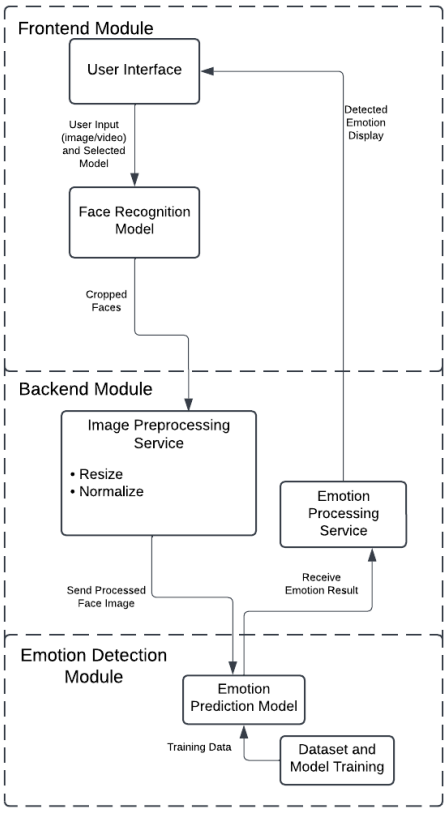


Figure 10: System Architecture Diagram for Emotion Recognition Application

3.5 External interfaces

Our application incorporates external interfaces and standards to enhance its functionality. We utilize a pre-trained face detection model from FaceAPI in our ReactJS frontend to identify faces in uploaded images and video streams. This allows us to focus on developing our own emotion recognition models using TensorFlow and PyTorch. The application accepts standard image and video file formats for uploads and interacts with webcam hardware using established protocols, ensuring compatibility and ease of use for the end-user.

3.6 Technology selection

We chose ReactJS and Electron to develop a responsive and cross-platform desktop application. ReactJS enables efficient UI development with reusable components, while Electron packages the web application into a desktop environment. For backend processing, we selected Python with FastAPI library. FaceAPI is used for face detection, leveraging its robust computer vision capabilities and easy integration into the frontend module. TensorFlow and PyTorch are employed to develop our custom emotion recognition models, taking advantage of their strengths in deep learning and neural network implementation.

3.7 Technical documentation Frontend

The frontend module provides the user interface. In its core, the program allows user to upload different type of images or videos and stream from webcam. By interacting with the backend, it provides real-time emotion recognition results.

To understand the flow of the program, the class diagram is provided below. Since React is not typical object-oriented programming language, as ”classes” we treated separate files. All of them are grouped into categories: Components, Context, Utilities and External Libraries.

Let’s explore each category in details, starting with Components.

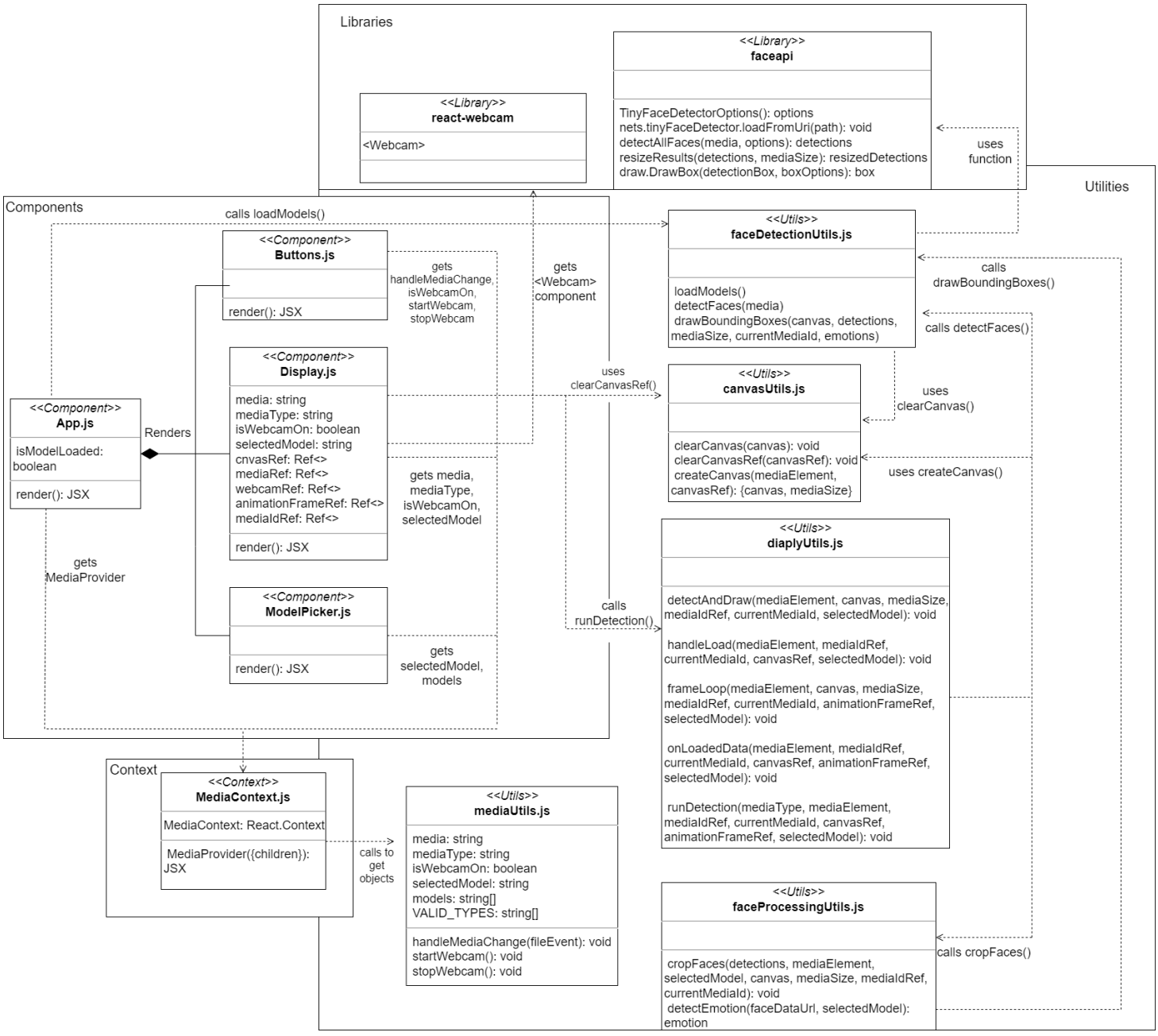


Figure 11: Frontend Class Diagram

Components

This group contains files responsible for rendering the interface and initiating the emotion recognition process. The main wrapper component is App.js, which loads the Face Recognition model available in FaceAPI (we chose TinyFaceDetector) and renders the rest of the components. It also wraps all components in the MediaContext.Provider that gives shared access to variables created in mediaUtils.js. Other components are responsible for rendering buttons, display and model picker.

Context

This category contains only a single file, MediaContext.js. The React context allows us to use the shared variables across different files without passing them from one to another by wrapping them in MediaContext.Provider. All needed variables are created in mediaUtils.js.

Utilities

Utility files contain functions that handle specific tasks in the emotion recognition process. These files ensure modularity by separating concerns.

mediaUtils.js initializes key variables and handles media input, faceDetectionUtils.js integrates with FaceAPI for face detection and emotion bounding boxes, canvasUtils.js manages canvas creation and clearing, displayUtils.js oversees the emotion recognition process via the main function runDetection, and faceProcessingUtils.js handles face cropping and emotion detection through backend API communication.

External libraries

As mentioned previously, FaceAPI library was used for face recognition. Additionally, the webcam-react library was incorporated. It was used for its Webcam component that allows easy manipulations with webcam state and stream.

3.8 Technical documentation Backend

The backend acts as an intermediary between the frontend and the models. It preprocesses the received images, selects the appropriate model using the

received model name and sends the results back to the frontend.

The class diagram (Figure 12) shows main components of the backend code. In the same way as in frontend, we treat each file as its own class. For clarity and readability classes that exist for each model are combined into one since all of them share similar functionality.

The backend is composed of several key modules that handle different responsibilities such as starting the server, managing API endpoints, configuring middleware, and loading models. Below is a detailed breakdown of each component.

Main

The main.py file initializes the FastAPI application and rsets up essential backend components by running most important functions. It starts with preloading models to ensure that all of them are ready at run-time. Next, it configures CORS middleware to allow unproblematic communication with the frontend. Finally, it creates the prediction endpoint by including the API router.

Middleware

In the backend, communication between the frontend and the backend requires enabling Cross-Origin Resource Sharing (CORS). CORS is a security mechanism enforces by web browsers to prevent unauthorized requests between different domains. The cors.py file configures the CORS settings. It ensures that the frontend can successfully make requests to the backend without being blocked by the browser. While CORS is not required for the final desktop application, this setup was crucial for the web-based version that was used during development and can still be accessed using the source code.

Schemas

The backend uses the only schema, EmotionScheme. This schema ensures type validation in the API response by specifying emotion and confidence fields. The schema is defined by the pydantic library which helps to maintain strict data validation and improves clarity of the API's response.

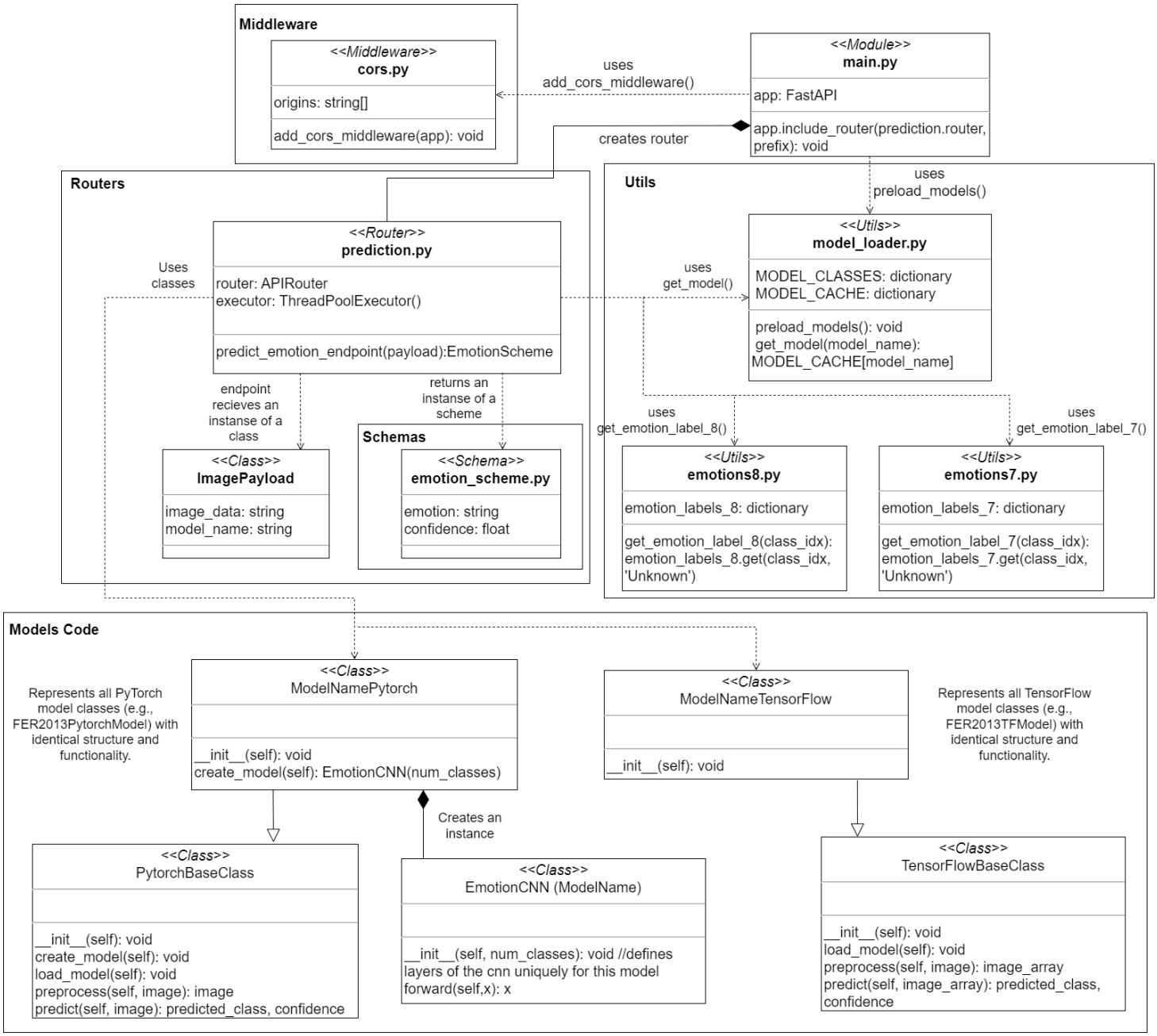


Figure 12: Class Diagram

Routers

The prediction.py router defines the main API endpoint for emotion recognition. The endpoint is presented below:

```
POST /predict_emotion
```

   Request Body:

```
{
  "image_data": "<base64_encoded_image>",
  "model_name": "<model_name>"
}
```

   Response:

```
{
  "emotion": "<predicted_emotion>",
  "confidence": <confidence_score>
}
```

   Possible Status Codes:

- **200 OK**: Emotion prediction was successful.
- **400 Bad Request**: Invalid image data or model name.
- **500 Internal Server Error**: An unexpected error occurred.

The endpoint accepts a POST request in the form of an ImagePayload.The payload consists of an image data and model name. The endpoint returns the predicted emotion along with the confidence rate. When endpoint receives a request, it decodes the base64 image data, retrives requested model and runs prediction asynchronously using ThreadPoolExecutor to avoid blocking other requests. In case of errors, appropriate status codes and messages are returned.

Utils

The utils group includes utility modules for emotion labeling and model management.

emotions_7.py and emotions_8.py define emotion labels for models, mapping predicted indices to emotions with fallback to "Unknown" for out-of-bounds indices. model_loader.py manages model loading and caching, using the preload_model function to load models into a cache and the get_model function to retrieve them, raising an error if unavailable.

Models code

The backend supports emotion recognition using models built with both PyTorch and TensorFlow. Each framework require its own parent class that encapsulates common functionality such as preprocessing input image, loading the model and running prediction.

PyTorch models, inheriting from PytorchBaseClass, require explicit architecture definitions through unique EmotionCNN classes, while TensorFlow models only require loading a specific .h5 file. Preprocessing parameters, such as image size, are defined individually for each model.

3.9 Deployment Documentation

Important Note: The following guide is intended for Windows environments only.

To deploy application and get the working executable, the following steps must be completed:

Package backend into an executable file: navigate to the "Backend" directory and using pyinstaller write the following command in your virtual environment:

```
pyinstaller --onedir --add-data
↪ "app/models/*.pth;app/models" --add-data
↪ "app/models/*.h5;app/models" --hidden-import "torch"
↪ --hidden-import "tensorflow" --name backend app/main.py
```

This will create a new directory named dist, which contains subfolder "backend". It has the executable file "backend.exe" as well as the folder with all required models and libraries for the backend. Bundle frontend with the backend: move generated "backend" directory to the "Frontend/src". Build the react application and bundle build with Electron application by running:

```
npm run build
npx electron-builder
```

Navigate to the ”dist” directory and use the generated setup executable file. It will install the application in the following directory

```
C:\Users\<Username>\AppData\Local\Programs\frontend
```

The installed application is now ready for use.

3.10 Installation Instruction

Provided source code for frontend and backend modules lacks libraries and dependencies that are required to use the development version of the application. For the forntend module the Node.js (the version used during development is 20.12.2). Move to the ”Frontend” directory and run this command:

```
npm install
```

All the required libraries contained in the package.json will be installed to the project. To start the program in both web version and with electron application this command is used:

```
npm run electron-dev
```

To activate the backend, move the provided directory. You are required to have Python installed (used version during development is 3.11.7). Install backend dependencies to your virtual environment and run the server with the second command:

```
pip install -r requirements.txt
uvicorn app.main:app --reload
```

Both frontend and backend have their own tests: unit and integration. To run them, use following commands

```
npm test
pytest app/test/
```

3.11 User's Manual

Loading the app

Start the application using an app.exe file that was installed following the Deployment Guide. At first a console will open, this is because the backend was also turned into executable file and runs first. While backend is loading the models, the user will have a window as below. It may take some time for the backend to load all the models (around a minute).

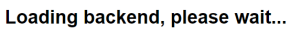


Figure 13: Loading backend window

Initial Interface

When models are finally loaded, the user will have 3 buttons and a dropbox to their disposal. The user can choose one of eight available models which are named according to the set they were trained on and a framework in which they were implemented. The user can select media using buttons available below.

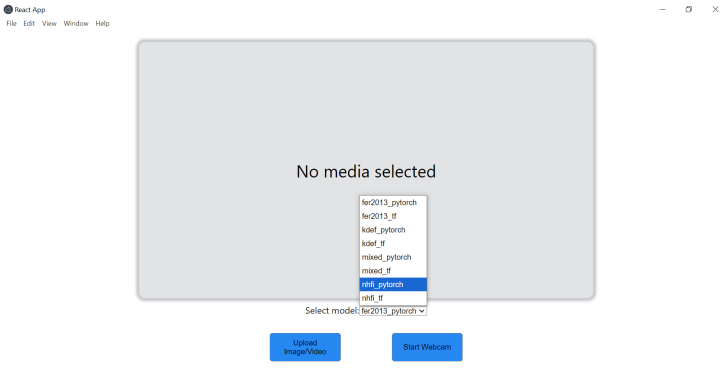


Figure 14: Initial interface window

Image and Video emotion recognition

If users uploads image or video, the media will be statically displayed. Each detected face will have bounding boxes with emotion label. In case of video upload, display will have a player for pausing/playing the video.

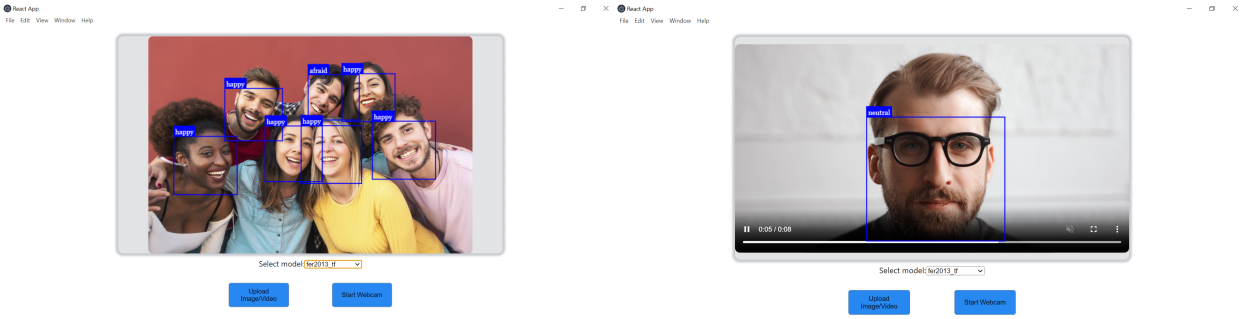


Figure 15: Image and Video recognition windows

Webcam emotion recognition

If "start Webcam" button will be pressed, the webcam stream will immediately start. The previous buttons will disappear and "Stop Webcam" will show up. If the user presses it, the webcam stream will be terminated and the user will return to the initial screen.

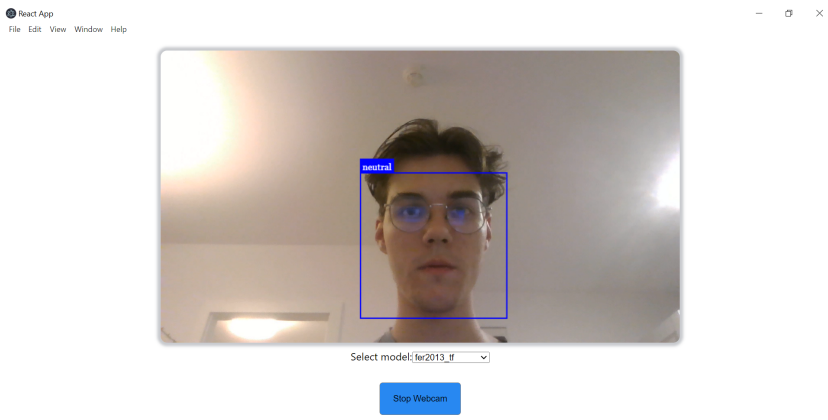


Figure 16: Webcam interface window

This concludes the User Manual as well as the application documentation.

4 Numerical Experiments

This chapter presents the experimental evaluations of our models on each dataset. We compare performance metrics, including accuracy, F1-score, and confusion matrices, and discuss model sizes, training speeds, and domain-based variability.

We also compared TensorFlow vs. PyTorch models under the same conditions and examined how dataset complexity affected final performance.

Experiments were conducted on Kaggle's GPU-accelerated environment, featuring a GPU P100 and 16 GB of memory. Each model was trained for **5 epochs** in **Phase 1** (frozen base), followed by **20 epochs** in **Phase 2** (unfrozen), yielding a total of 25 epochs per model. However, the model with the best validation accuracy treats as final.

We used batch sizes ranging from 8 to 32 (depending on memory constraints) and employed the Adam optimizer with default β parameters.

4.1 KDEF Results

We first tested on the **Karolinska Directed Emotional Faces (KDEF)** dataset [4]. As described in Chapter 1, KDEF is fairly balanced across seven emotions, each having 630 images in training set.

TensorFlow Model. Achieved about **92%** test accuracy after final epoch. Showed an average F1-score of around 0.92 Model size: ∼170 MB in `.h5` format.

PyTorch Model. Achieved **91%** test accuracy on 25th epoch. Final `.pth` file size: ∼56 MB.

Overall, because KDEF features controlled lighting and well-posed expressions, classification is relatively straightforward. The only emotion with which model struggles is afraid that is misclassified as surprised with 17% chance for TensorFlow model and 20% chance for Pytorch model.

Such a high result was expected from this dataset and not aplicable to the real world data, as it was mentioned in Chapter 1[5].

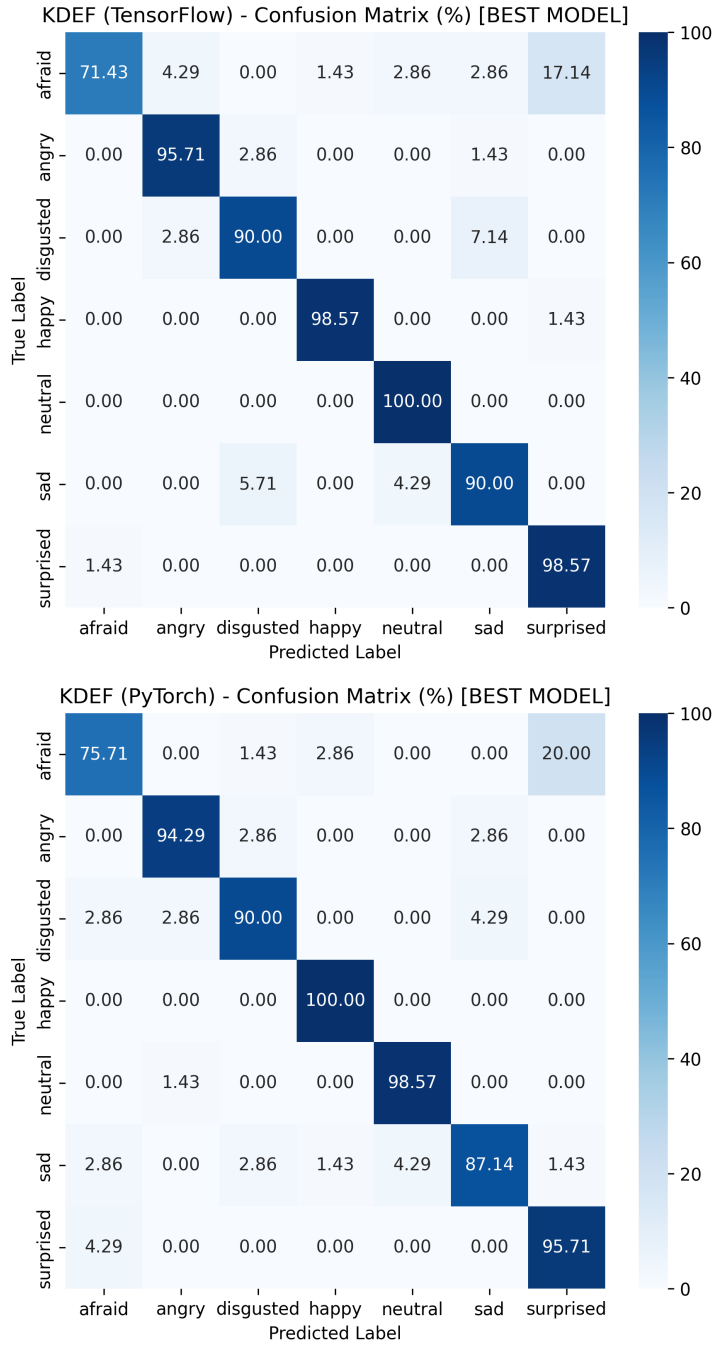


Figure 17: Confusion matrices for KDEF dataset predictions using TensorFlow (first) and PyTorch (second).

4.2 NHFI Results

Next, we evaluated the models on the **Natural Human Face Images (NHFI)** dataset [8].

TensorFlow: Achieved **78%** accuracy on the test set, with an F1-score in the range of 0.72–0.91 for most classes. The best model version was saved after 14th epoch of phase 2. Final .h5 file size: 170 MB.

Misclassification	Percentage	Possible Reason
Angry → Surprised	20.00%	Mouth shape overlap
Happy → Surprised	10.00%	Shared smiling features, wide eyes, or mouth shape overlap

Table 1: Notable misclassifications observed in NHFI TensorFlow model predictions.

PyTorch: Initially, using PyTorch's default "include_top" approach yielded only 46–50% accuracy; removing the original VGG16 classifier and replacing it with our custom head significantly improved performance.

Final accuracy reached **72%** once the custom classification head was precisely matched and was reached after second phase's 18th epoch. Final .pth file size: 56 MB.

Misclassification	Percentage	Possible Reason
Surprised → Angry	16.43%	Mouth shape overlap
Happy → Neutral	11.43%	Overlapping features (smile intensity)
Sad → Neutral	11.43%	Subtle expression overlap
Disgusted → Sad	11.43%	Similar mouth and eye shapes

Table 2: Notable misclassifications observed in NHFI PyTorch model predictions.

The noticeable difference in accuracy between same classes in TensorFlow and PyTorch models could be attributed to differences in preprocessing pipelines, training algorithms, or the ability to reach optimal performance at different epochs. These results highlight the importance of adapting training strategies and preprocessing methods to suit specific frameworks and datasets.

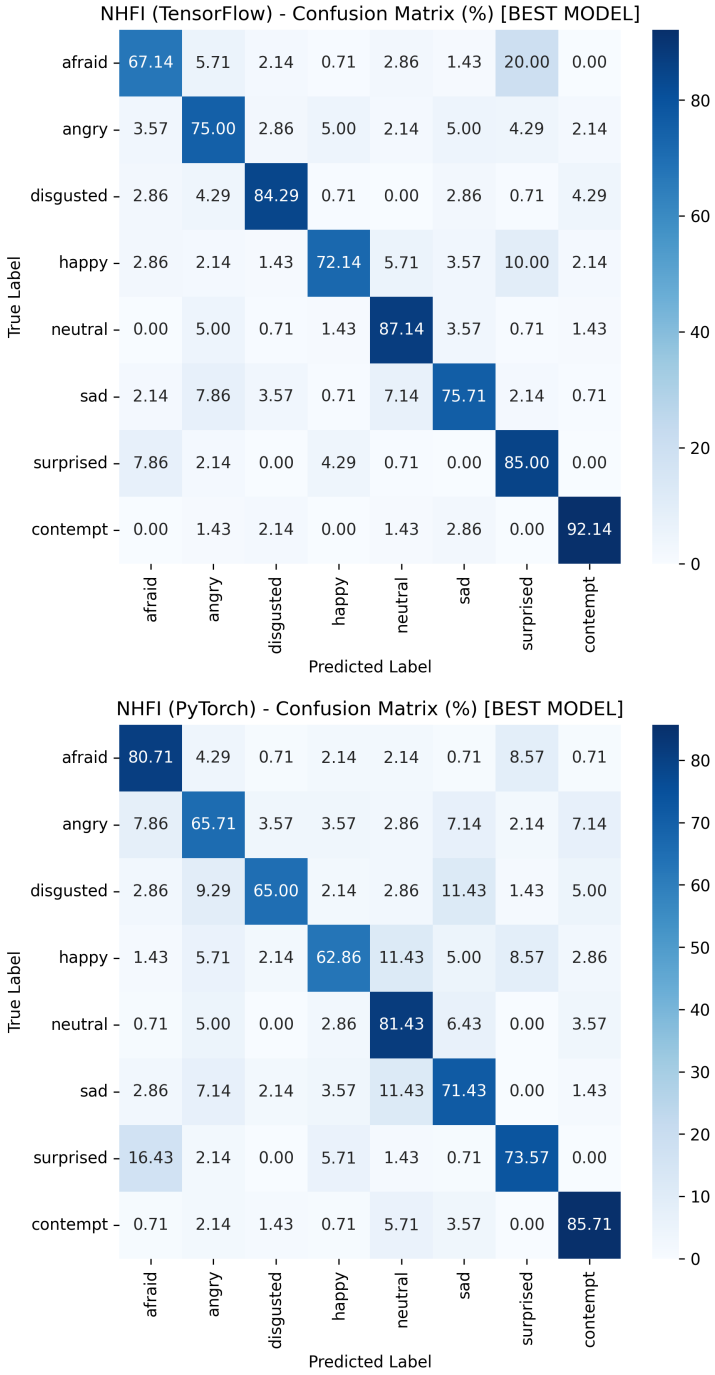


Figure 18: Confusion matrices for NHFI dataset predictions using TensorFlow (first) and PyTorch (second).

4.3 FER2013 Results

FER2013 [6] is the largest dataset utilized in this study, featuring over 50k training images after augmentation, all originally presented in 48×48 grayscale resolution.

TensorFlow Results. The TensorFlow implementation achieved **67%** test accuracy, indicating strong generalization. Most notable were the high precision and recall scores for *disgusted* and *happy*, with F1-scores of **0.89** and **0.81**, respectively. However, the model struggled significantly with *afraid* and *sad*, where the F1-scores dropped to **0.51** and **0.48**.

The best model version for TensorFlow was saved after the 8th epoch of phase 2. Model size is ∼178 MB.

PyTorch Results. The PyTorch implementation closely mirrored TensorFlow in terms of accuracy, achieving **66%** test accuracy. *Disgusted* and *happy* again emerged as the most distinguishable classes, while *afraid* and *sad* exhibited the lowest F1-scores at **0.51** and **0.48**, respectively.

The PyTorch model achieved its best results during the 18th epoch of phase 2, showcasing the framework's consistency in handling large-scale datasets. Final size: ∼56 MB.

Both frameworks showed similar misclassifications, that's why they are presented in one table.

Misclassification	**Percentage in TensorFlow**	**Percentage in Pytorch**	**Possible Reason**
Afraid ↔ Angry	11.25%/14.77%	11.70%/13.98%	Overlap in facial tension features
Afraid ↔ Sad	14.32%/18.75%	17.27%/17.05%	Shared signs of distress
Angry ↔ Sad	14.55%/10.80%	12.95%/13.07%	Similar brow furrow and downturned mouth
Sad → Neutral	15.11%/14.77%	15.68%/16.48%	Expression intensity mismatch

Table 3: Notable misclassifications observed in FER2013 models predictions.

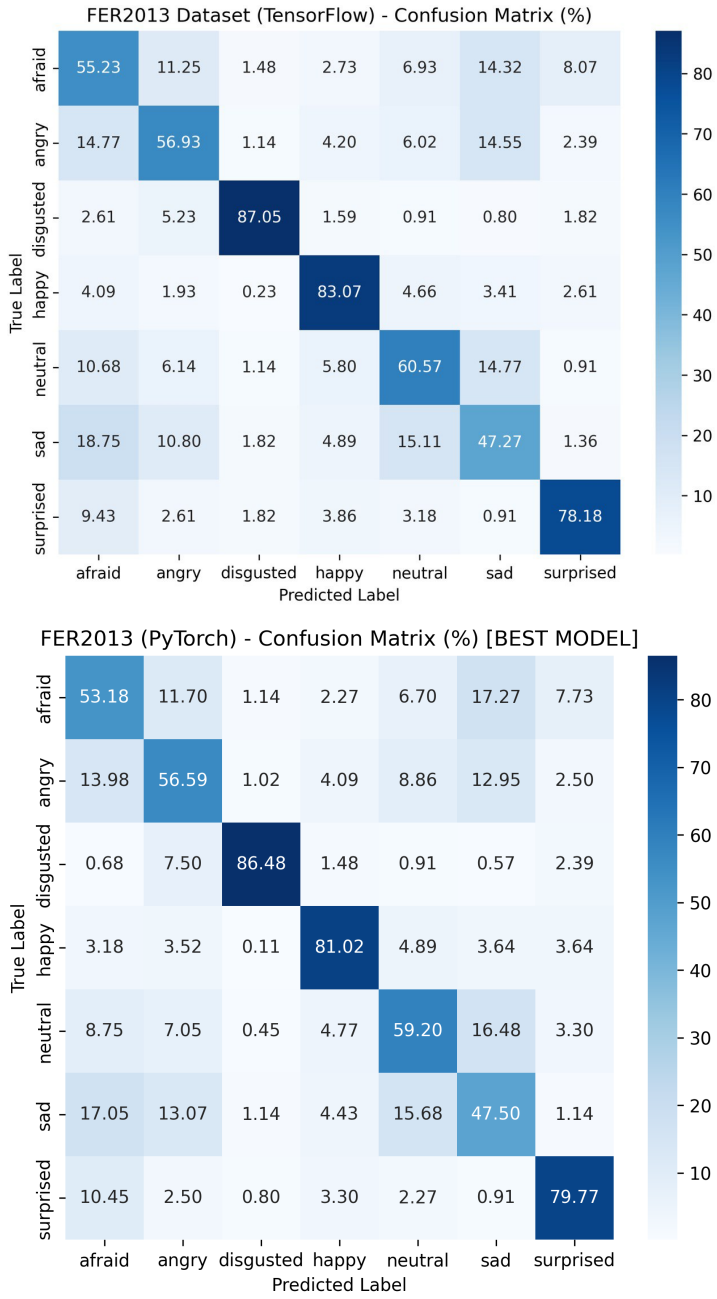


Figure 19: Confusion matrices for FER2013 dataset predictions using TensorFlow (top) and PyTorch (bottom).

The consistent performance across frameworks reaffirms the robustness of the VGG16 architecture, while also highlighting areas where further preprocessing, augmentation, or sampling strategies could be employed to mitigate class imbalance and improve classification accuracy for underrepresented emotions.

4.4 Mixed Dataset Results

Lastly, we tested a **Mixed** dataset of roughly 20k images in training set.

TensorFlow Model. Achieved **70%** test accuracy, with an average F1-score of **0.70**. The best model version was saved after the **8th epoch** of phase 2. Model size: ∼178 MB in `.h5` format.

Misclassification	Percentage	Possible Reason
Afraid → Sad	14.57%	Shared signs of distress
Afraid ↔ Surprised	14.29%/10.86%	Wide-open eyes creating ambiguity
Angry ↔ Sad	14.86%/9.14%	Similar brow furrow and downturned mouth
Neutral ↔ Sad	15.43%/13.71%	Subtle downward mouth curvature

Table 4: Notable misclassifications observed in TensorFlow Mixed Dataset model predictions.

PyTorch Model. Achieved **68%** test accuracy, with an average F1-score of **0.68**. The best model version was saved after the **18th epoch** of phase 2. Final `.pth` file size: ∼56 MB.

Misclassification	Percentage	Possible Reason
Afraid → Angry	10.29%	Overlap in tense facial expressions
Afraid ↔ Surprised	12.86%/18.29%	Wide-open eyes creating ambiguity
Angry ↔ Sad	10.29%/16.00%	Similar brow furrow and downturned mouth
Neutral → Angry	11.43%	Similar tense or focused expressions
Neutral → Sad	15.43%	Subtle downward mouth curvature

Table 5: Notable misclassifications observed in PyTorch Mixed Dataset model predictions.

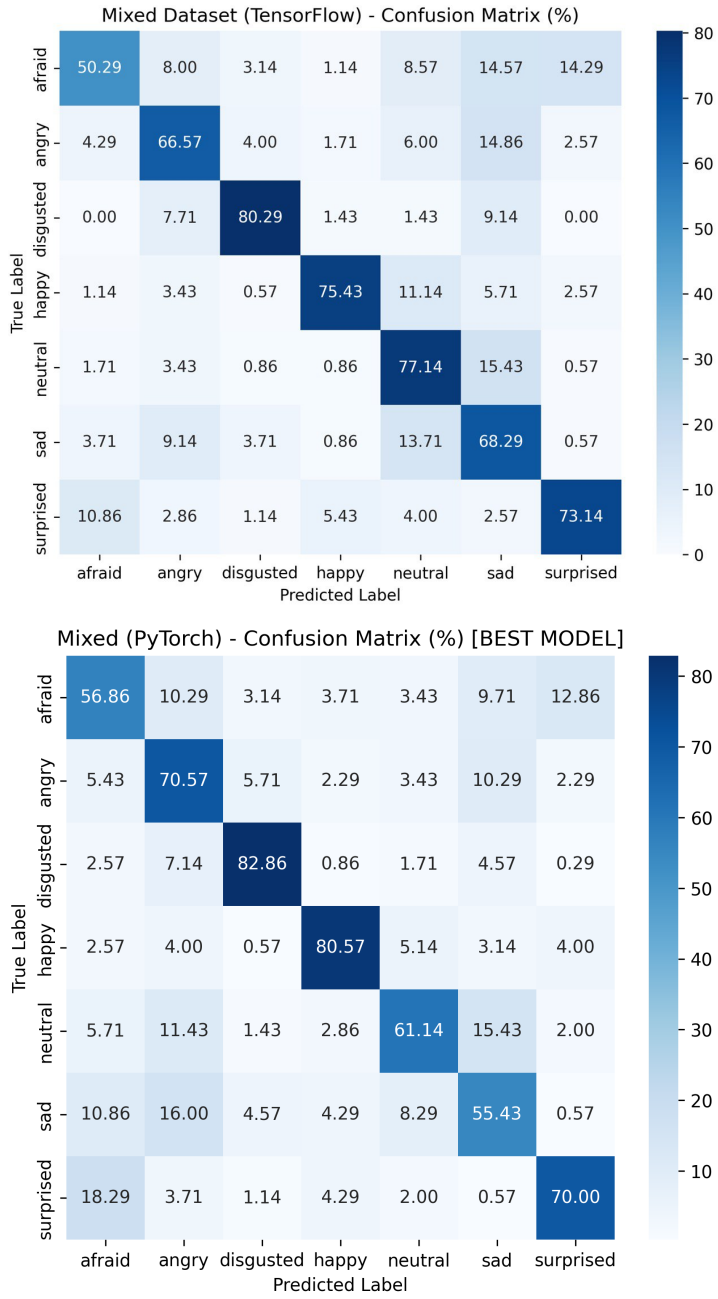


Figure 20: Confusion matrices for Mixed dataset predictions using TensorFlow (first) and PyTorch (second).

The mixed dataset performed better than FER2013 (70% TensorFlow, 68% PyTorch) but worse than KDEF and NHFI. While KDEF's high accuracy comes from its controlled setup, making it less applicable to real-world conditions, the mixed dataset provides a more balanced and realistic benchmark. Common misclassifications indicate the need for better augmentation and feature refinement to improve real-world generalization.

4.5 TensorFlow vs. PyTorch

Despite the fact that both frameworks followed the same structure, we saw some noticeable differences. This section concludes our observations.

- **Framework Behavior:** TensorFlow proved slightly more "plug-and-play," especially with Keras callbacks, whereas PyTorch offered more flexible debugging but demanded careful alignment of the classification head.
- **Size:** PyTorch models weigh **56 MB** each, while TensorFlow models are 3 times bigger with **170 MB**
- **Speed:** Backend tests showed that PyTorch models finish the API call much faster. On average it takes them from 30 ms. TensorFlow once again shows some significant difference with average around 350 ms.
- **Accuracy:** Despite close results, TensorFlow would always show accuracy level a few percentages higher.

To conclude, PyTorch's small size and high speed make it slightly more suitable for real-time application, however it's hard to neglect the fact that TensorFlow constantly shows slightly better results sometimes exceeding 5%.

4.6 Remarks and Conclusions

Across all tested datasets, the **two-phase VGG16 approach** consistently yielded solid results, generally **exceeding 60%** accuracy in every scenario and surpassing **70%** in several. Both frameworks can effectively train emotion recognition models when hyperparameters and architectures are matched. These experiments confirm that transfer learning—especially with a well-understood architecture like VGG16 remains a powerful method for developing robust facial expression classifiers.

Summary

This work was devoted to the development of an emotion recognition application using deep learning models trained on facial expression images. We evaluated four datasets—**KDEF**, **NHFI**, **FER2013**, and a **Mixed** set—and built two VGG16-based models per dataset (TensorFlow and PyTorch).

Here's what we found out:

1. **Model Performance**: All models achieved 60%+ accuracy, with some exceeding 70%. Framework differences were minimal (<5%), emphasizing architecture and training over framework choice.

2. **Data Diversity**: KDEF (controlled) yielded consistent results, while NHFI's added "contempt" class introduced subtle overlaps. FER2013, the largest dataset, achieved lower accuracy due to its 48x48 resolution (suboptimal for VGG16's 224x224 input), but its real-world nature makes it more practical for deployment. The Mixed set improved generalization but caused confusion among similar expressions.

3. **Transfer Learning**: A two-phase approach (freezing then unfreezing VGG16 layers) accelerated convergence and reduced overfitting, especially for smaller datasets like KDEF.

4. **Application**: We developed a desktop app (ReactJS + Electron front-end, FastAPI + Python back-end) for real-time or batch emotion detection, enabling users to compare results across eight models.

Conclusion: VGG16-based CNNs reliably classify emotions across diverse datasets. TensorFlow and PyTorch implementations showed comparable performance, suggesting framework choice depends on developer preference or deployment needs. For real-world applications, we recommend using large datasets with resolutions close to 224x224, as they generalize better. Our application demonstrates practical emotion detection, paving the way for further research in automated facial expression analysis.

Bibliography

[1] R. Guo, H. Guo, L. Wang, M. Chen, D. Yang, and B. Li, "Development and application of emotion recognition technology — a systematic literature review," *BMC Psychology*, 2024. Available at: https://bmcpsychology.biomedcentral.com/articles/10.1186/s40359-024-01581-4. Accessed: 26 January 2025.

[2] P. Huang, "Decoding Emotions: Intelligent visual perception for movie image classification using sustainable AI in entertainment computing," *Entertainment Computing*, 2024. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1875952124000648. Accessed: 26 January 2025.

[3] S. Tippannavar, S. D. Yashwanth, K. M. Puneeth, M. P. Madhu Sudan, B. N. Chandrashekar Murthy, and Eshwari A Madappa, "Advances and Challenges in Human Emotion Recognition Systems: A Comprehensive Review," *Journal of Trends in Computer Science and Smart Technology*, 2023. Available at: https://www.researchgate.net/publication/375601116_Advances_and_Challenges_in_Human_Emotion_Recognition_Systems_A_Comprehensive_Review. Accessed: 26 January 2025.

[4] D. Lundqvist, A. Flykt, and A. Öhman, "Karolinska Directed Emotional Faces (KDEF)," *Official Dataset Website*, 1998. Available at: https://kdef.se/. Accessed: 17 January 2025.

[5] M. Jaiswal, Z. Aldeneh, and E. Mower Provost, "Controlling for Confounders in Multimodal Emotion Classification via Adversarial Learning," *2019 International Conference on Multimodal Interaction (ICMI)*, ACM, 2019. Available at: https://dl.acm.org/doi/fullHtml/10.1145/3340555.3353731. Accessed: 17 January 2025.

[6] P.-L. Carrier and A. Courville, "Challenges in Representation Learning: Facial Expression Recognition Challenge," *Official Kaggle Dataset webpage*, 2013. Available at: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data. Accessed: 18 January 2025.

[7] Y. Khaireddin and Z. Chen, "Facial Emotion Recognition: State of the Art Performance on FER2013," *arXiv preprint arXiv:2105.03588*, 2021.

Available at: https://arxiv.org/abs/2105.03588. Accessed: 18 January 2025.

[8] S. Vaidya, "Natural Human Face Images for Emotion Recognition," *Official Kaggle Dataset webpage*, 2020. Available at: https://www.kaggle.com/datasets/sudarshanvaidya/random-images-for-face-emotion-recognition. Accessed: 19 January 2025.

[9] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, "Image Data Augmentation for Deep Learning: A Survey," *arXiv preprint arXiv:2204.08610*, 2022. Available at: https://arxiv.org/abs/2204.08610. Accessed: 20 January 2025.

[10] A. De Arriba, M. Oriol, and X. Franch, "Merging Datasets for Emotion Analysis," *IEEE Xplore*, 2021. Available at: https://ieeexplore.ieee.org/document/9680305. Accessed: 20 January 2025.

[11] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, 2010, pp. 94–101.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. Available at: https://pytorch.org/. Accessed: 26 January 2025.

[13] M. Abadi, et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from https://tensorflow.org/. Accessed: 26 January 2025.

[14] T. Wolf, et al., "Transformers: State-of-the-Art Natural Language Processing," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, 2020, pp. 38–45. Available at: https://github.com/huggingface/transformers. Accessed: 26 January 2025.

[15] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *International Conference on Machine Learning (ICML)*, 2019, pp. 6105–6114.

[16] A. Dosovitskiy, et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *International Conference on Learning Representations (ICLR)*, 2021. Available at: `https://arxiv.org/abs/2010.11929`. Accessed: 26 January 2025.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2012.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., "Going Deeper with Convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations (ICLR)*, 2015. Available at: `https://arxiv.org/abs/1412.6980`. Accessed: 27 January 2025.

[21] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations (ICLR)*, 2015. Available at: `https://arxiv.org/abs/1409.1556`. Accessed: 27 January 2025.

List of Figures

List of Tables