Eduvest – Journal of Universal Studies
Volume 5 Number 7, July, 2025
p- ISSN 2775-3735- e-ISSN 2775-3727

Enhancing Work Safety Systems Through Real-Time Speech Emotion Detection Classifier Using CNN Algorithm

Narendra Rahman Handwi, Rila Mandala
President University, Idonesia
Email: narendrarahman2@gmail.com, rilamandala@president.ac.id

ABSTRACT

Speech emotion detection has emerged as a significant research area due to its potential applications in various domains. In work safety systems, the ability to accurately recognize emotions can provide vital information about the mental state of workers, which can be utilized to prevent work accidents and ensure a safer work environment. The objective of this study is to develop a speech emotion detection classifier using the CNN algorithm. The classifier aims to accurately classify emotions from speech signals, enabling real-time recognition of workers' emotional states. By achieving this objective, the study aims to contribute to the enhancement of work safety systems. The proposed methodology involves training a Convolutional Neural Network (CNN) model using a comprehensive dataset of labeled speech samples. The dataset will encompass various emotions, including happiness, sadness, anger, and fear. The CNN model will be trained to extract relevant features from speech signals and learn the patterns associated with different emotional states. Preprocessing techniques, such as audio segmentation, feature extraction, and data augmentation, will be employed to enhance the training process. The expected result of this study is a robust speech emotion detection classifier that can accurately classify emotions from speech signals. The classifier will be capable of real-time emotion recognition, providing immediate insights into workers' emotional states. By integrating this classifier into work safety systems, proactive measures can be taken to prevent work accidents based on workers' emotional conditions.

KEYWORDS *Speech emotion detection, Convolutional Neural Network (CNN), work safety systems, emotion recognition, real-time classification.*

INTRODUCTION

Speech emotion detection has emerged as a critical area of research with applications across various domains. Understanding and accurately recognizing emotions from speech signals has the potential to significantly impact work safety systems. Workplaces, particularly those involving high-risk environments, can benefit immensely from real-time emotion detection as it provides valuable insights into the mental states of workers. This information can be leveraged to prevent work-related accidents, enhance employee well-being, and create a safer work environment.

Traditionally, work safety systems have primarily focused on physical safety measures, machinery safeguards, and safety protocols. While these measures are essential, they often overlook the emotional well-being of workers, which plays a crucial role in accident prevention. The integration of real-time speech emotion detection can bridge this gap by providing continuous monitoring and assessment of workers' emotional states.

The existing work safety systems face several challenges that hinder their effectiveness in ensuring a safe work environment. One major issue is the limited emotional monitoring, as conventional safety systems lack the capability to monitor and assess the emotional states of workers in real-time. Accidents in the workplace often occur due to emotional stress, fatigue, or other factors affecting workers' mental states. Unfortunately, current safety systems are predominantly reactive and do not address emotional well-being proactively. Additionally,

human supervisors may not always accurately gauge the emotional conditions of workers, leading to potential misinterpretations or biases in assessing safety risks.

Previous research has made significant contributions to the development of emotion detection systems using speech signals, but there are notable gaps in applying these systems to work safety environments. For example, traditional approaches have largely focused on machine learning methods like *Hidden Markov Models* (HMMs) and *Support Vector Machines* (SVMs), which have been used for classifying emotional states (Jiang et al., 2019). However, these methods struggle with accuracy and real-time processing, which are crucial in high-risk work environments. Additionally, research on integrating emotional monitoring into work safety systems has been limited, particularly in addressing the emotional well-being of workers in real-time, which this study aims to bridge. Unlike previous studies, this research focuses on developing a speech emotion detection classifier using *Convolutional Neural Networks* (CNNs), a more advanced technique that has shown promise in audio-related tasks and could significantly enhance the accuracy and efficiency of emotion detection systems (Zhou et al., 2021).

The novelty of this research lies in several key aspects. First, there is a need to develop a robust speech emotion detection classifier specifically tailored for work safety systems, enabling effective responses to workers' emotional states in real-time. Additionally, existing research lacks comprehensive studies that address a broad range of emotions, such as happiness, sadness, anger, and fear, which are essential for creating well-rounded emotion detection systems. Moreover, much of the prior research has focused on traditional machine learning methods like *Hidden Markov Models* (HMMs) and *Support Vector Machines* (SVMs), which may not be as effective as newer techniques. Finally, while *Convolutional Neural Networks* (CNNs) have shown promise in various audio-related tasks, their potential in developing advanced speech emotion detection classifiers for work safety systems remains underexplored and warrants further investigation.

The primary objective of this research is to develop an effective speech emotion detection classifier using *Convolutional Neural Networks* (CNN). The classifier aims to accurately classify emotions from speech signals and enable real-time recognition of workers' emotional states. By achieving these objectives, this study seeks to contribute to the enhancement of work safety systems, promoting a safer work environment through proactive measures.

This research will focus on the development of a speech emotion detection classifier using CNN and the utilization of a comprehensive dataset containing labeled speech samples covering various emotions such as happiness, sadness, anger, and fear. The study will also implement preprocessing techniques, including audio segmentation, feature extraction, and data augmentation, to improve the training process and enhance accuracy. Furthermore, the developed classifier will be integrated into existing work safety systems for real-time emotional recognition and proactive safety measures.

Based on the research objectives, the following hypotheses are formulated: Hypothesis 1 (H1) posits that a speech emotion detection classifier based on *Convolutional Neural Networks* (CNN) can accurately classify emotions from speech signals, while Hypothesis 2 (H2) suggests that real-time emotion detection using the CNN classifier will contribute to proactive safety measures, thereby preventing work accidents based on workers' emotional conditions.

METHOD

This research aims to enhance workplace safety systems through real-time speech emotion detection using the *Convolutional Neural Network* (CNN) algorithm. A diverse and representative dataset is collected from various sources containing different emotion types such as happiness, sadness, anger, and neutral expressions. The research methodology consists of eight systematic phases: importing libraries for data manipulation, inputting data from prepared datasets, visualizing data using bar graphs to understand emotion distribution, augmenting data to improve quality and diversity through techniques like noise injection and time shifting, building CNN models according to predetermined structures, training models using augmented data through iterative processes, evaluating model accuracy on test data by displaying loss and accuracy graphs, and predicting labels on test data for comprehensive model performance evaluation.

The data for this study is collected from a variety of publicly available speech emotion datasets, ensuring diversity and representativeness across different emotion categories such as happiness, sadness, anger, and neutral expressions. Sources like the Emo-DB dataset and RAVDESS (*The Ryerson Audio-Visual Database of Emotional Speech and Song*) are used, as they offer labeled audio samples containing multiple emotional expressions. These datasets provide a sufficient range of speech samples recorded under controlled environments to ensure quality and consistency. The data collection process includes audio recordings of speech from different speakers, genders, and age groups to reflect natural variations in emotional expression.

The analysis begins by applying data augmentation techniques to enhance the diversity and quality of the dataset. Data augmentation methods such as noise injection and time shifting are used to simulate various real-world conditions, increasing the robustness of the model. This also helps prevent overfitting and ensures that the model is capable of generalizing well to unseen data. The *Convolutional Neural Network* (CNN) model is built and trained using these augmented datasets. The model architecture is carefully designed, considering factors like the number of layers, filter sizes, and activation functions. The training process is iterative, with the model continuously learning from the data, adjusting its weights based on the feedback received from the loss function during backpropagation. To evaluate the performance of the trained model, accuracy and loss metrics are computed using a separate test dataset that was not part of the training process. The results are visualized in the form of accuracy and loss graphs to assess the model's ability to classify emotions accurately and its capacity to generalize to new data. Finally, the model's predictions on the test data are analyzed to determine the effectiveness of real-time emotion detection and its potential application in workplace safety systems.

RESULT AND DISCUSSION

Implementation

All design stages are implemented into program code using the Python programming language version 3.10. Some packages from the library used are as follows:

Import Libraries

This code snippet (Figure 1) imports a comprehensive set of libraries essential for

processing audio data, visualizing results, building and training a neural network model, and evaluating its performance. It includes modules for operating system interactions (`os`), data manipulation (`pandas`, `numpy`), and random number generation (`random`). For audio processing, it uses `librosa` and `IPython.display` to load audio files, extract features, and display rich media. Data visualization is handled by `seaborn` and `matplotlib.pyplot`, along with `matplotlib.colors.Normalize` for scaling data values. The neural network model is constructed and trained using `tensorflow.keras` layers and callbacks, with `sklearn.model_selection.train_test_split` for splitting datasets. Model evaluation employs metrics from `sklearn.metrics`. Plot configurations are set with `%matplotlib inline` and `%config InlineBackend.figure_format='retina'` for high-resolution inline plots, and warnings are suppressed using `warnings.filterwarnings("ignore")`. This integrated approach facilitates comprehensive machine learning workflows in audio data analysis.

```
# Importing libraries
import os
import pandas as pd
import random
import numpy as np
import librosa
import librosa.display
import IPython.display as ipd
from IPython.core.display import display
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, BatchNormalization, Dropout, Flatten, Dense, MaxPool2D
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
%matplotlib inline
%config InlineBackend.figure_format='retina'
import warnings
warnings.filterwarnings("ignore")
```

Figure 1. Import Libraries

Data Input

The code (Figure 2) is used to connect Google Colab with Google Drive. The first line imports the `drive` module from the `google.colab` library, which provides functions to access Google Drive. The second line calls the `mount` function from the module to mount Google Drive into the Colab file system at the `/content/drive` directory. This allows users to access files and folders in Google Drive directly from Colab, making it easier to save and access data while working in Colab.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 1. Data Input Drive

This code (Figure 3) extracts data from the TESS (Toronto Emotional Speech Set) dataset by traversing the specified directory, storing the full file paths in the `paths` list,

calculating and storing the durations of audio files in the `duration` list using `librosa.get_duration`, and extracting emotion labels from the filenames. These labels are processed and stored in the `labels` list. A DataFrame `df_tess` is then created with columns for paths, durations, the dataset identifier 'TESS', and emotion labels, with the 'ps' label replaced by 'surprise' for consistency. Finally, a sample of five rows from the DataFrame is displayed to verify the data integration.

```
[ ] # Getting TESS data:
    paths, labels, duration = [], [], []

    for dirname, _, filenames in os.walk('/content/drive/MyDrive/Voice Recognition/toronto-emotional-speech-set-tess'):
        for filename in filenames:
            paths.append(os.path.join(dirname, filename))
            duration.append(round(librosa.get_duration(filename=paths[-1]), 3))
            label = filename[::-1].split('_')[0][::-1]
            labels.append(label[:-4].lower())

    df_tess = pd.DataFrame({'path':paths,'duration': duration, 'dataset': 'TESS', 'emotion':labels})
    df_tess['emotion'] = df_tess['emotion'].replace(['ps'], 'surprise')
    df_tess.sample(5)
```

	path	duration	dataset	emotion
5169	/content/drive/MyDrive/Voice Recognition/toron...	2.057	TESS	happy
3721	/content/drive/MyDrive/Voice Recognition/toron...	2.008	TESS	happy
1639	/content/drive/MyDrive/Voice Recognition/toron...	2.312	TESS	sad
2234	/content/drive/MyDrive/Voice Recognition/toron...	1.894	TESS	surprise
3583	/content/drive/MyDrive/Voice Recognition/toron...	1.675	TESS	angry

Figure 2. Data Input Toronto

This code snippet (Figure 4) retrieves data from the RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song) dataset by recursively traversing the specified directory using `os.walk`. For each file found (`filenames`), it captures the full path (`paths`), calculates the duration of the audio file using `librosa.get_duration`, and extracts the emotion label based on the filename pattern. Emotions are mapped to predefined categories such as 'neutral', 'calm', 'happy', 'sad', 'angry', 'fear', 'disgust', and 'surprise' according to specific codes embedded in the filenames. These details are organized into a DataFrame `df_ravdess` with columns for file paths, durations, the dataset identifier 'RAVDESS', and corresponding emotion labels. A sample of five randomly selected rows from the DataFrame is displayed to ensure the accuracy of data extraction and processing.

```
[ ] # Getting RAVDESS data
    paths, labels, duration = [], [], []

    for dirname, _, filenames in os.walk('/content/drive/MyDrive/Voice Recognition/ravdess-emotional-speech-audio'):
        for filename in filenames:
            paths.append(os.path.join(dirname, filename))
            duration.append(round(librosa.get_duration(filename=paths[-1]), 3))
            label = filename[::-1].split('_')[0][::-1]
            if label[6:8] == '01':
                labels.append('neutral')
            elif label[6:8] == '02':
                labels.append('calm')
            elif label[6:8] == '03':
                labels.append('happy')
            elif label[6:8] == '04':
                labels.append('sad')
            elif label[6:8] == '05':
                labels.append('angry')
            elif label[6:8] == '06':
                labels.append('fear')
            elif label[6:8] == '07':
                labels.append('disgust')
            elif label[6:8] == '08':
                labels.append('surprise')

    df_ravdess = pd.DataFrame({'path':paths,'duration': duration, 'dataset': 'RAVDESS', 'emotion':labels})
    df_ravdess.sample(5)
```

	path	duration	dataset	emotion
2558	/content/drive/MyDrive/Voice Recognition/ravde...	3.370	RAVDESS	fear
807	/content/drive/MyDrive/Voice Recognition/ravde...	3.737	RAVDESS	calm
2084	/content/drive/MyDrive/Voice Recognition/ravde...	3.370	RAVDESS	happy
1954	/content/drive/MyDrive/Voice Recognition/ravde...	4.071	RAVDESS	disgust
2188	/content/drive/MyDrive/Voice Recognition/ravde...	3.737	RAVDESS	calm

Figure 3. Data Input RAVDESS

This code (Figure 5) segment initializes three empty lists: `paths`, `labels`, and `duration`. It then walks through the directory '/content/drive/MyDrive/Voice Recognition/surrey-audiovisual-expressed-emotion-savee/' using `os.walk`, iterating over each filename in `filenames`. It ensures that only files ending with '.wav' are processed. For each valid file, it captures the full file path (`paths`) and extracts the emotion label from the filename (`labels`). Emotion labels are determined based on the first character of the reversed filename (`filename[::-1].split('_')[0][::-1]`), mapping specific codes ('a' for angry, 'd' for disgust, 'f' for fear, 'h' for happy, 'n' for neutral, 's' for sad or surprise) to their corresponding emotions. Assertions are used to verify that `paths` and `labels` lists are of the same length before calculating and appending audio durations (`duration`) using `librosa.get_duration`. Another assertion ensures that `paths` and `duration` lists are also of the same length. Finally, a DataFrame `df_savee` is created with columns for file paths (`path`), durations (`duration`), dataset identifier 'SAVEE', and emotion labels (`emotion`). A sample of five randomly selected rows from `df_savee` is displayed to confirm the correct extraction and organization of data.

```
[ ] # Initializing lists
    paths, labels, duration = [], [], []

    # Walking through the directory to get file paths and labels
    for dirname, _, filenames in os.walk('/content/drive/MyDrive/Voice Recognition/surrey-audiovisual-expressed-emotion-savee/'):
        for filename in filenames:
            if filename.endswith('.wav'):  # Ensure only .wav files are processed
                paths.append(os.path.join(dirname, filename))
                label = filename[::-1].split('_')[0][::-1]
                if label[:1] == 'a':
                    labels.append('angry')
                elif label[:1] == 'd':
                    labels.append('disgust')
                elif label[:1] == 'f':
                    labels.append('fear')
                elif label[:1] == 'h':
                    labels.append('happy')
                elif label[:1] == 'n':
                    labels.append('neutral')
                elif label[:1] == 's':
                    if label[:2] == 'sa':
                        labels.append('sad')
                    else:
                        labels.append('surprise')

    # Ensure lists are of the same length before adding durations
    assert len(paths) == len(labels), "Paths and labels lists are not the same length."

    # Calculate durations
    for file in paths:
        duration.append(round(librosa.get_duration(filename=file), 3))

    # Ensure duration list is of the same length
    assert len(paths) == len(duration), "Paths and duration lists are not the same length."

    # Creating DataFrame
    df_savee = pd.DataFrame({'path': paths, 'duration': duration, 'dataset': 'SAVEE', 'emotion': labels})
    df_savee.sample(5)
```

	path	duration	dataset	emotion
413	/content/drive/MyDrive/Voice Recognition/surre...	3.290	SAVEE	disgust
435	/content/drive/MyDrive/Voice Recognition/surre...	5.856	SAVEE	surprise
18	/content/drive/MyDrive/Voice Recognition/surre...	3.312	SAVEE	fear
171	/content/drive/MyDrive/Voice Recognition/surre...	3.833	SAVEE	neutral
16	/content/drive/MyDrive/Voice Recognition/surre...	4.361	SAVEE	disgust

Figure 4. Data Input SAVEE

The code snippet (Figure 6) integrates the CREMA-D dataset into the workflow by defining the directory containing the audio files, listing all files within it, and initializing lists to store file paths and corresponding emotions. It iterates over each file, appends the full file path to `file_path`, and maps parts of the filename to predefined emotion labels (`fatigue`, `angry`, `disgust`, `drowsiness`, `happy`, `neutral`, and `unknown`). A DataFrame `Crema_df` is created with columns for file paths, emotions, and durations (calculated using `librosa.get_duration`), and a dataset identifier column labeled 'CREMA-D'. A sample of the DataFrame is displayed to verify the data integration.

```
# Adding CREMA-D dataset
Crema = '/content/drive/MyDrive/Voice Recognition/cremad/AudioWAV/'
crema_directory_list = os.listdir(Crema)

file_emotion = []
file_path = []

for file in crema_directory_list:
    file_path.append(Crema + file)
    part = file.split('_')
    if part[2] == 'SAD':
        file_emotion.append('fatigue')
    elif part[2] == 'ANG':
        file_emotion.append('angry')
    elif part[2] == 'DIS':
        file_emotion.append('disgust')
    elif part[2] == 'FEA':
        file_emotion.append('drowsiness')
    elif part[2] == 'HAP':
        file_emotion.append('happy')
    elif part[2] == 'NEU':
        file_emotion.append('neutral')
    else:
        file_emotion.append('unknown')

Crema_df = pd.DataFrame({'path': file_path, 'emotion': file_emotion})
Crema_df['duration'] = Crema_df['path'].apply(lambda x: round(librosa.get_duration(filename=x), 3))
Crema_df['dataset'] = 'CREMA-D'
Crema_df.sample(5)
```

	path	emotion	duration	dataset
5126	/content/drive/MyDrive/Voice Recognition/crema...	neutral	2.035	CREMA-D
1748	/content/drive/MyDrive/Voice Recognition/crema...	angry	2.669	CREMA-D
2388	/content/drive/MyDrive/Voice Recognition/crema...	angry	2.469	CREMA-D
97	/content/drive/MyDrive/Voice Recognition/crema...	happy	1.969	CREMA-D
5132	/content/drive/MyDrive/Voice Recognition/crema...	fatigue	2.970	CREMA-D

Figure 5. Data Input CREMA-D

Data Visualization

This code (Figure 7) segment utilizes Matplotlib and Seaborn to visually represent the characteristics of audio data stored in the DataFrame `df`. The first subplot (`axes[0]`) plots a stacked bar chart showing the distribution of audio files categorized by emotion and dataset. It groups the data by 'emotion' and 'dataset', computes the count of samples for each group, and stacks the bars to illustrate how emotions are distributed across different datasets. The second subplot (`axes[1]`) uses Seaborn's violin plot to display the distribution of audio sample durations (`duration`) across various emotions within each dataset. This plot provides insights into the range and distribution of audio durations for different emotional categories. Overall, these visualizations help in understanding both the categorical distribution and temporal characteristics of the audio dataset, aiding in further analysis and interpretation of the data.

Figure 6. Data Visualization one

The `show_audio` function in the provided code (Figure 8) iterates through each unique emotion category in the DataFrame `df` and visualizes corresponding audio features including waveform, spectrogram with fundamental frequency, and Mel-frequency cepstral coefficients (MFCCs). It first filters `df` to select rows based on the current emotion, loads the audio file using Librosa, and plots these features on separate subplots within a single figure. This function facilitates the comprehensive exploration and understanding of audio characteristics associated with different emotions present in the dataset, aiding in both qualitative analysis and model development for emotion recognition tasks.

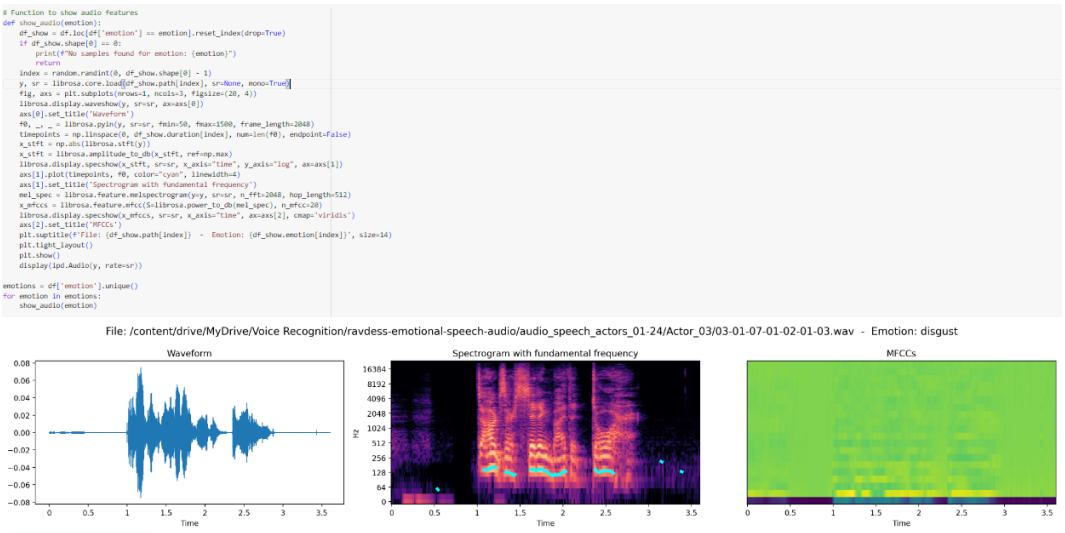


Figure 7. Data Visualization Two

Modelling

This code snippet (Figure 9) utilizes the Keras library from TensorFlow to construct a Convolutional Neural Network (CNN) architecture for the purpose of classifying audio data, particularly for emotion recognition. The model comprises convolutional layers with ReLU activation, max-pooling layers for downsampling, batch normalization layers for stabilizing learning, and dropout layers for regularization to prevent overfitting. It also includes densely connected layers with ReLU activation and a softmax output layer for multi-class classification.

The model summary provides insights into the layer types, output shapes, and parameter counts, facilitating further compilation and training for specific classification tasks.

```
from tensorflow import keras
from tensorflow.keras.layers import (Conv2D, BatchNormalization, Dropout, Flatten, Dense, MaxPool2D)
from tensorflow.keras import initializers

# Create convolutional neural network and return summary
model = keras.Sequential()
model.add(Conv2D(filters=64, kernel_size=5, strides=(2, 2), activation="relu", input_shape=x_tr.shape[1:]))
model.add(MaxPool2D(pool_size=2))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=4, strides=(2, 1), activation="relu"))
model.add(MaxPool2D(pool_size=2))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=7, activation="softmax"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 13, 73, 64)        1664

 max_pooling2d (MaxPooling2  (None, 6, 36, 64)         0
 D)

 batch_normalization (Batch  (None, 6, 36, 64)         256
 Normalization)

 conv2d_1 (Conv2D)           (None, 2, 33, 32)         32800

 max_pooling2d_1 (MaxPoolin  (None, 1, 16, 32)         0
 g2D)

 batch_normalization_1 (Bat  (None, 1, 16, 32)         128
 chNormalization)

 flatten (Flatten)           (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 7)                 455

=================================================================
Total params: 109223 (426.65 KB)
Trainable params: 109031 (425.90 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
```

Figure 8. Modelling

Train a Neural Network Model

The provided code (Figure 10) segment compiles a neural network model using `adam` optimizer, `sparse_categorical_crossentropy` loss function, and accuracy as the evaluation metric. It then sets up early stopping with a patience of 5 epochs to prevent overfitting and save the best model weights using a checkpoint mechanism. The `model.fit()` function trains the model using training data `X_train` and `y_train`, validating on `X_test` and `y_test`, with 20

epochs and a batch size of 32. During training, it utilizes callbacks `es` (EarlyStopping) and `mc` (ModelCheckpoint) to monitor validation loss, ensuring that the model stops training early if the validation loss does not improve after 5 epochs and saving only the best model weights based on validation loss.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
mc = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=32, callbacks=[es, mc])
```

```
Epoch 1/20
401/401 [==============================] - 159s 388ms/step - loss: 1.5118 - accuracy: 0.4696 - val_loss: 1.0950 - val_accuracy: 0.5868
Epoch 2/20
401/401 [==============================] - 152s 378ms/step - loss: 1.1398 - accuracy: 0.5742 - val_loss: 1.0871 - val_accuracy: 0.5655
Epoch 3/20
401/401 [==============================] - 152s 379ms/step - loss: 1.0393 - accuracy: 0.6043 - val_loss: 1.0686 - val_accuracy: 0.5964
Epoch 4/20
401/401 [==============================] - 161s 403ms/step - loss: 0.9821 - accuracy: 0.6235 - val_loss: 0.9086 - val_accuracy: 0.6470
Epoch 5/20
401/401 [==============================] - 151s 377ms/step - loss: 0.9396 - accuracy: 0.6415 - val_loss: 0.8961 - val_accuracy: 0.6545
Epoch 6/20
401/401 [==============================] - 156s 389ms/step - loss: 0.9058 - accuracy: 0.6555 - val_loss: 0.8801 - val_accuracy: 0.6514
Epoch 7/20
401/401 [==============================] - 153s 380ms/step - loss: 0.8762 - accuracy: 0.6714 - val_loss: 0.9114 - val_accuracy: 0.6514
Epoch 8/20
401/401 [==============================] - 157s 392ms/step - loss: 0.8401 - accuracy: 0.6839 - val_loss: 0.8645 - val_accuracy: 0.6676
Epoch 9/20
401/401 [==============================] - 156s 389ms/step - loss: 0.8184 - accuracy: 0.6925 - val_loss: 0.8276 - val_accuracy: 0.6826
Epoch 10/20
401/401 [==============================] - 151s 377ms/step - loss: 0.7920 - accuracy: 0.7034 - val_loss: 0.7842 - val_accuracy: 0.6916
Epoch 11/20
401/401 [==============================] - 152s 379ms/step - loss: 0.7826 - accuracy: 0.7064 - val_loss: 0.8479 - val_accuracy: 0.6798
Epoch 12/20
401/401 [==============================] - 158s 395ms/step - loss: 0.7471 - accuracy: 0.7213 - val_loss: 0.7167 - val_accuracy: 0.7272
Epoch 13/20
401/401 [==============================] - 152s 379ms/step - loss: 0.7166 - accuracy: 0.7333 - val_loss: 0.8160 - val_accuracy: 0.7001
Epoch 14/20
401/401 [==============================] - 155s 386ms/step - loss: 0.7095 - accuracy: 0.7369 - val_loss: 0.7308 - val_accuracy: 0.7150
Epoch 15/20
401/401 [==============================] - 152s 378ms/step - loss: 0.6786 - accuracy: 0.7513 - val_loss: 0.8552 - val_accuracy: 0.6951
Epoch 16/20
401/401 [==============================] - 152s 378ms/step - loss: 0.6669 - accuracy: 0.7551 - val_loss: 0.7332 - val_accuracy: 0.7232
Epoch 17/20
401/401 [==============================] - 152s 380ms/step - loss: 0.6506 - accuracy: 0.7584 - val_loss: 0.7125 - val_accuracy: 0.7391
Epoch 18/20
401/401 [==============================] - 151s 377ms/step - loss: 0.6322 - accuracy: 0.7650 - val_loss: 0.6642 - val_accuracy: 0.7528
Epoch 19/20
401/401 [==============================] - 154s 385ms/step - loss: 0.6273 - accuracy: 0.7675 - val_loss: 0.6564 - val_accuracy: 0.7594
Epoch 20/20
401/401 [==============================] - 155s 388ms/step - loss: 0.5939 - accuracy: 0.7801 - val_loss: 0.7496 - val_accuracy: 0.7325
```

Figure 9. Train a Neural Network Model

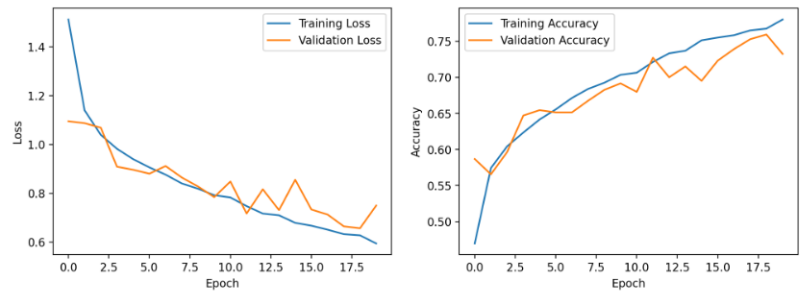
Display Model Accuracy

The provided code snippet (Figure 11) visualizes the training and validation performance metrics of a neural network model using Matplotlib. It creates a figure with two subplots: the first subplot plots the training and validation loss against epochs, helping to assess how well the model is minimizing its error during training and validation phases. The second subplot depicts the training and validation accuracy across epochs, showing how well the model is learning to classify the data correctly. These plots are essential for diagnosing potential issues such as overfitting or underfitting. After plotting, the code calculates predictions (`y_pred`) for the test set using the trained model and computes the accuracy score (`acc`) by comparing these predictions with the actual labels (`y_test`). This score quantitatively evaluates the model's performance on unseen data, providing insights into its effectiveness for classification tasks.

```
# Evaluate the model
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

y_pred = np.argmax(model.predict(X_test), axis=1)
acc = accuracy_score(y_test, y_pred)
print(f"Test accuracy: {acc}")
```




```
101/101 [==============================] - 10s 100ms/step
Test accuracy: 0.732521847690387
```

Figure 10. Display Model Accuracy

Predicting the Labels of Test Data

The provided code snippet (Figure 12) performs two critical tasks for evaluating a classification model's performance. First, it generates a confusion matrix (`cm`) using the `confusion_matrix` function from scikit-learn, which compares the predicted labels (`y_pred`) against the actual labels (`y_test`) of the test dataset. This matrix visualizes the count of true positive, false positive, true negative, and false negative predictions for each emotion category, facilitating an assessment of the model's accuracy in classifying different emotions. Secondly, it utilizes Seaborn's `heatmap` function to display the confusion matrix as a color-coded grid, with annotations (`annot=True`) showing the exact count in each cell (`fmt='d'`). This visualization aids in identifying which emotions are most often confused with each other by the model. Additionally, the code prints a `classification_report` which provides comprehensive metrics including precision, recall, F1-score, and support for each emotion category. These metrics offer deeper insights into the model's overall performance and its ability to correctly classify emotions based on the provided dataset.

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=emotions, yticklabels=emotions, cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print(classification_report(y_test, y_pred, target_names=emotions))
```

Figure 11. Predicting the Labels of Test Data

Emotion Messages

The code snippet (Figure 13) utilizes Google Colab's files.upload() function to allow users to upload audio files interactively. Once the file is uploaded, the script iterates through each uploaded file (uploaded.keys()), constructs the file path, and then calls a hypothetical function predict_emotion(path) to predict the emotion conveyed in the audio. The predicted emotion is then used to retrieve a corresponding message from the emotion_messages dictionary, which provides personalized feedback based on the predicted emotion. This workflow enables real-time emotion detection and feedback for uploaded audio files within the Google Colab environment.

```
# Dictionary mapping emotion labels to custom messages
emotion_messages = {
    0: "angry: It's okay to feel angry. Take a deep breath and try to relax.",
    1: "disgust: Something might be bothering you. Try to avoid the situation causing this feeling.",
    2: "fear: It's natural to feel scared. Remember to stay calm and seek support if needed.",
    3: "happy: Great to see you're happy! Keep up the positive vibes!",
    4: "neutral: You're feeling neutral. Stay balanced and continue your day.",
    5: "sad: You seem sad. Take some rest, and don't worry too much. Things will get better.",
    6: "surprise: Something surprising happened! Embrace the unexpected.",
    7: "fatigue: You seem tired. Make sure to get some rest and take care of yourself."
}
```

```
from google.colab import files
import numpy as np
import librosa
import tensorflow as tf

# Upload audio file
uploaded = files.upload()
for fn in uploaded.keys():
    path = '/content/' + fn
    predicted_emotion = predict_emotion(path)
    emotion_message = emotion_messages[predicted_emotion]
    print(f'The predicted emotion for the uploaded audio file {fn} is: {emotion_message}')
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Figure 13. Emotion Messages

Sending WhatsApp Messages

The script (Figure 14) is used to automate sending WhatsApp messages using Selenium WebDriver in Python. First, it prompts the user to input the recipient's phone number and the message to be sent. Then, it constructs a WhatsApp Web URL with the provided phone number and message. The WebDriver is used to open this WhatsApp Web URL, waits until the "Type a message" element appears using WebDriverWait, and then identifies the send button using XPath. Subsequently, it sends the message by clicking the appropriate send button.

```
print('Write recipient phone number in format +[Country Code][Area Code][Rest]')
notelp = str(input())
print("\nWrite message:")
message = str(input())
site = "https://web.whatsapp.com/send?phone={notelp}&text={quote{message}}"
wd.get(site)
#wd.get(link)
wd.implicitly_wait(10)
waiter = WebDriverWait(wd, 10)
time.sleep(1)
while not page_is_loading(wd):
  continue
waiter = WebDriverWait(wd, 10)
wd.implicitly_wait(20)
#input_box = waiter.until(EC.presence_of_element_located(( By.XPATH, ('//div[contains(text(), "Ketik pesan")]')))
#input_box = waiter.until(EC.presence_of_element_located(( By.XPATH, ('//div[contains(text(), "Ketik pesan")]')))

#input_box = wd.find_elements_by_xpath('//div[contains(text(), "Ketik pesan")]')
#print (input_box)
element = lambda d : d.find_elements(by=By.XPATH, value="//div/button/span[@data-icon='send']")

# Wait until send is found (in case of login)
loaded = WebDriverWait(wd, 50).until(method=element, message="User never signed in")

wd.implicitly_wait(10)
send = element(wd)[0]
time.sleep(5)
wd.close()
```

Figure 12. Sending WhatsApp Messages

CONCLUSION

This study concludes that integrating speech emotion detection into workplace safety systems significantly enhances safety by providing real-time insights into workers' emotional states, helping prevent accidents related to emotional stress and fatigue. The *Convolutional Neural Network* (CNN) classifier demonstrates superior accuracy compared to traditional methods like *Support Vector Machines* (SVM), validating its effectiveness for real-time emotion detection across a broad range of emotions including happiness, sadness, anger, and fear. However, the research exhibits limitations that necessitate future improvements, including expanding language capabilities to incorporate multiple languages such as Indonesian for broader applicability, optimizing detection algorithms to reduce training and detection time for faster real-time applications, and conducting extensive field testing by integrating the CNN-based emotion detection classifier into real-world work safety systems to provide practical insights into performance and identify areas for further improvement.

REFERENCES

Abedi, H., Ma, M., He, J., Yu, J., Ansariyan, A., & Shaker, G. (2023). Deep learning-based in-cabin monitoring and vehicle safety system using a 4D imaging radar sensor. *IEEE Sensors Journal*.

Ahmed, M. R., Islam, S., Islam, A. M., & Shatabda, S. (2023). An ensemble 1D-CNN-LSTM-GRU model with data augmentation for speech emotion recognition. *Expert Systems with Applications*, *218*, Article 119633.

Al-Dujaili, M. J., & Ebrahimi-Moghadam, A. (2023). Speech emotion recognition: A comprehensive survey. *Wireless Personal Communications*, *129*(4), 2525-2561.

Alsabhan, W. (2023). Human-computer interaction with a real-time speech emotion recognition with ensembling techniques 1D convolution neural network and attention. *Sensors*, *23*(3), Article 1386.

An, H. (2023, May). Speech recognition of speaker identity based on convolutional neural networks. In *International Conference on Computer Graphics, Artificial Intelligence, and Data Processing (ICCAID 2022)* (Vol. 12604, pp. 653-659). SPIE.

Atkar, S. N., Agrawal, R., Dhule, C., Morris, N. C., Saraf, P., & Kalbande, K. (2023, May). Speech emotion recognition using dialogue emotion decoder and CNN classifier. In *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)* (pp. 94-99). IEEE.

Bertero, D., & Fung, P. (2017). A first look into a convolutional neural network for speech emotion detection. *Proceedings of International Conference on Acoustics, Speech and Signal Processing* (pp. 5115-5119). Clear Water Bay.

Bhanusree, Y., Kumar, S. S., & Rao, A. K. (2023). Time-distributed attention-layered convolution neural network with ensemble learning using random forest classifier for speech emotion recognition. *Journal of Information and Communication Technology*, *22*(1), 49-76.

Chen, L. W., & Rudnicky, A. (2023, June). Exploring Wav2vec 2.0 fine tuning for improved speech emotion recognition. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 1-5). IEEE.

Deepika, C., & Kuchibhotla, S. (2023). Feature extraction model for speech emotion detection with prodigious precedence assortment model using fuzzy-based convolution neural networks. *Soft Computing*, 1-9.

Feng, T., Hebbar, R., & Narayanan, S. (2023). Trustser: On the trustworthiness of fine-tuning pre-trained speech embeddings for speech emotion recognition. *arXiv preprint arXiv:2305.11229*.

Hamid, B., Aleissa, E., & Rauf, A. (2012). Anticipating software fault proneness using classifier ensemble: An optimized approach. *Proceedings of the IASTED International Conference on Software Engineering* (pp. 1-6).

Isaac, S., Haruna, K., Ahmad, M. A., & Mustapha, R. (2023). Deep reinforcement learning with hidden Markov model for speech recognition. *Journal of Technology and Innovation*, 01-05.

Jain, M., Narayan, S., Balaji, P., Bharath, K. P., Bhowmick, A., Khartik, R., & Muthu, R. K. (2017). Speech emotion recognition using support vector machine. *International Journal of Smart Home*, *6*(2), 101-108.

Jain, U., Nathani, K., Ruban, N., Raj, A. N. J., Zhuang, Z., & Mahesh, V. G. V. (2018). Cubic SVM classifier based feature extraction and emotion detection from speech. *Proceedings of International Conference on Sensor Networks and Signal Processing* (pp. 386-391). Xi'an.

Lakshmi, K. L., Muthulakshmi, P., Nithya, A. A., Jeyavathana, R. B., Usharani, R., Das, N. S., & Devi, G. N. R. (2023). Recognition of emotions in speech using deep CNN and RESNET. *Soft Computing*, 1-17.

Li, W., Xue, J., Tan, R., Wang, C., Deng, Z., Li, S., Guo, G., & Cao, D. (2023). Global-local-feature-fused driver speech emotion detection for intelligent cockpit in automated driving. *IEEE Transactions on Intelligent Vehicles*.

Liu, Z. T., Han, M. T., Wu, B. H., & Rehman, A. (2023). Speech emotion recognition based on convolutional neural network with attention-based bidirectional long short-term memory network and multi-task learning. *Applied Acoustics*, *202*, Article 109178.

Mantegazza, I., & Ntalampiras, S. (2023, June). Italian speech emotion recognition. In *2023 24th International Conference on Digital Signal Processing (DSP)* (pp. 1-5). IEEE.

Min, C., Lin, H., Li, X., Zhao, H., Lu, J., Yang, L., & Xu, B. (2023). Finding hate speech with auxiliary emotion detection from self-training multi-label learning perspective. *Information Fusion*, *96*, 214-223.

Mukherjee, S., Mundra, S., & Mundra, A. (2023). Speech emotion recognition using convolutional neural networks on spectrograms and mel-frequency cepstral coefficients images. In *Information and Communication Technology for Competitive Strategies (ICTCS 2022) Intelligent Strategies for ICT* (pp. 33-41). Springer Nature Singapore.

Naren, M., & Sandhiya, M. (2023, May). Enhancing the accuracy in classifying human emotion via speech recognition using novel support vector machine compared with convolution neural network. In *AIP Conference Proceedings* (Vol. 2602, No. 1). AIP Publishing.

Pan, S. T., & Wu, H. J. (2023). Performance improvement of speech emotion recognition systems by combining 1D CNN and LSTM with data augmentation. *Electronics*, *12*(11), Article 2436.

Prabhakar, G. A., Basel, B., Dutta, A., & Rao, C. V. R. (2023). Multichannel CNN-BLSTM architecture for speech emotion recognition system by fusion of magnitude and phase spectral features using DCCA for consumer applications. *IEEE Transactions on Consumer Electronics*.

Prakash, P. R., Anuradha, D., Iqbal, J., Galety, M. G., Singh, R., & Neelakandan, S. (2023). A novel convolutional neural network with gated recurrent unit for automated speech emotion recognition and classification. *Journal of Control and Decision*, *10*(1), 54-63.

Sadovsky, E., Jakubec, M., & Jarina, R. (2023, April). Speech command recognition based on convolutional spiking neural networks. In *2023 33rd International Conference Radioelektronika (Radioelektronika)* (pp. 1-5). IEEE.

Saleem, N., Gao, J., Irfan, R., Almadhor, A., Rauf, H. T., Zhang, Y., & Kadry, S. (2023). DeepCNN: Spectro-temporal feature representation for speech emotion recognition. *CAAI Transactions on Intelligence Technology*.

Sarker, S., Akter, K., & Mamun, N. (2023, February). A text independent speech emotion recognition based on convolutional neural network. In *2023 International Conference on Electrical, Computer and Communication Engineering (ECCE)* (pp. 1-4). IEEE.

Semwal, N., Kumar, A., & Narayanan, S. (2017). Automatic speech emotion detection system using multi-domain acoustic feature selection and classification models. *International Conference on Identity, Security, and Behavior Analysis* (pp. 1-6). Visakhapatnam.

Shahin, I., Alomari, O. A., Nassif, A. B., Afyouni, I., Hashem, I. A., & Elnagar, A. (2023). An efficient feature selection method for Arabic and English speech emotion recognition using Grey Wolf Optimizer. *Applied Acoustics*, *205*, Article 109279.

Sharma, D., Cheema, A. P., Reddy, K. K., Reddy, C. K., Ram, G. B., Avinash, G., & Reddy, P. K. (2023, January). Speech emotion recognition system using SVD algorithm with HMM model. In *2023 International Conference for Advancement in Technology (ICONAT)* (pp. 1-5). IEEE.

Shegokar, P., & Sircar, P. (2016). Continuous wavelet transform based speech emotion recognition. *International Conference on Signal Processing and Communication Systems (ICSPCS)* (pp. 1-8). Surfer's Paradise.

Singh, J., Saheer, L. B., & Faust, O. (2023). Speech emotion recognition using attention model. *International Journal of Environmental Research and Public Health*, *20*(6), Article 5140.

Singh, V., & Prasad, S. (2023). Speech emotion recognition system using gender dependent convolution neural network. *Procedia Computer Science*, *218*, 2533-2540.

Vasuki, P. (2023). Design of hierarchical classifier to improve speech emotion recognition. *Computer Systems Science & Engineering*, *44*(1).

Wagner, J., Triantafyllopoulos, A., Wierstorf, H., Schmitt, M., Burkhardt, F., Eyben, F., & Schuller, B. W. (2023). Dawn of the transformer era in speech emotion recognition: Closing the valence gap. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhao, J., Mao, X., & Chen, L. (2019). Speech emotion recognition using deep 1D & 2D CNN LSTM networks. *Biomedical Signal Processing and Control*, *47*, 312-323.

Zhao, Y., Guo, M., Sun, X., Chen, X., & Zhao, F. (2023). Attention-based sensor fusion for emotion recognition from human motion by combining convolutional neural network and weighted kernel support vector machine and using inertial measurement unit signals. *IET Signal Processing*, *17*(4), Article e12201.