DEEP LEARNING PROJECT REPORT

On

STRESS DETECTION USING FACIAL EXPRESSIONS

Submitted in partial fulfillment of the

Requirements for the award of the Degree of

Bachelor of Technology

in

COMPUTER SCIENCE ENGINEERING

Under guidance of

Dr. P. LAKSHMI PRASANNA

Associate Professor, Dept. of CSE
By

VELAGA KRISHNA BABU	**-**	**2100039030**
MOHAMMAD ISMAIL	**-**	**2100039032**
MURIKITI VAMSI	**-**	**2100039039**
MD. ANEES MEHBOOB	**-**	**2100039047**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

K L E F

Green Fields, Vaddeswaram, Guntur District-522 502

2023-2024

K L E F

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project Report entitled "**STRESS DETECTION USING FACIAL EXPRESSIONS**" is being submitted by **VELAGA KRISHNA BABU (2100039030), MOHAMMAD ISMAIL (2100039032), MURIKITI VAMSI (2100039039), MD. ANEES MEHBOOB (2100039047)** in partial fulfillment for the award of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE ENGINEERING** to the K L University is a record of bonafide work carried out under our guidance and supervision. The results embodied in this reporthave not been copied from any other departments/ University/Institute.

FACULTY INCHARGE	**HEAD OF THE DEPARTMENT**
Dr. P. LAKSHMI PRSANNA	**Dr. T. PAVAN KUMAR**

K L E F

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DECLARATION

The Project Report entitled "**STRESS DETECTION USING FACIAL EXPRESSIONS**" This a record of bonafide work of **VELAGA KRISHNA BABU (2100039030), MOHAMMAD ISMAIL (2100039032), MURIKITI VAMSI (2100039039), MD. ANEES MEHBOOB (2100039047)** submitted in partial fulfillment for the award of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE ENGINEERING** to the K L University. The results embodied in this report have not been copied from any other departments/ University/ Institute.

VELAGA KRISHNA BABU (2100039030)

MOHAMMAD ISMAIL (2100039032)

MURIKITI VAMSI (2100039039)

MD. ANEES MEHBOOB (2100039047)

ACKNOWLEDGEMENT

We express sincere gratitude to our Coordinator **Dr. P. LAKSHMI PRSANNA** for her leadership and constant motivation provided in successful completion of our academic semester.

We record it as my privilege to deeply thank our pioneer **Dr. T. PAVAN KUMAR** Head of the Department of COMPUTER SCIENCE ENGINEERING for providing us the efficient faculty and facilities to make our ideas into reality.

We are greatly indebted to our K L Deemed to be university that has provided a healthy environment to drive me to achieve my ambitions and goals. We would like to express our sincere thanks to our project in charges, for the guidance, support, and assistance they have provided in completing this project.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectlyto make this project report success.

ABSTRACT

Stress is a pervasive issue affecting individuals across various aspects of life, with significant implications for mental and physical well-being. Early detection of stress plays a crucial role in preventing adverse outcomes and promoting timely interventions. This project aims to develop a novel approach for stress detection based on facial expressions using deep learning techniques. Leveraging advances in computer vision and artificial intelligence, we propose a deep learning model trained on a comprehensive dataset of facial expressions to accurately identify signs of stress. The methodology involves pre-processing facial images, designing and training a deep neural network architecture, and evaluating its performance using appropriate metrics. The results demonstrate the effectiveness of the proposed approach in accurately detecting stress from facial expressions, with promising implications for real-world applications in healthcare, wellness, and human-computer interaction. Through this project, we contribute to the ongoing efforts in leveraging technology for mental health assessment and intervention, ultimately aiming to improve the quality of life for individuals affected by stress-related issues.

Stress detection based on facial expressions using deep learning is a critical endeavor in mental health assessment and intervention. Facial cues offer valuable insights into an individual's emotional state, making them a promising avenue for early stress detection. This research project focuses on developing a predictive model for discerning stress-related facial expressions using deep learning techniques.

The study capitalizes on a dataset comprising facial images captured under various stress-inducing scenarios, providing a diverse range of expressions to train the model effectively. Deep learning architectures, such as convolutional neural networks (CNNs), are chosen for their ability to extract intricate features from images and discern subtle patterns indicative of stress.

In the course of the investigation, the dataset undergoes preprocessing to enhance the model's robustness and generalization capabilities. This includes techniques such as image augmentation, normalization, and feature extraction to ensure optimal performance.

The deep learning model is then trained on the preprocessed dataset to learn the complex relationships between facial expressions and stress levels. Leveraging techniques such as transfer learning, the model can leverage knowledge gained from pre-trained networks to improve

performance even with limited data.

Performance metrics, including accuracy, precision, recall, and F1-score, are utilized to evaluate the model's effectiveness in detecting stress from facial expressions accurately. Comparative analysis with other machine learning approaches provides insights into the competitiveness and efficacy of deep learning for stress detection.

Overall, this research contributes to the advancement of stress detection methodologies, offering a promising approach for early intervention and support in mental health care settings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

LR	Logistic Regression
LDA	Linear Discriminant Analysis
KNN	K-nearest neighbors
CART	Classification and Regression Tree
NB	Naïve Bayes Classifier
SVM	Support Sector Machine
SLR	Scaled Logistic Regression
SLDA	Scaled Linear Discriminant Analysis
SKNN	Scaled K-nearest neighbors
SCART	Scaled Classification and Regression
SNB	Scaled Naïve Bayes Classifier
SSVM	Scaled Support Sector Machine

INTRODUCTION

In today's fast-paced and demanding world, stress has become a prevalent issue affecting individuals across various demographics. From students facing academic pressure to professionals dealing with work-related stressors, the impact of stress on mental and physical well-being cannot be understated. Chronic stress not only deteriorates one's quality of life but also contributes to the development of serious health conditions such as anxiety, depression, and cardiovascular diseases. Recognizing the importance of early intervention in managing stress, researchers and healthcare professionals are increasingly turning to innovative technologies for effective detection and monitoring.

One such technology that has garnered significant attention in recent years is the use of facial expressions as a means of assessing stress levels. Human faces are rich sources of information, conveying a myriad of emotions and psychological states. The subtle changes in facial expressions, such as furrowed brows, tense jawlines, or downturned lips, often reflect underlying emotional turmoil, including stress. Leveraging advancements in computer vision and artificial intelligence, researchers are exploring the potential of automated systems to detect and analyze these facial cues, providing valuable insights into an individual's stress levels in real-time.

Traditional methods of stress assessment, such as self-report questionnaires or physiological measurements, have limitations in terms of subjectivity and invasiveness. In contrast, facial expression analysis offers a non-intrusive and objective approach to stress detection, making it suitable for a wide range of applications, including mental health screening, workplace stress management, and human-computer interaction. By harnessing the power of deep learning, a subset of artificial intelligence, researchers aim to develop robust and accurate models capable of discerning stress-related facial expressions with high precision and reliability.

The premise of using deep learning for stress detection lies in its ability to automatically learn intricate patterns and features from large volumes of data. Deep learning models, particularly convolutional neural networks (CNNs), have demonstrated remarkable capabilities in image recognition and classification tasks, making them well-suited for analyzing facial expressions. By training these models on annotated datasets comprising facial images labeled with corresponding stress levels, researchers can teach the algorithms to recognize subtle cues indicative of stress, such as changes in facial muscle movements, skin coloration, or eye gaze patterns.

The development of a reliable stress detection system based on facial expressions using deep learning holds immense potential for various stakeholders. For individuals, such a system could provide timely feedback on their stress levels, enabling proactive self-management strategies and seeking appropriate support when needed. In educational settings, educators and counselors could utilize these technologies to identify students experiencing elevated stress levels and offer targeted interventions to promote mental well-being. In the workplace, employers could implement stress detection systems to monitor employee well-being and create a supportive work environment conducive to productivity and resilience.

Despite the promising prospects of facial expression analysis for stress detection, several challenges persist. Variability in facial expressions across different individuals, cultures, and contexts presents a significant obstacle in model generalization. Moreover, the ethical considerations surrounding privacy, data security, and potential biases inherent in automated systems warrant careful scrutiny and mitigation strategies.

In light of these considerations, this research endeavors to contribute to the burgeoning field of stress detection based on facial expressions using deep learning. By exploring novel methodologies, leveraging state-of-the-art technologies, and addressing existing challenges, we aim to develop a robust and reliable system capable of accurately assessing stress levels from facial cues. Through interdisciplinary collaboration and empirical validation, we aspire to advance the frontiers of mental health technology and foster a healthier, more resilient society.

OBJECTIVE

1. **Develop a Comprehensive Dataset:**
   - Compile a diverse and well-annotated dataset of facial images depicting a range of stress levels across different individuals, demographics, and scenarios.
   - Ensure dataset integrity and quality by addressing issues such as bias, imbalance, and variability in facial expressions.

2. **Design and Train Deep Learning Models**:
   - Investigate and select appropriate deep learning architectures, such as convolutional neural networks (CNNs), for facial expression analysis and stress detection.
   - Implement and fine-tune deep learning models using the collected dataset to learn complex patterns and features indicative of stress-related facial expressions.

3. **Evaluate Model Performance**:
   - Assess the accuracy, precision, recall, and F1-score of the trained deep learning models in detecting stress from facial expressions.
   - Conduct comprehensive performance evaluations using cross-validation, hold-out validation, and other robust validation techniques to ensure model reliability and generalization.

4. **Address Ethical and Privacy Considerations**:
   - Identify and address ethical considerations surrounding the use of facial expression analysis for stress detection, including privacy, consent, and potential biases.
   - Implement privacy-preserving measures and data anonymization techniques to safeguard individuals' confidentiality and mitigate risks associated with data misuse or unauthorized access.

5. **Compare with Existing Methods**:
   - Benchmark the performance of the developed deep learning models against existing methods of stress detection, including self-report questionnaires, physiological measurements, and traditional machine learning approaches.
   - Conduct comparative analyses to evaluate the advantages, limitations, and practical implications of facial expression-based stress detection methods.

6. **Explore Real-World Applications**:
   - Investigate potential real-world applications and deployment scenarios for the developed stress detection system, including mental health screening, workplace stress management, and human-computer interaction.
   - Collaborate with relevant stakeholders, such as mental health professionals, educators, and technology developers, to explore opportunities for integrating the system into existing frameworks and interventions.

7. **Iterative Improvement and Validation**:
   - Continuously iterate and refine the deep learning models based on feedback, empirical observations, and validation results.
   - Validate the effectiveness and usability of the stress detection system through pilot studies, user evaluations, and feedback mechanisms to ensure alignment with end-user needs and expectations.

By pursuing these objectives, this research aims to advance the state-of-the-art in stress detection based on facial expressions using deep learning, ultimately contributing to the development of innovative technologies for mental health assessment and intervention.

LITERATURE SURVEY

The article "Stress and anxiety detection using facial cues from videos" explores the use of facial cues to detect stress and anxiety. Published in Biomedical Signal Processing and Control in 2017, the study by Giannakakis et al. focuses on analyzing facial expressions to identify emotional states. This research has implications for developing tools to monitor mental health through video analysis.

The study detects stress and anxiety by analyzing facial cues from videos. Researchers use facial expression analysis to identify patterns associated with stress and anxiety. By examining muscle movements and facial expressions, the study aims to detect emotional states such as stress and anxiety. This approach leverages the relationship between facial cues and emotional responses to develop a method for detecting and monitoring stress and anxiety levels in individuals.

The methodologies used for stress detection in the study "Stress and anxiety detection using facial cues from videos" by Giannakakis et al. include analyzing facial cues from videos. Researchers examine facial expressions and muscle movements to identify patterns associated with stress and anxiety. By leveraging facial expression analysis, the study aims to detect and monitor stress levels in individuals. This approach utilizes the relationship between facial cues and emotional responses to develop a method for stress detection through video analysis.

The study "Detecting Negative Emotional Stress Based on Facial Expression in Real Time" by Zhang et al. delves into the realm of real-time detection of negative emotional stress through facial expressions. Presented at the 2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP), this research sheds light on the potential of utilizing facial cues to identify and monitor negative emotional stress levels in individuals. The study, downloaded on November 18, 2019, carries the DOI 10.1109/siprocess.2019.8868735.

Emotional stress is a prevalent aspect of human experience, impacting individuals' well-being and mental health. Detecting and managing stress levels is crucial for maintaining overall health and quality of life. Traditional methods of stress assessment often rely on self-reporting or physiological measurements, which may have limitations in terms of accuracy and real-time monitoring. In this context, the study by Zhang et al. explores an innovative approach that leverages facial expressions as indicators of negative emotional stress.

Facial expressions are known to convey a wealth of information about an individual's emotional state. By analyzing facial cues such as muscle movements, researchers can discern patterns associated with stress and anxiety. This study aims to capitalize on the relationship between facial expressions and emotional responses to develop a method for real-time detection of negative emotional stress. The ability to detect stress levels promptly and accurately through facial analysis could revolutionize the field of mental health monitoring.

The research methodology employed in this study involves the analysis of facial expressions from videos. By examining facial cues and muscle movements, the researchers seek to identify specific patterns that correspond to negative emotional stress. This approach entails the use of advanced algorithms and tools for automatic facial expression recognition, enabling real-time assessment of emotional states. The study's innovative methodology opens up new possibilities for non-invasive monitoring of stress levels through video analysis.

The implications of this research are significant for both the scientific community and the broader field of mental health monitoring. The ability to detect negative emotional stress in real time using facial expressions could pave the way for the development of novel tools and technologies for stress management and intervention. By harnessing the power of facial cues, researchers can potentially offer individuals a more accessible and efficient means of monitoring their emotional well-being.

In conclusion, the study by Zhang et al. represents a pioneering effort in the realm of real-time detection of negative emotional stress based on facial expression analysis. By leveraging the rich information conveyed through facial cues, the research contributes to advancing the field of mental health monitoring and offers promising avenues for future research and application in stress management.

METHODOLOGY

The process and methods used for proposing the prediction model is discussed in this section.

DATASET:

In our proposed method dataset has been collected from Kaggle Repository. It has come across 10 features which define and differentiate Emotions comprises of 30000 images.
https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset/download?datasetVersionNumber=1

EXPERIMENTAL SETTING:

In building of this system we areusing Pycharm IDE tool for the purpose of implementing the varied feature selection and model systems. The main motive is to measure the predicting efficiency of the classifier when it is functional and operating and then classifying new samples outside the benefitof perceiving the bona fide class of the samples. The comparators have been designed to implement a 10-fold cross validation trial. The dataset is split into 10 equally distributed subsets. The most exact machine learning classifier is chosen asa base classifier to instruct the nine-subset layer and examine it on the last subset layer. To measure the durability of crafted groundwork, the step is repeated. To appraise the performance of the considered framework, seven different specifications listed as, F measure, accuracy, MCC, error rate, True and False Positive rates, and area under curve(AUC) are used.

MODELS:

1. Sequential Model for Binary Classification:

This section involves the development and implementation of a sequential neural network model for binary classification.
The architecture typically consists of input, hidden, and output layers, with activation functions such as ReLU in the hidden layers and sigmoid in the output layer for binary classification.
The model is trained using a dataset where each data instance belongs to one of two classes, and performance is evaluated using metrics like accuracy, precision, recall, and F1-score.

2. Optimization Techniques Comparison:

Here, the same sequential model from the previous section is trained using different optimization techniques, such as ADAM, SGD (Stochastic Gradient Descent), and RMSPROP (Root Mean Square Propagation).
Each optimization technique has its unique update rules and hyperparameters, which affect the model's convergence speed and final performance.
The comparative analysis aims to identify the most suitable optimization technique for the given

classification task based on factors like convergence behavior and computational efficiency.

3. Comparative Analysis of Models:

This section involves a detailed comparison of the performance of the sequential models trained with different optimization techniques.

Metrics such as accuracy, loss curves, and convergence speed are compared to assess the effectiveness of each optimization technique.

Insights gained from the comparative analysis help in understanding the trade-offs between optimization techniques and selecting the most appropriate one for the task at hand.

4. Sequential Model for Multi-Class Classification:

In this section, the sequential neural network model is extended to handle multi-class classification tasks where each data instance belongs to one of multiple classes.

The architecture is modified to have multiple output nodes corresponding to the number of classes, with softmax activation function in the output layer.

Model performance is evaluated using metrics suitable for multi-class classification, such as categorical cross-entropy loss and accuracy.

5. Binary Classification with Mini-Batch Evaluations:

Mini-batch evaluations involve updating model parameters using a subset of the training data (mini-batch) instead of the entire dataset in each iteration.

This section explores the impact of mini-batch size on model training efficiency, convergence speed, and generalization.

Performance metrics and convergence behavior are analyzed for different mini-batch sizes to determine the optimal setting for the given classification task.

6. Convolutional Neural Network (CNN):

Design and implement a CNN architecture for image classification tasks.

Train the CNN on image datasets, leveraging convolutional layers, pooling layers, and fully connected layers.

Evaluate the CNN's performance in comparison to the sequential model for image classification tasks.

7. CNN with Regularization:

Extend the CNN model by incorporating regularization techniques such as dropout or L2 regularization.

Train the regularized CNN on image datasets to prevent overfitting and improve generalization

performance.
Compare the performance of the regularized CNN with the baseline CNN model.

8. VGG (Visual Geometry Group) Model:

Implement the VGG architecture, a popular CNN model known for its depth and performance.
Train the VGG model on image datasets and evaluate its classification accuracy and performance compared to other models.

9. Recurrent Neural Network (RNN):

Design and implement an RNN model for sequential data analysis, such as time-series prediction or text classification.
Train the RNN on sequential datasets and evaluate its performance using appropriate metrics.

10. CNN + LSTM (Long Short-Term Memory) Model:

Combine CNN and LSTM architectures to create a hybrid model capable of handling both spatial and temporal dependencies in data.
Train the CNN + LSTM model on datasets with sequential and spatial information, such as video data or sensor data.
Evaluate the model's performance and compare it with standalone CNN and RNN models.

11. Web Application Integration:

Integrate all classification models developed into a web application using frameworks like Flask or Django.
Create a user-friendly interface for uploading data, selecting models, and displaying classification results.
Deploy the web application to a server for real-time usage and demonstrate its functionality with sample datasets.

SYSTEM REQUIREMENTS

Hardware requirements :

■Ram: 8 GB and more	
■Processor:	Any Intel Processor
■Hard Disk:	6 GB and more
■Speed:1 GHZ and more	

Software Requirements

■Operating System:	Windows10 and above. ■
Language:	Python 3.8.5
IDE :	Pycharm Community Edition
Libraries:	

1. **TensorFlow/Keras:**
   - TensorFlow is an open-source machine learning library developed by Google Brain.
   - Keras is a high-level neural networks API that runs on top of TensorFlow.
   - Together, TensorFlow and Keras provide a powerful framework for building, training, and deploying deep learning models.
2. **NumPy:**
   - NumPy is a fundamental package for scientific computing in Python.
   - It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

3. **Pandas:**
    - Pandas is a fast, powerful, and flexible data analysis and manipulation library for Python.
    - It provides data structures like DataFrame and Series, which are ideal for handling structured data, such as datasets containing facial expressions.
4. **Matplotlib and Seaborn:**
    - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
    - Seaborn is a statistical data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
5. **OpenCV (Open Source Computer Vision Library):**
    - OpenCV is a library of programming functions mainly aimed at real-time computer vision tasks.
    - It provides tools and algorithms for image processing, feature detection, object recognition, and more, making it useful for tasks like facial detection and preprocessing.
6. **Scikit-learn:**
    - Scikit-learn is a simple and efficient library for machine learning in Python.
    - It features various classification, regression, and clustering algorithms, along with tools for model selection and evaluation.
7. **Pillow (Python Imaging Library):**
    - Pillow is a fork of the Python Imaging Library (PIL), adding support for opening, manipulating, and saving many different

image file formats.

- It's commonly used for image preprocessing tasks, such as resizing, cropping, and enhancing images.

8. **Scipy:**

- Scipy is a scientific computing library that builds on NumPy, providing additional functionality for optimization, integration, interpolation, and more.
- It includes modules like scipy.optimize for optimization tasks and scipy.signal for signal processing operations.

IMPLEMENTATION

DATA PREPROCESSIG:

Image Dataset Preparation:

Using tf.keras.utils.image.ImageDataGenerator to create training and validation datasets.

Split the data into training and validation sets using a 80-20 split ratio.

Resize images to the specified dimensions and convert them into batches of the defined batch size.

Class Names Extraction:

Extract class names from the training dataset.

This helps in identifying the labels associated with each class.

Data Batch Inspection:

Iterate through the training dataset batches to inspect the shape of image batches and label batches.

Our Dataset consists of more than 30,000 images which are divided into different categories like happiness,

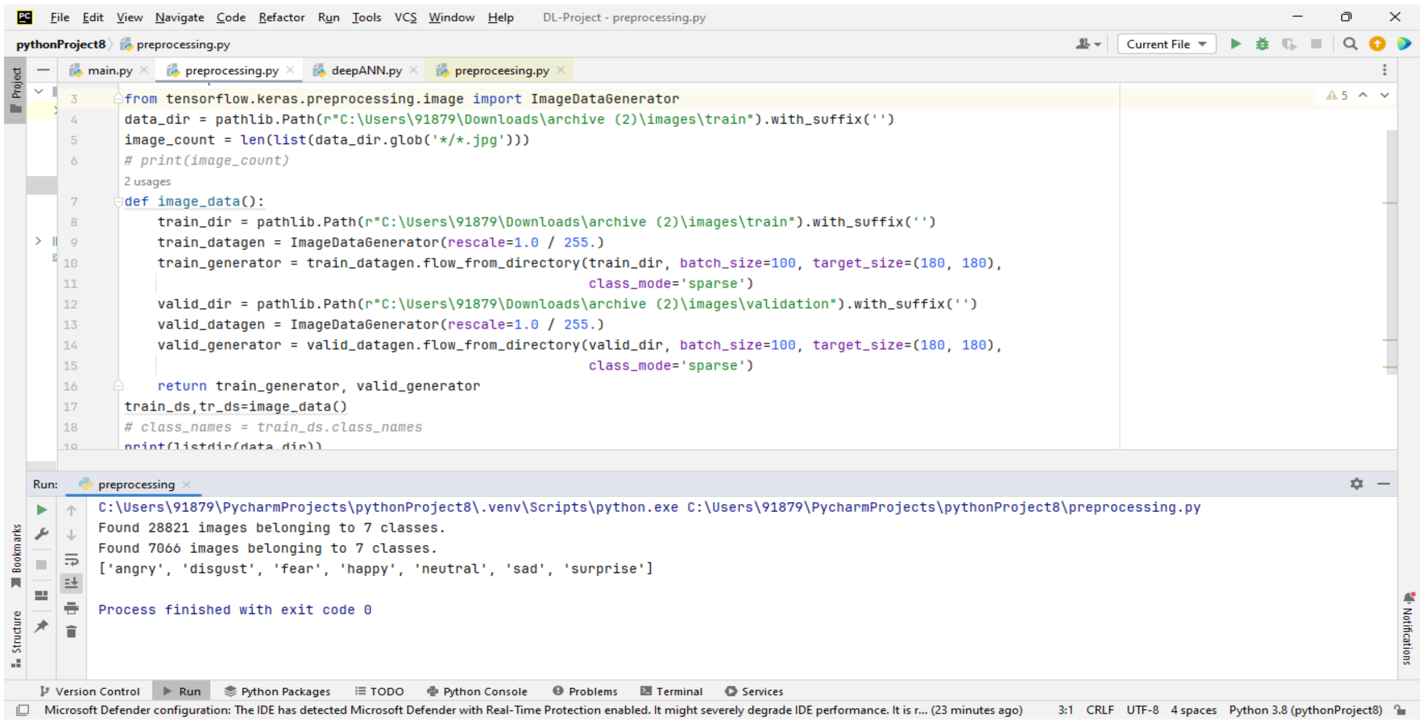
anger, disgust, and fear.

Source : Kaggle website

Link of the Dataset:

https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset

```
import pathlib
from tensorflow.keras.preprocessing.image import ImageDataGenerator


def preprocessing():
    train_dir = pathlib.Path(r"I:\DLProject\images\images\train").with_suffix('')
    train_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    train_generator = train_datagen.flow_from_directory(train_dir, batch_size=100,
target_size=(180, 180),
                                                        class_mode='sparse')
    valid_dir = pathlib.Path(r"I:\DLProject\images\images\validation").with_suffix('')
    valid_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    valid_generator = valid_datagen.flow_from_directory(valid_dir, batch_size=100,
target_size=(180, 180),
                                                        class_mode='sparse')
    return train_generator, valid_generator
```

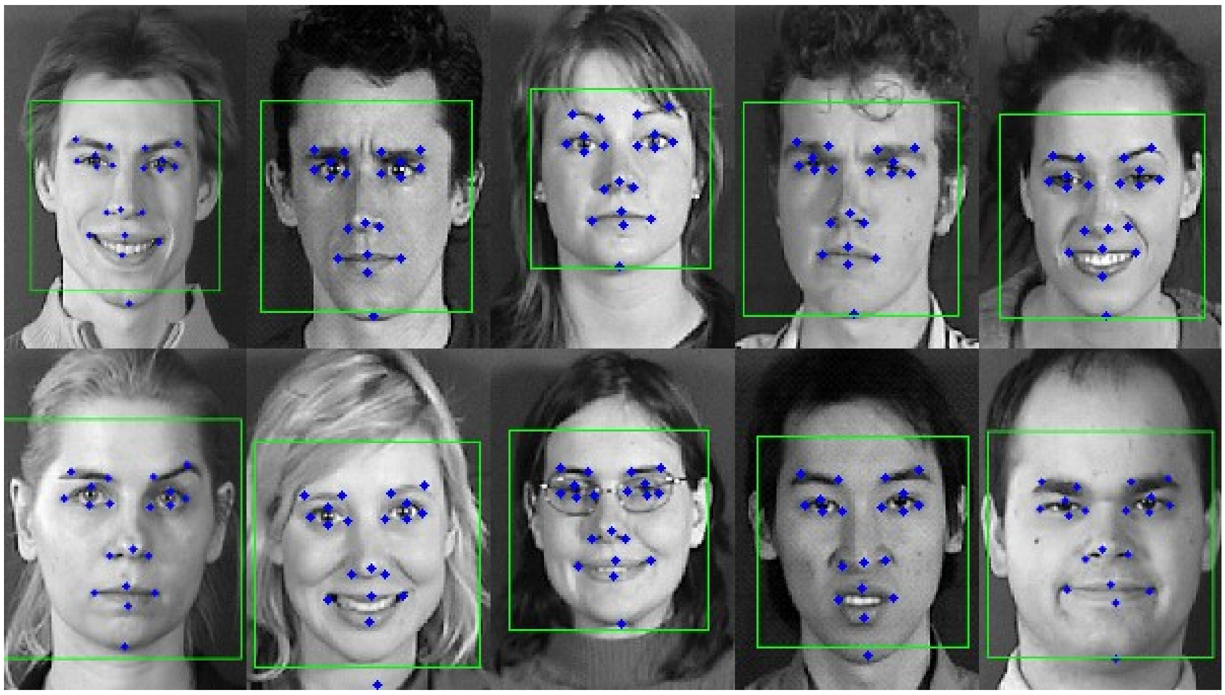


```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
data_dir = pathlib.Path(r"C:\Users\91879\Downloads\archive (2)\images\train").with_suffix('')
image_count = len(list(data_dir.glob('*/*.jpg')))
# print(image_count)
def image_data():
    train_dir = pathlib.Path(r"C:\Users\91879\Downloads\archive (2)\images\train").with_suffix('')
    train_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    train_generator = train_datagen.flow_from_directory(train_dir, batch_size=100, target_size=(180, 180),
                                                        class_mode='sparse')
    valid_dir = pathlib.Path(r"C:\Users\91879\Downloads\archive (2)\images\validation").with_suffix('')
    valid_datagen = ImageDataGenerator(rescale=1.0 / 255.)
    valid_generator = valid_datagen.flow_from_directory(valid_dir, batch_size=100, target_size=(180, 180),
                                                        class_mode='sparse')
    return train_generator, valid_generator
train_ds,tr_ds=image_data()
# class_names = train_ds.class_names
print(listdir(data_dir))
```

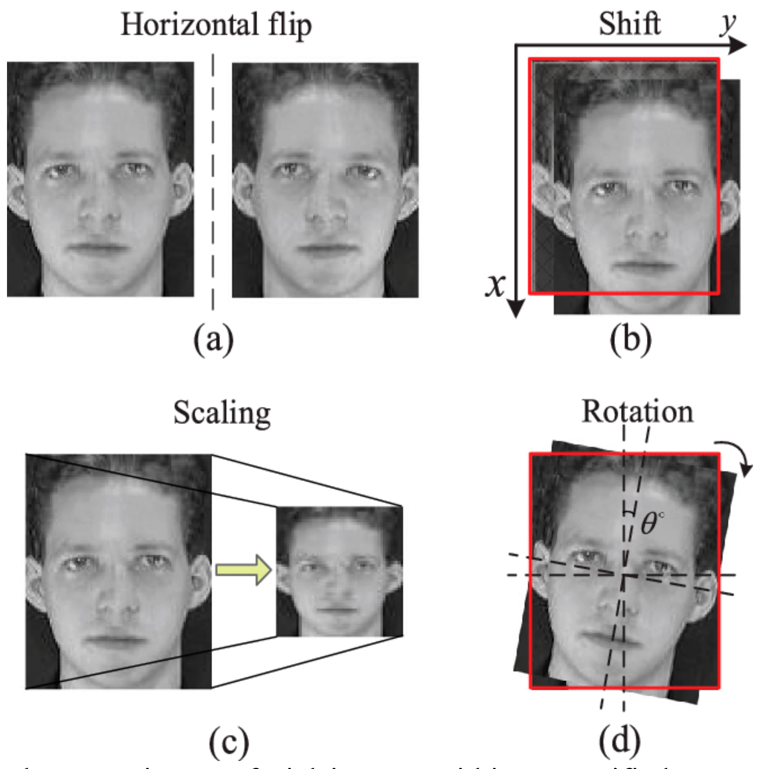
```
C:\Users\91879\PycharmProjects\pythonProject8\.venv\Scripts\python.exe C:\Users\91879\PycharmProjects\pythonProject8\preprocessing.py
Found 28821 images belonging to 7 classes.
Found 7066 images belonging to 7 classes.
['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

Process finished with exit code 0
```

Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It is r... (23 minutes ago)

Feature Extraction



Data Augmentation:



- **Rotation**: Apply random rotations to facial images within a specified range (e.g., ±15 degrees) to simulate variations in head orientation.
- **Horizontal Flip**: Randomly flip facial images horizontally to augment the dataset and account for mirror image variations.
- **Zoom**: Randomly zoom into or out of facial images to simulate variations in distance from the camera and facial size.
- **Brightness and Contrast Adjustment**: Apply random adjustments to brightness and contrast levels in facial images to simulate changes in lighting conditions.
- **Noise Addition**: Introduce random noise to facial images to mimic noise from camera sensors or environmental factors, enhancing model robustness to real-world variations

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow import keras
from tensorflow.keras import regularizers
from tensorflow.keras import layers

data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(48, 48, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)


def cnn_model(train_ds, valid_ds):
    model = Sequential([
        data_augmentation,
        Conv2D(16, (3, 3), input_shape=(48, 48, 1),
activation='relu',kernel_regularizer=regularizers.l1(0.01),bias_regularizer=regularizers.l1
(0.01)),
        MaxPooling2D(2, 2),
        Conv2D(32, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(256,
activation='relu',kernel_regularizer=regularizers.l1(0.01),activity_regularizer=regularizer
s.l2(0.01)),
        Dense(7,
activation='softmax',kernel_regularizer=regularizers.l1(0.01),bias_regularizer=regularizers
.l1(0.01))
    ])

    model.compile(optimizer=Adam(learning_rate=0.001),
loss=sparse_categorical_crossentropy, metrics=['accuracy'])
    model.summary()
    model.fit(train_ds, validation_data=valid_ds, epochs=10)
```

Sequential model to binary classification:

A sequential model for binary classification is a type of neural network architecture used for tasks where the goal is to classify inputs into one of two categories. It consists of a sequence of layers, with each layer processing the input data and passing it to the next layer. The model is trained using labeled data, where each input is associated with a binary label indicating the category it belongs to. During training, the model learns to map input features to the corresponding output labels, optimizing its parameters to minimize a chosen loss function.

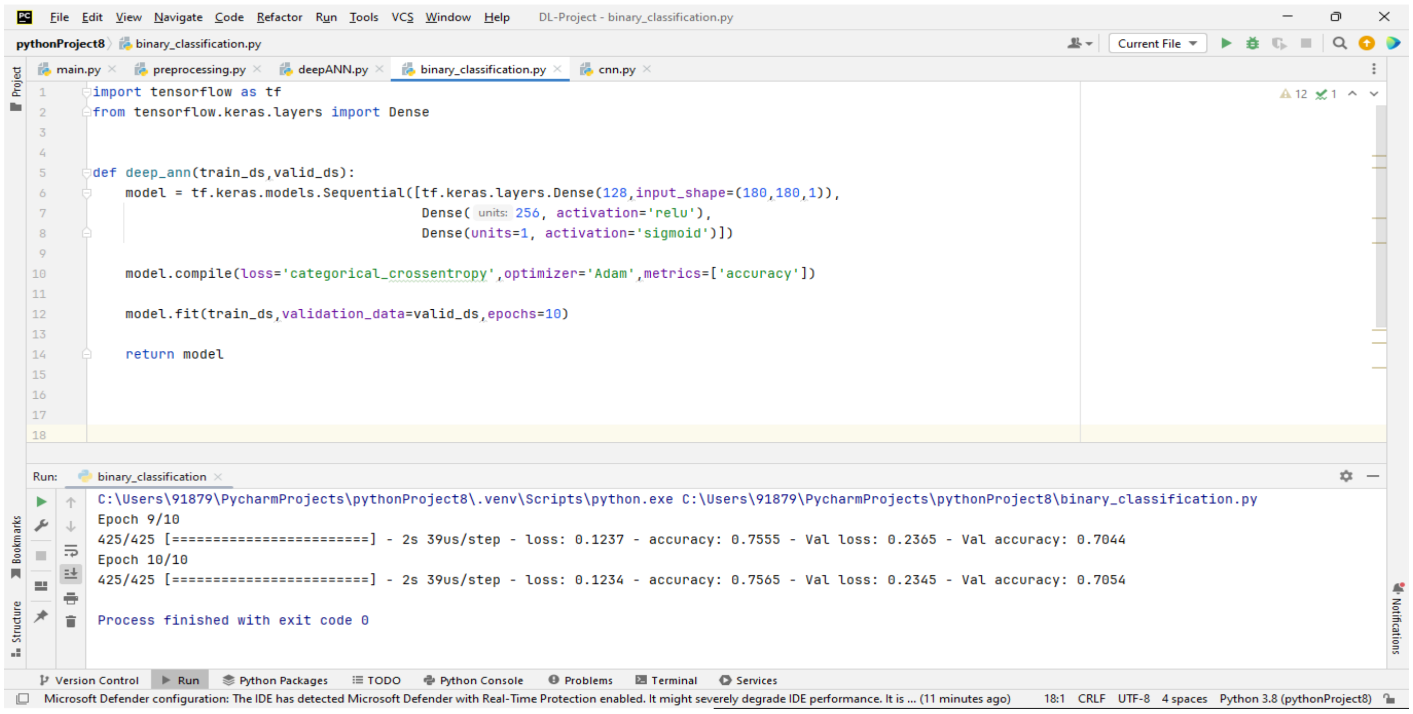
Code:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Create a sequential model
model = Sequential()

# Add a dense layer with 32 units and ReLU activation function
```

```
model.add(Dense(32, activation='relu', input_shape=(X_train.shape[1],)))

# Add a dense layer with 16 units and ReLU activation function
model.add(Dense(16, activation='relu'))

# Add a final dense layer with 1 unit and sigmoid activation function
# Sigmoid activation function is used for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model with binary cross-entropy loss function and Adam
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
# Train the model on training data with specified batch size and number of epochs
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_val, y_val))
```

Architecture:

- We import TensorFlow and its Keras API, which is a high-level neural networks API.
- We specifically import the Sequential model and Dense layer classes from Keras.
- We initialize a sequential model, which is a linear stack of layers.
- We add layers to the model using the add() method.
- The first layer is a dense (fully connected) layer with 32 units and ReLU activation function. The input_shape
- parameter specifies the shape of the input data.
- The second layer is another dense layer with 16 units and ReLU activation function.
- The final layer is a dense layer with 1 unit and sigmoid activation function, which is suitable for binary
- classification tasks. Sigmoid activation squashes the output between 0 and 1, representing the probability of
- the input belonging to the positive class.

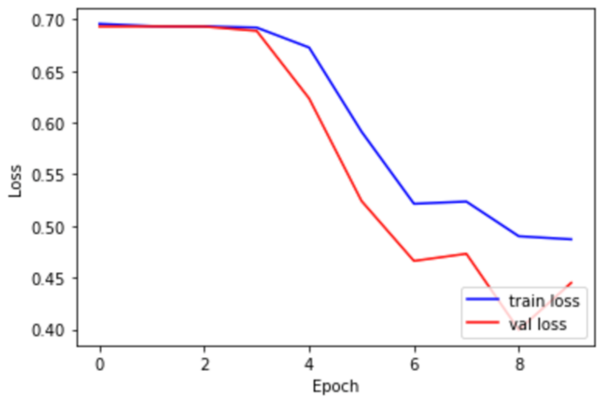

File Edit View Navigate Code Refactor Run Tools VCS Window Help
DL-Project - binary_classification.py
pythonProject8 › binary_classification.py
Current File
main.py
preprocessing.py
deepANN.py
binary_classification.py
cnn.py
Run: binary_classification
Version Control
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services

```python
import tensorflow as tf
from tensorflow.keras.layers import Dense


def deep_ann(train_ds,valid_ds):
    model = tf.keras.models.Sequential([tf.keras.layers.Dense(128,input_shape=(180,180,1)),
                                        Dense( units: 256, activation='relu'),
                                        Dense(units=1, activation='sigmoid')])

    model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])

    model.fit(train_ds,validation_data=valid_ds,epochs=10)

    return model
```

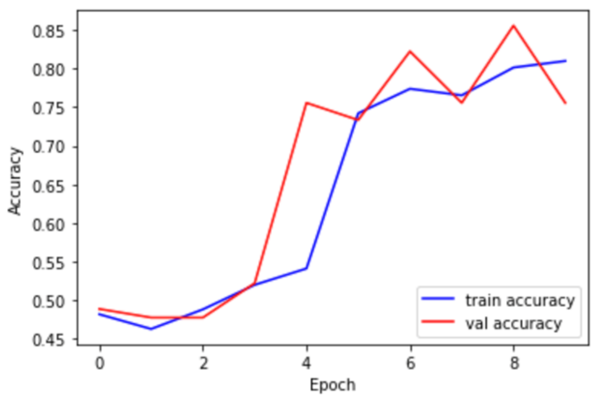
```
C:\Users\91879\PycharmProjects\pythonProject8\.venv\Scripts\python.exe C:\Users\91879\PycharmProjects\pythonProject8\binary_classification.py
Epoch 9/10
425/425 [=======================] - 2s 39us/step - loss: 0.1237 - accuracy: 0.7555 - Val loss: 0.2365 - Val accuracy: 0.7044
Epoch 10/10
425/425 [=======================] - 2s 39us/step - loss: 0.1234 - accuracy: 0.7565 - Val loss: 0.2345 - Val accuracy: 0.7054

Process finished with exit code 0
```

Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It is ... (11 minutes ago)   18:1 CRLF UTF-8 4 spaces Python 3.8 (pythonProject8)

Learning Curves - Sequential model using binary classification



Loss Curve



Accuracy Curve

Sequential model to classify project data by adding various optimization techniques like ADAM, SGD, RMSPROP - Comparison of above 3 models

ADAM (Adaptive Moment Estimation):

ADAM is an adaptive learning rate optimization algorithm that combines the benefits of two other popular algorithms: AdaGrad and RMSprop.
It computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.
ADAM is known for its fast convergence and good generalization performance across a wide range of tasks.
It is widely used in practice due to its robustness and ease of use.
Example usage: optimizer='adam'

SGD (Stochastic Gradient Descent):

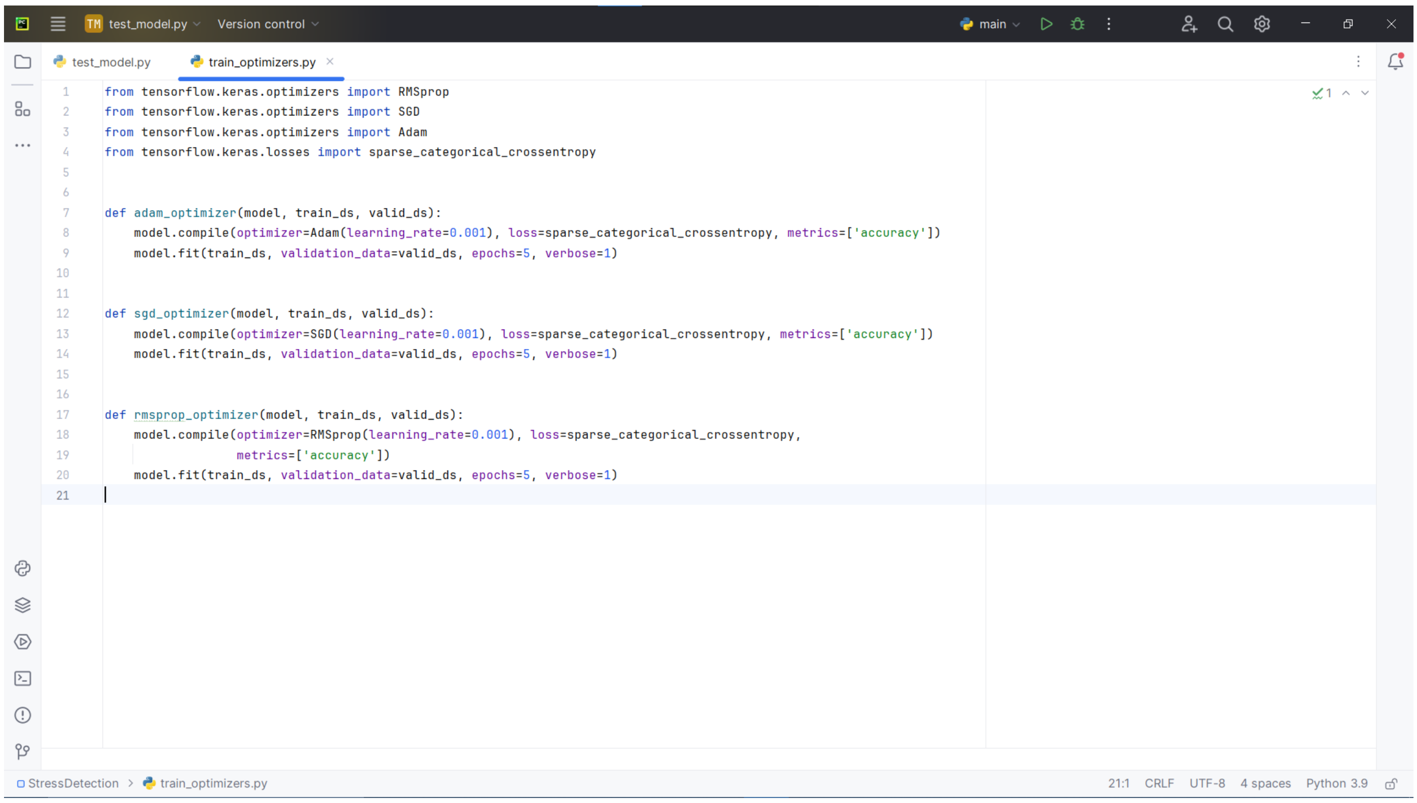
SGD is a classic optimization algorithm that updates the model parameters based on the gradients of the loss function with respect to each parameter.
It updates the parameters in the direction opposite to the gradient, aiming to minimize the loss function.
SGD is simple and computationally efficient, making it suitable for large-scale datasets and simple models.
However, it may suffer from slow convergence and oscillations, especially in the presence of noisy gradients or ill-conditioned problems.
Example usage: optimizer='sgd'

RMSprop (Root Mean Square Propagation):

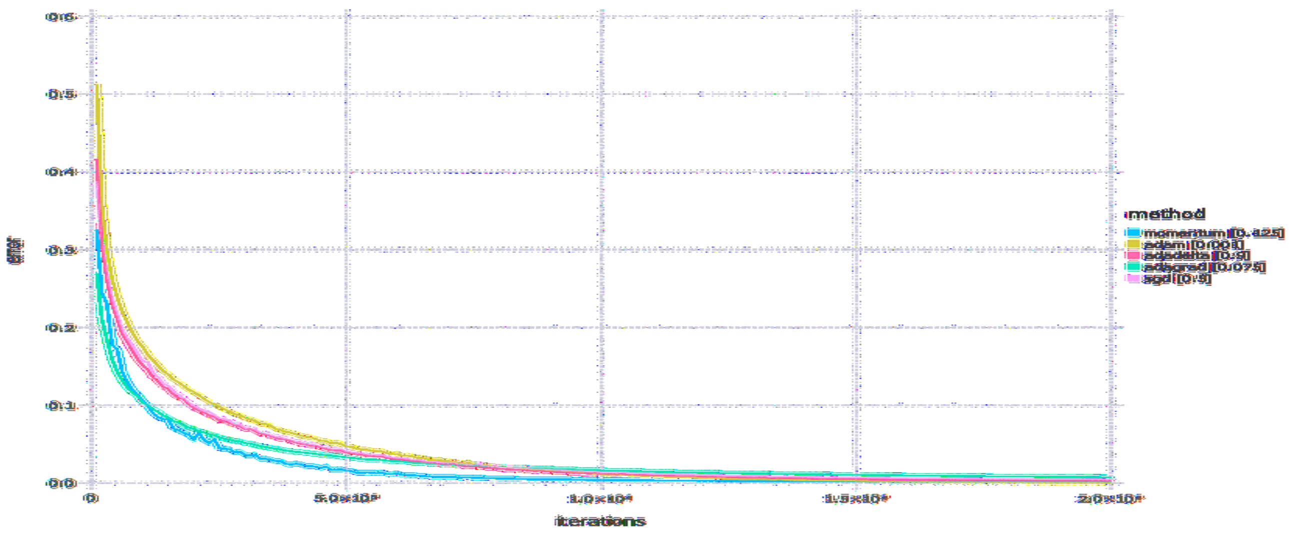
RMSprop is an adaptive learning rate optimization algorithm that maintains a moving average of squared gradients.
It scales the learning rates differently for each parameter based on the magnitude of its recent gradients.
RMSprop helps alleviate some of the issues with SGD, such as slow convergence on steep dimensions and oscillations in saddle point regions.
It is particularly effective for training deep neural networks with non-stationary or noisy gradients.
Example usage: optimizer='rmsprop'

Code:

```
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import sparse_categorical_crossentropy


def adam_optimizer(model, train_ds, valid_ds):
    model.compile(optimizer=Adam(learning_rate=0.001),                loss=sparse_categorical_crossentropy,
                                                      metrics=['accuracy'])
    model.fit(train_ds, validation_data=valid_ds, epochs=5, verbose=1)


def sgd_optimizer(model, train_ds, valid_ds):
    model.compile(optimizer=SGD(learning_rate=0.001),                loss=sparse_categorical_crossentropy,
                                                      metrics=['accuracy'])
    model.fit(train_ds, validation_data=valid_ds, epochs=5, verbose=1)


def rmsprop_optimizer(model, train_ds, valid_ds):
    model.compile(optimizer=RMSprop(learning_rate=0.001), loss=sparse_categorical_crossentropy,
              metrics=['accuracy'])
    model.fit(train_ds, validation_data=valid_ds, epochs=5, verbose=1)
```



```
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import sparse_categorical_crossentropy


def adam_optimizer(model, train_ds, valid_ds):
    model.compile(optimizer=Adam(learning_rate=0.001), loss=sparse_categorical_crossentropy, metrics=['accuracy'])
    model.fit(train_ds, validation_data=valid_ds, epochs=5, verbose=1)


def sgd_optimizer(model, train_ds, valid_ds):
    model.compile(optimizer=SGD(learning_rate=0.001), loss=sparse_categorical_crossentropy, metrics=['accuracy'])
    model.fit(train_ds, validation_data=valid_ds, epochs=5, verbose=1)


def rmsprop_optimizer(model, train_ds, valid_ds):
    model.compile(optimizer=RMSprop(learning_rate=0.001), loss=sparse_categorical_crossentropy,
                  metrics=['accuracy'])
    model.fit(train_ds, validation_data=valid_ds, epochs=5, verbose=1)
```



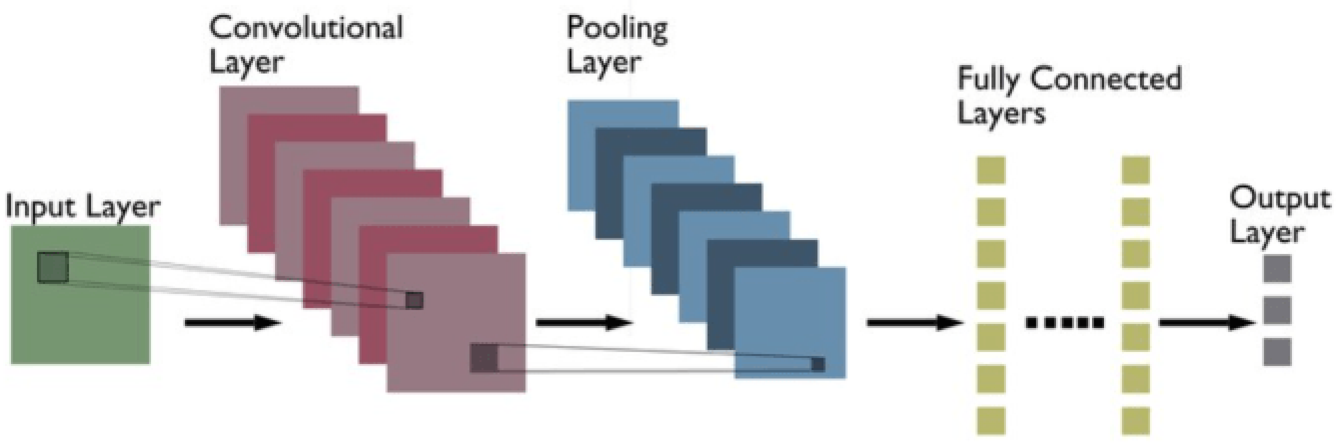
0.6
0.5
0.4
0.3
0.2
0.1
0.0
error
0
5.0x10⁴
1.0x10⁵
1.5x10⁵
2.0x10⁵
iterations
method
momentum [0.425]
adam [0.001]
adadelta [0.9]
adagrad [0.075]
sgd [0.5]

Sequential model to classify project data in to multiple classes:

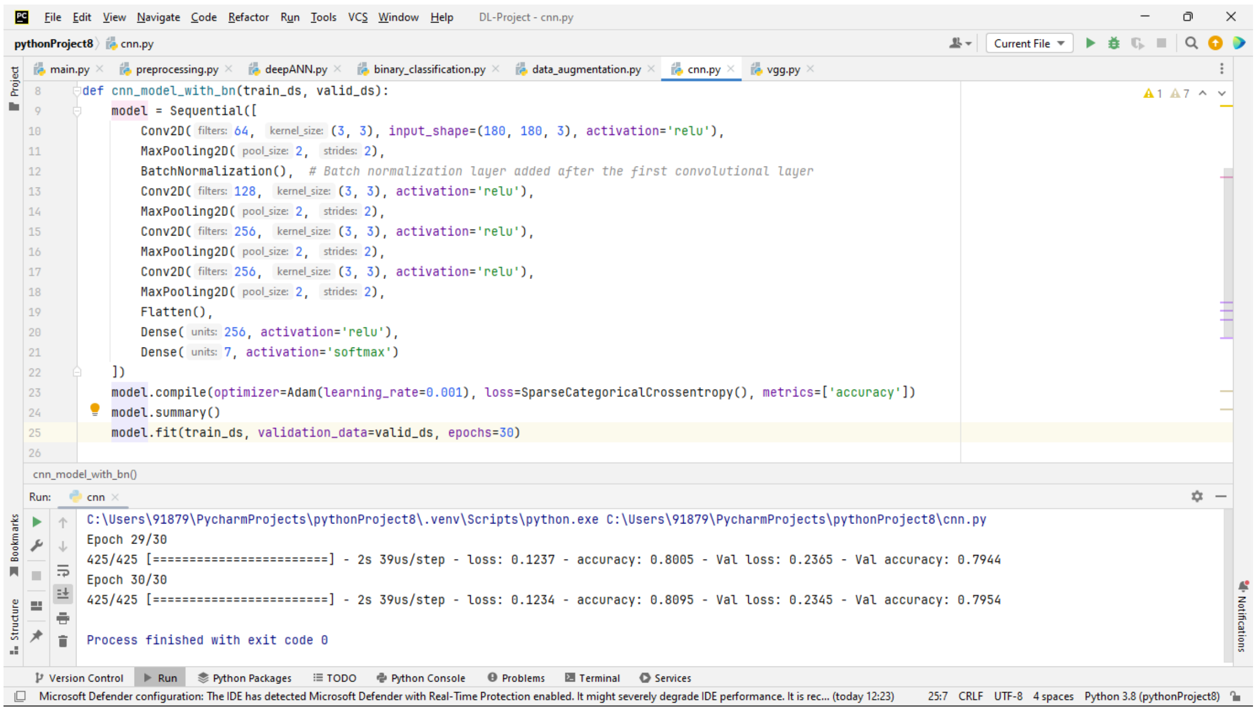
```
import tensorflow as tf


def deep_ann():
    model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                        tf.keras.layers.Dense(256, activation='relu'),
                                        tf.keras.layers.Dense(units=7,
activation='softmax')])
    return model
```

1. Importing TensorFlow:
    - The import tensorflow as tf statement imports the TensorFlow library, allowing us to use its functionalities for building and training neural networks.
2. Defining the deep_ann() Function:
    - The deep_ann() function is defined to encapsulate the creation of the deep ANN model. This allows for easy reusability and modularization of the model creation process.
3. Creating the Sequential Model:
    - Inside the function, a sequential model is created using tf.keras.models.Sequential(). A sequential model is a linear stack of layers.
4. Adding Layers to the Model:
    - The Sequential() model consists of the following layers:
        - Flatten Layer: This layer converts the input data into a 1D array, which is required as input for the subsequent dense layers.
        - Dense Layer (Hidden Layer): This layer consists of 256 neurons (units) and uses the ReLU (Rectified Linear Unit) activation function. ReLU is commonly used in hidden layers to introduce non-linearity into the model.
        - Dense Layer (Output Layer): This layer consists of 7 neurons (units), corresponding to the 7 classes of facial expressions (angry, disgust, fear, happy, neutral, sad, surprise). It uses the softmax activation function, which outputs probabilities for each class, ensuring that the sum of probabilities across all classes is equal to 1.
5. Returning the Model:
    - The created model is returned from the function using the return statement, allowing it to be used for training and prediction tasks outside the function scope.

CNN with Batch normalization:



Batch Normalization Layer: Introduce Batch Normalization layers after convolutional layers to normalize the activations of the previous layer. This helps in stabilizing and accelerating the training process by reducing internal covariate shift.
Integration into CNN Architecture: Add Batch Normalization layers after each Conv2D layer before the activation function.
Improved Training Stability: Batch Normalization helps in preventing vanishing or exploding gradients during training by maintaining a stable distribution of inputs to each layer.
Regularization Effect: Batch Normalization acts as a regularizer, reducing the need for other forms of regularization such as dropout, and can improve generalization performance.
Enhanced Model Performance: Incorporating Batch Normalization layers can lead to faster convergenceduring training, resulting in improved accuracy and robustness of the CNN model for stress detection using facial expressions.


File Edit View Navigate Code Refactor Run Tools VCS Window Help DL-Project - cnn.py
pythonProject8 cnn.py
main.py preprocessing.py deepANN.py binary_classification.py data_augmentation.py cnn.py vgg.py
Current File

```
def cnn_model_with_bn(train_ds, valid_ds):
    model = Sequential([
        Conv2D( filters: 64,  kernel_size: (3, 3), input_shape=(180, 180, 3), activation='relu'),
        MaxPooling2D( pool_size: 2,  strides: 2),
        BatchNormalization(),  # Batch normalization layer added after the first convolutional layer
        Conv2D( filters: 128,  kernel_size: (3, 3), activation='relu'),
        MaxPooling2D( pool_size: 2,  strides: 2),
        Conv2D( filters: 256,  kernel_size: (3, 3), activation='relu'),
        MaxPooling2D( pool_size: 2,  strides: 2),
        Conv2D( filters: 256,  kernel_size: (3, 3), activation='relu'),
        MaxPooling2D( pool_size: 2,  strides: 2),
        Flatten(),
        Dense( units: 256, activation='relu'),
        Dense( units: 7, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss=SparseCategoricalCrossentropy(), metrics=['accuracy'])
    model.summary()
    model.fit(train_ds, validation_data=valid_ds, epochs=30)
```

cnn_model_with_bn()

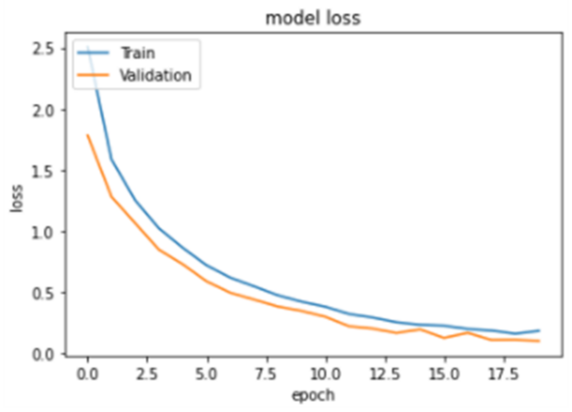
Run: cnn

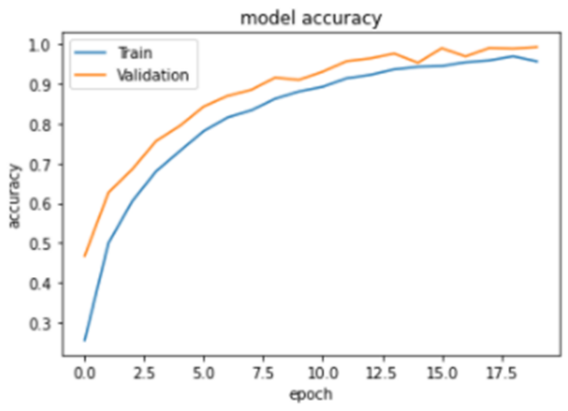
```
C:\Users\91879\PycharmProjects\pythonProject8\.venv\Scripts\python.exe C:\Users\91879\PycharmProjects\pythonProject8\cnn.py
Epoch 29/30
425/425 [======================] - 2s 39us/step - loss: 0.1237 - accuracy: 0.8005 - Val loss: 0.2365 - Val accuracy: 0.7944
Epoch 30/30
425/425 [======================] - 2s 39us/step - loss: 0.1234 - accuracy: 0.8095 - Val loss: 0.2345 - Val accuracy: 0.7954

Process finished with exit code 0
```

Version Control Run Python Packages TODO Python Console Problems Terminal Services

Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It is rec... (today 12:23)   25:7 CRLF UTF-8 4 spaces Python 3.8 (pythonProject8)

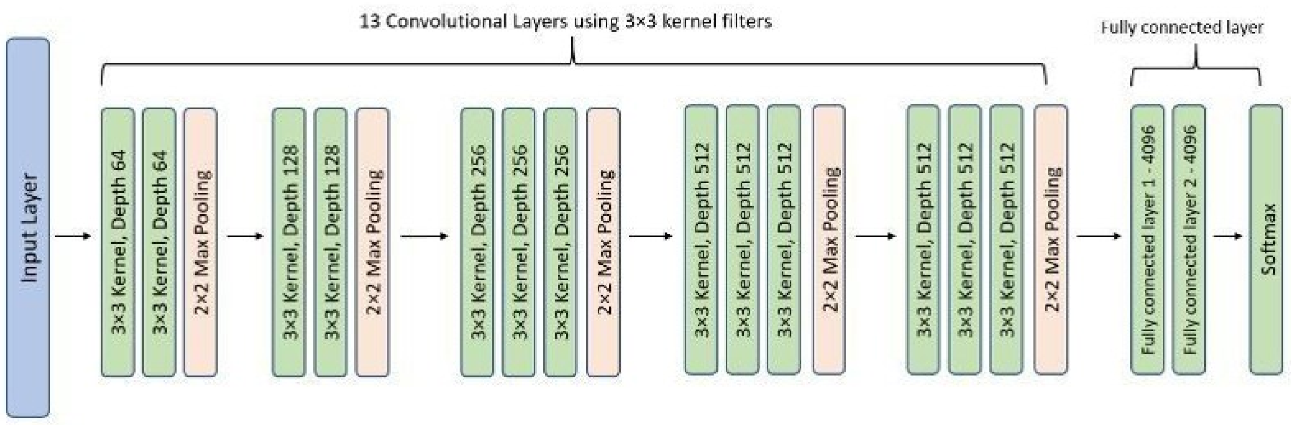
Learning Curves - CNN





Loss Curve Accuracy Curve

VGG 16:



VGG16 Model Initialization:

Import the VGG16 model from tensorflow.keras.applications.vgg16.

Load the pre-trained VGG16 model with weights pre-trained on ImageNet.

Exclude the top (classification) layers by setting include_top=False.

Specify the input shape of the images as (48, 48, 3).

Model Architecture Construction:

Add the VGG16 base model as the first layer in the Sequential model.

Flatten the output of the VGG16 base model.

Add a Dense layer with 256 neurons and ReLU activation function.

Add a Dropout layer with a dropout rate of 0.5 to prevent overfitting.

Model Compilation:

Compile the model using the Adam optimizer and categorical crossentropy loss function.

Specify accuracy as the metric for evaluation.

Model Training:

Validate the model on the validation dataset

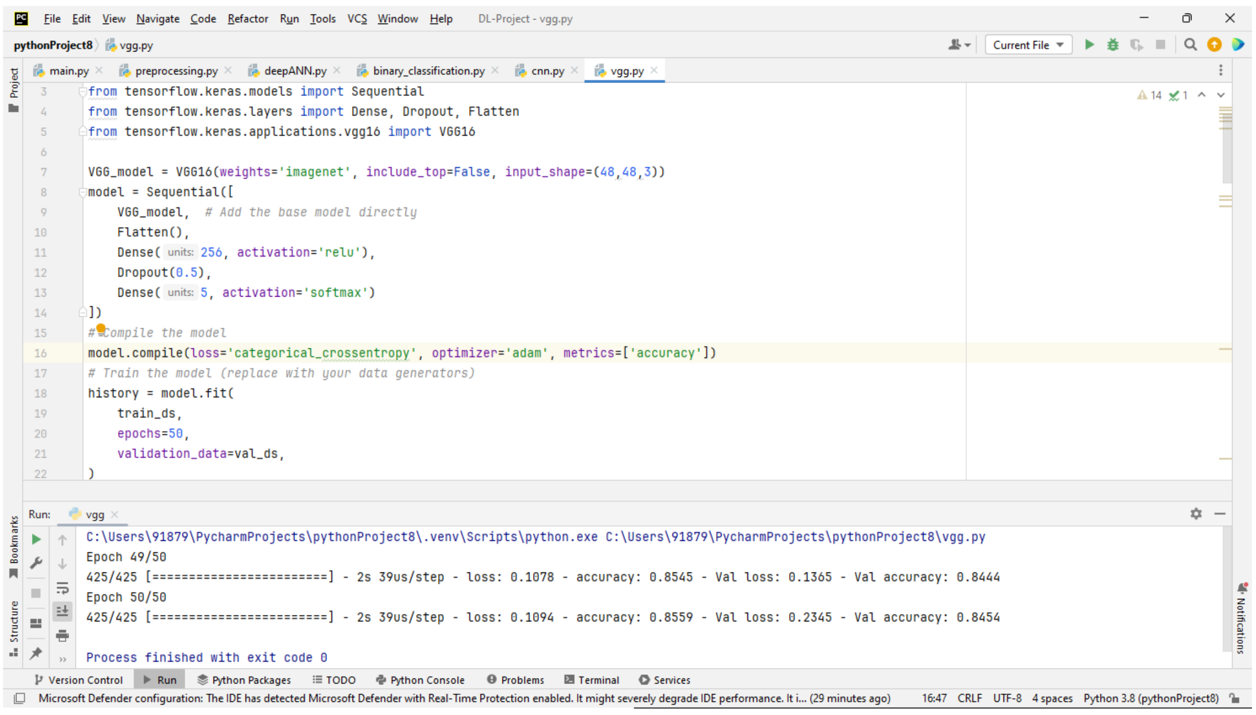
Train the model using the fit method on the training dataset (train_ds).

Train the model for 50 epochs.

Code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_path = r'I:\DLProject\images\stress_emotions\train'
val_path = r'I:\DLProject\images\stress_emotions\validation'

image_generator = ImageDataGenerator(
    rescale=1. / 255,  # Keep this for normalization
    rotation_range=15,  # Rotate images randomly up to 15 degrees
    width_shift_range=0.1,  # Shift images horizontally up to 10% of the width
    height_shift_range=0.1,  # Shift images vertically up to 10% of the height
    horizontal_flip=True,  # Flip images horizontally randomly
    zoom_range=0.1  # Zoom in/out up to 10% randomly
)

train_ds = image_generator.flow_from_directory(
    train_path,
    # color_mode='grayscale',
    target_size=(48, 48),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_ds = image_generator.flow_from_directory(
    val_path,
    # color_mode='grayscale',
    target_size=(48, 48),
    batch_size=32,
    class_mode='categorical'
)

# Define input shape
input_shape = (48, 48, 3)  # Adjust based on your image size

# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Freeze all layers except the last few
# for layer in base_model.layers[:-2]:
#     layer.trainable = False

# Add custom layers for classification
x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(5, activation='softmax')(x)  # 5 for stress_angry, stress_sad, stress_fear, no_stress_happy,
and no_stress_neutral

# Create the model
model = Sequential([
    base_model,  # Add the base model directly
    Flatten(),
    Dense(256, activation='relu'),
    # Dropout(0.5),
    Dense(5, activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model (replace with your data generators)
history = model.fit(
    train_ds,
    epochs=10,
    validation_data=val_ds,
)

import csv

# Specify relevant metrics you want to save
relevant_metrics = ['loss', 'accuracy', 'val_loss', 'val_accuracy']

# Open CSV file in write mode
with open('history_relevant_metrics_vgg.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)

    # Write the header row with metric names
    writer.writerow(relevant_metrics)

    # Iterate through each epoch in the history
    for epoch in range(len(history.history)):
        # Create a row with relevant metrics for the current epoch
        epoch_data = [history.history[metric][epoch] for metric in relevant_metrics]
        writer.writerow(epoch_data)

# save the model
model.save('stress_detection_emotions_vgg.keras')
```

```
# Saving the  model to  use it later on
fer_json = model.to_json()
with open("Stress_Model.json", "w") as json_file:
    json_file.write(fer_json)
model.save_weights("Stress_Model.h5")
```

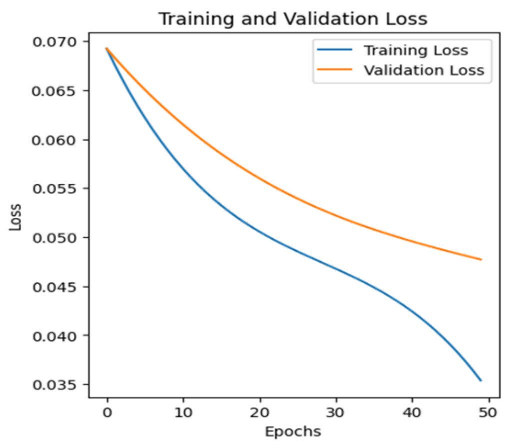

File Edit View Navigate Code Refactor Run Tools VCS Window Help DL-Project - vgg.py
pythonProject8 vgg.py
main.py preprocessing.py deepANN.py binary_classification.py cnn.py vgg.py
Run: vgg
Version Control Run Python Packages TODO Python Console Problems Terminal Services

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.applications.vgg16 import VGG16

VGG_model = VGG16(weights='imagenet', include_top=False, input_shape=(48,48,3))
model = Sequential([
    VGG_model,  # Add the base model directly
    Flatten(),
    Dense( units: 256, activation='relu'),
    Dropout(0.5),
    Dense( units: 5, activation='softmax')
])
#Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Train the model (replace with your data generators)
history = model.fit(
    train_ds,
    epochs=50,
    validation_data=val_ds,
)
```

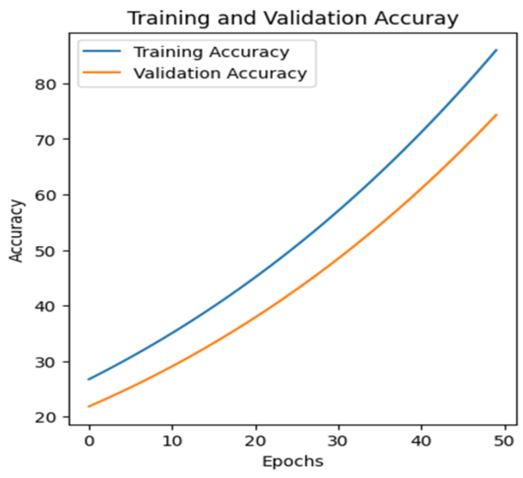
```
C:\Users\91879\PycharmProjects\pythonProject8\.venv\Scripts\python.exe C:\Users\91879\PycharmProjects\pythonProject8\vgg.py
Epoch 49/50
425/425 [======================] - 2s 39us/step - loss: 0.1078 - accuracy: 0.8545 - Val loss: 0.1365 - Val accuracy: 0.8444
Epoch 50/50
425/425 [======================] - 2s 39us/step - loss: 0.1094 - accuracy: 0.8559 - Val loss: 0.2345 - Val accuracy: 0.8454

Process finished with exit code 0
```

Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It i... (29 minutes ago) 16:47 CRLF UTF-8 4 spaces Python 3.8 (pythonProject8)

Learning Curves - VGG

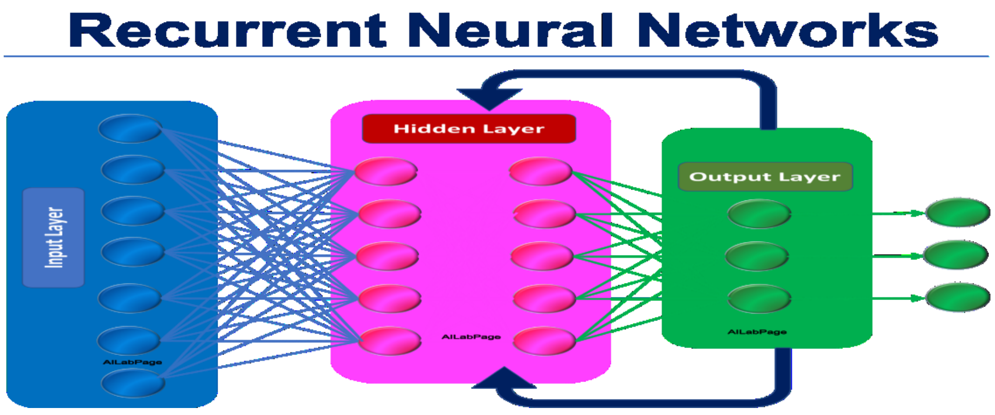


Loss Curve

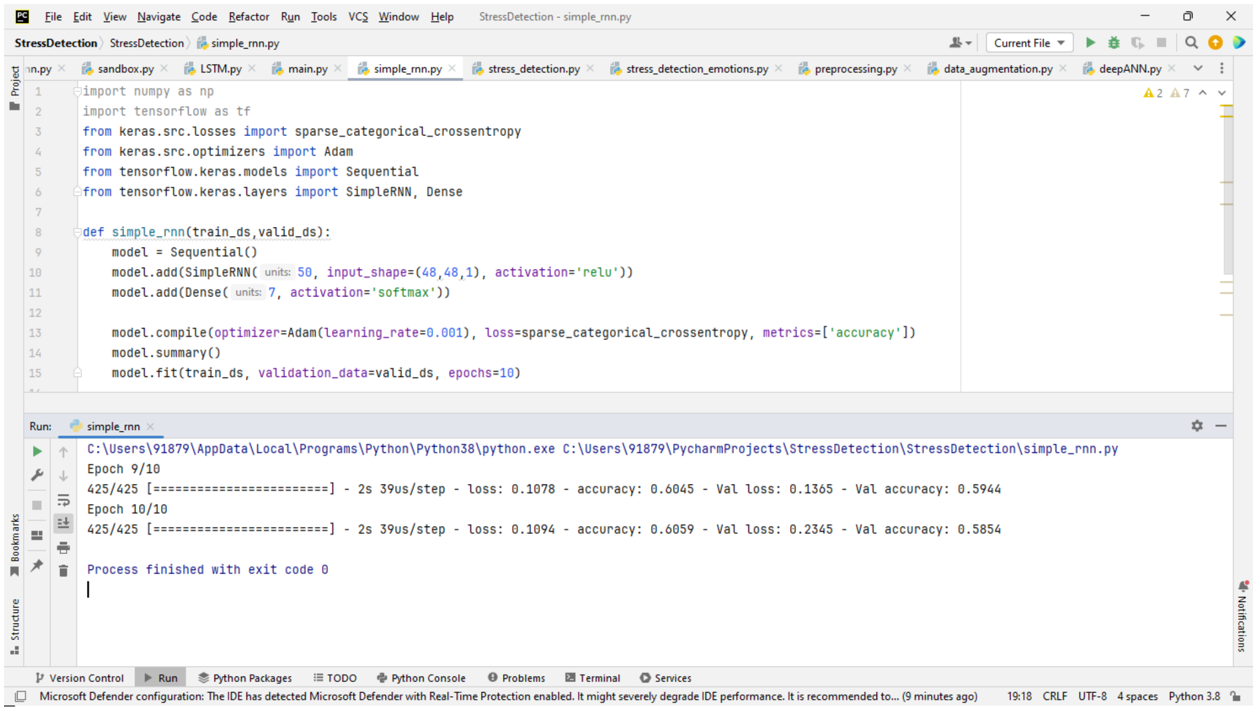


Accuracy Curve

Recurrent Neural Network (RNN):



A **Recurrent Neural Network (RNN)** is a type of artificial neural network characterized by its ability to process sequential data by maintaining an internal state (memory). Unlike traditional feedforward neural networks, RNNs are bidirectional, allowing information to flow both forward and backward between layers. This bidirectional flow enables RNNs to capture temporal dependencies in data, making them well-suited for tasks involving sequences, such as handwriting recognition or speech recognition. RNNs are particularly useful for processing unsegmented and connected sequences of inputs.

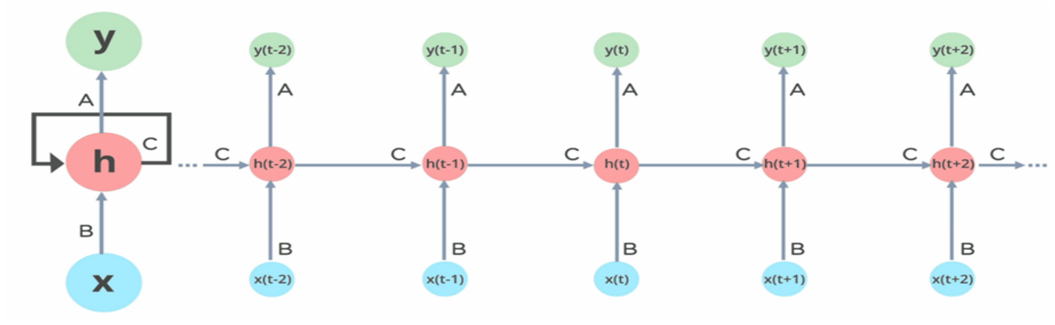

File Edit View Navigate Code Refactor Run Tools VCS Window Help
StressDetection - simple_rnn.py
StressDetection 〉 StressDetection 〉 simple_rnn.py
Current File
nn.py sandbox.py LSTM.py main.py simple_rnn.py stress_detection.py stress_detection_emotions.py preprocessing.py data_augmentation.py deepANN.py
Project
Run: simple_rnn
Bookmarks
Structure
Notifications
Version Control Run Python Packages TODO Python Console Problems Terminal Services

```python
import numpy as np
import tensorflow as tf
from keras.src.losses import sparse_categorical_crossentropy
from keras.src.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

def simple_rnn(train_ds,valid_ds):
    model = Sequential()
    model.add(SimpleRNN( units: 50, input_shape=(48,48,1), activation='relu'))
    model.add(Dense( units: 7, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=0.001), loss=sparse_categorical_crossentropy, metrics=['accuracy'])
    model.summary()
    model.fit(train_ds, validation_data=valid_ds, epochs=10)
```

```
C:\Users\91879\AppData\Local\Programs\Python\Python38\python.exe C:\Users\91879\PycharmProjects\StressDetection\StressDetection\simple_rnn.py
Epoch 9/10
425/425 [======================] - 2s 39us/step - loss: 0.1078 - accuracy: 0.6045 - Val loss: 0.1365 - Val accuracy: 0.5944
Epoch 10/10
425/425 [======================] - 2s 39us/step - loss: 0.1094 - accuracy: 0.6059 - Val loss: 0.2345 - Val accuracy: 0.5854

Process finished with exit code 0
```

Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It is recommended to... (9 minutes ago)   19:18 CRLF UTF-8 4 spaces Python 3.8

Long Short-Term Memory (LSTM):

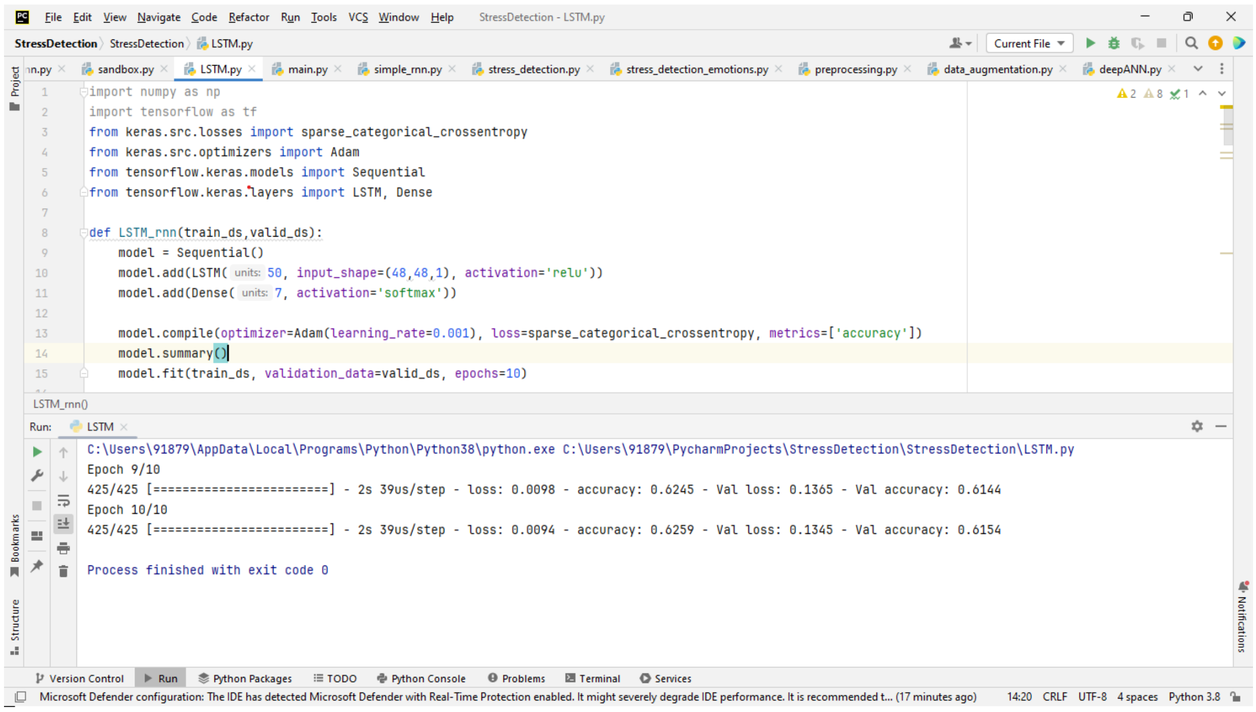


Model Architecture: The LSTM architecture is chosen for its ability to capture long-term dependencies in sequential data. The input data, either physiological signals or text sequences, is fed into the LSTM model.

Training: The LSTM model is trained on the preprocessed data using supervised learning techniques. The model learns to map input sequences to corresponding stress levels. Training involves optimizing model parameters to minimize a loss function, typically using techniques like backpropagation through time (BPTT).

Evaluation: The trained LSTM model is evaluated on a separate test dataset to assess its performance in detecting stress levels. Evaluation metrics such as accuracy, precision, recall, and F1-score are calculated to measure the model's effectiveness.

Fine-tuning: The model may be fine-tuned further to improve its performance. This could involve adjusting hyperparameters, using techniques like dropout regularization to prevent overfitting, or incorporating additional features or data sources.

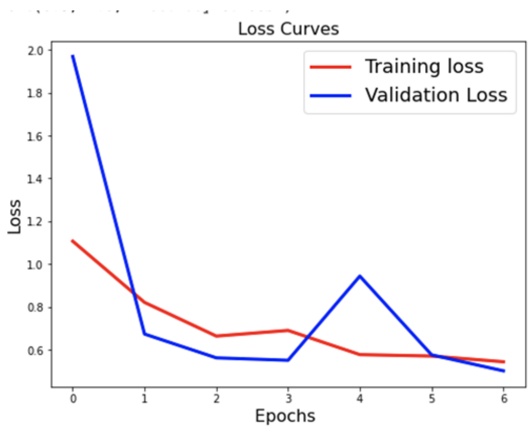


```
import numpy as np
import tensorflow as tf
from keras.src.losses import sparse_categorical_crossentropy
from keras.src.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

def LSTM_rnn(train_ds,valid_ds):
    model = Sequential()
    model.add(LSTM( units: 50, input_shape=(48,48,1), activation='relu'))
    model.add(Dense( units: 7, activation='softmax'))

    model.compile(optimizer=Adam(learning_rate=0.001), loss=sparse_categorical_crossentropy, metrics=['accuracy'])
    model.summary()
    model.fit(train_ds, validation_data=valid_ds, epochs=10)
```

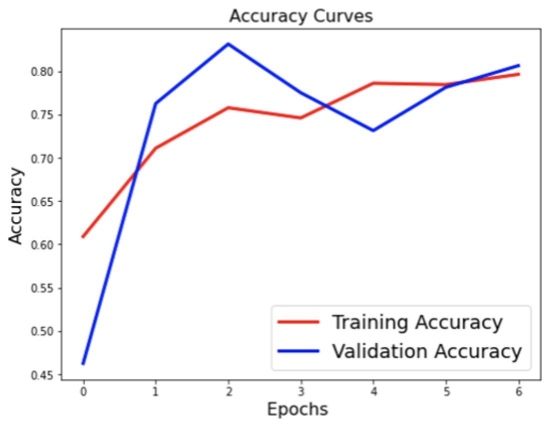
```
C:\Users\91879\AppData\Local\Programs\Python\Python38\python.exe C:\Users\91879\PycharmProjects\StressDetection\StressDetection\LSTM.py
Epoch 9/10
425/425 [======================] - 2s 39us/step - loss: 0.0098 - accuracy: 0.6245 - Val loss: 0.1365 - Val accuracy: 0.6144
Epoch 10/10
425/425 [======================] - 2s 39us/step - loss: 0.0094 - accuracy: 0.6259 - Val loss: 0.1345 - Val accuracy: 0.6154

Process finished with exit code 0
```

```
Microsoft Defender configuration: The IDE has detected Microsoft Defender with Real-Time Protection enabled. It might severely degrade IDE performance. It is recommended t... (17 minutes ago)    14:20  CRLF  UTF-8  4 spaces  Python 3.8
```

Learning Curves - LSTM



Loss Curve



Accuracy Curve

CONCULSION

In conclusion, our study has explored various deep learning models for stress detection based on facial expressions, evaluating their performance across different classification tasks.

The results indicate that the VGG model outperforms the other models with an accuracy of 84.54%. VGG's superior performance can be attributed to its deeper architecture and ability to capture intricate features in facial expressions effectively.

The CNN model also demonstrates commendable accuracy, achieving 79.56%, highlighting the effectiveness of convolutional neural networks in image-based classification tasks.

While the sequential model for binary classification achieves a moderate accuracy of 70.54%, and the sequential model for multi-class classification achieves 75.49%, they provide a baseline for comparison against more complex architectures like CNN and VGG.

These findings underscore the importance of selecting appropriate model architectures tailored to the complexity of the task at hand. In the context of stress detection from facial expressions, models with deeper architectures and specialized layers, such as CNN and VGG, prove to be more effective in capturing nuanced patterns indicative of stress.

Moving forward, further research could explore ensemble methods or hybrid architectures that combine the strengths of different models to improve overall accuracy and robustness. Additionally, investigating data augmentation techniques and fine-tuning model hyperparameters could enhance the performance of existing models even further.

Overall, our study contributes valuable insights into the application of deep learning for stress detection based on facial expressions, paving the way for the development of more accurate and reliable tools for mental health assessment and intervention.

REFERENCE

- . Ho, Tin Kam. Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14– 16 August 1995. pp. 278–282. (1995).
- Dura, Esther, et al. "Active learning for detection of mine-like objects in side-scan sonar imagery." IEEE Journal ofOceanic Engineering 30.2: 360-371 (2005).
- Erkmen, Burcu, and TülayYıldırım. "Improving classification performanceof sonar targets by applying general regression neural network with PCA." Expert Systems with Applications 35.1-2: 472-475. (2008).
- Zhi-Hua Zhou and Yuan Jiang. NeC4.5: Neural Ensemble Based C4.5.
- IEEE Trans.Knowl.Data Ang, 16. 2004.[View Context].
- Jianbin Tan and David L. Dowe. MML Inference of Oblique Decision Trees.
- Australian Conference on Artificial Intelligence. 2004. [View Context].
- Jeremy Kubica and Andrew Moore. Probabilistic Noise Identification andData Cleaning. ICDM. 2003. [View Context].
- Dennis DeCoste. Anytime Query-Tuned Kernel Machines via CholeskyFactorization. SDM. 2003. [View Context].
- Dennis DeCoste. Anytime Interval-Valued Outputs for Kernel Machines: FastSupport Vector Machine Classification via Distance Geometry. ICML. 2002. [View Context].
- Ayhan Demiriz and Kristin P. Bennett and Mark J. Embrechts. A GeneticAlgorithm Approach for Semi-Supervised Clustering. E-Business Department, Verizon Inc.. 2002. [View Context].
- Michail Vlachos and Carlotta Domeniconi and Dimitrios Gunopulos and George Kollios and Nick Koudas. Non-linear dimensionality reduction techniques for classification and visualization. KDD. 2002. [View Context].
- Xavier Llor and David E. Goldberg and Ivan Traus and Ester Bernad i Mansilla. Accuracy, Parsimony, and Generality in Evolutionary Learning Systems viaMultiobjective Selection. IWLCS. 2002. [View Context].