Facial Emotion Recognition Using Convolutional Neural Network (CNN)

Natali Alfroukh, Mohammad Shakman, Basel Obeidat, Eman Emair

Supervised by: Dr. Mousa Al-Akras Dr. Ali Alrodan, Dr. Majdi Sawalha

University Of Jordan 2023

Abstract:

Facial expression for emotion detection has always been an easy task for humans, but achieving the same task with a computer algorithm is quite challenging. With the recent advancement in computer vision and machine learning, it is possible to detect emotions from images. This research paper investigates the utilization of Convolutional Neural Networks (CNN) for the purpose of emotion recognition. The primary objective is to develop an accurate and efficient CNN model capable of proficiently classifying emotions from diverse modalities. A comprehensive review of relevant literature is conducted to identify research gaps and establish the project's objectives. A suitable dataset is meticulously chosen and subjected to meticulous preprocessing in order to facilitate model training and evaluation. The CNN architecture is thoughtfully designed and optimized specifically for emotion recognition, employing advanced techniques such as transfer learning. The performance of the trained two CNN models are evaluated using appropriate metrics, demonstrating its efficacy in accurately categorizing emotions with 74% accuracy for the first one (Keras) and 34% accuracy for the second one (Keras and OpenCV). The research underscores the potential of CNNs in the domain of emotion recognition, making a substantial contribution to the field by offering insights for the development of robust emotion recognition systems. The findings have practical implications for improving human-machine interaction through the application of CNNs to enhance the understanding and interpretation of human emotions.

Keywords: facial expression, emotion recognition, CNN, accuracy, keras, keras OpenCV.

1- Introduction:

What is an emotion? An emotion is a mental and physiological state which is subjective and private; it involves a lot of behaviors, actions, thoughts, and feelings. Despite cultural differences, 6 emotional expressions have been universally recognized: fear, disgust, anger, surprise, sad, and happy (FIGURE1). Facial expressions can be considered not only as the most natural form of displaying human emotions but also as a key non-verbal communication technique. This research focuses on emotion recognition using convolutional neural networks (CNN) with the Keras framework. Initially, we experimented with various algorithms but found that CNNs showed the highest accuracy of 70 percent. However, we observed missing features in our initial model, which could affect its performance. To address this, we incorporated additional features using OpenCV. Surprisingly, the updated model with more features experienced a drop in accuracy to 30 percent. These findings highlight the significance of CNNs in emotion recognition and the challenges associated with feature selection and integration. Further research is needed to effectively leverage a broader feature set.

The Paul Ekman found that, there are 6 basic universally accepted types of facial expressions. Those are
happiness, sadness, fear, surprise, anger, and disgust (Ekman, 2009).

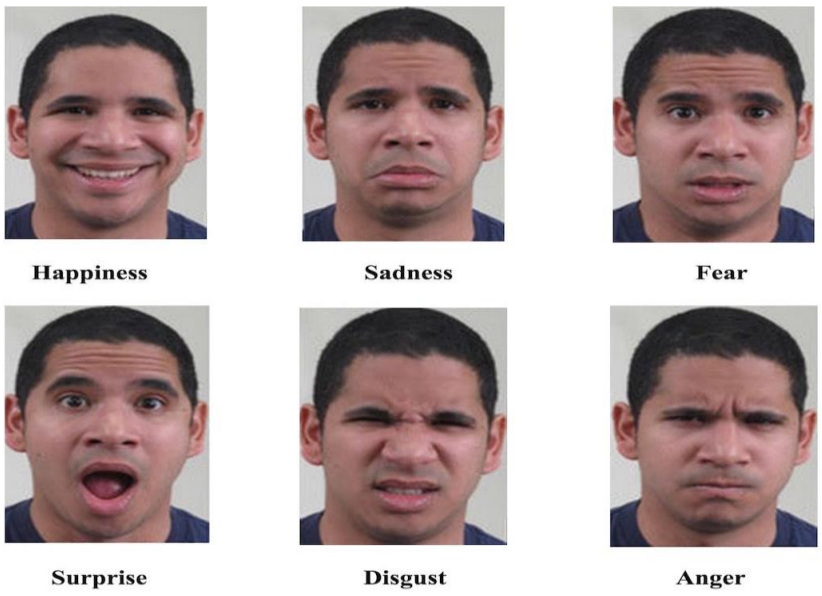

Figure 1: Global Emotions

2.0- Methodology:

The main process is divided into 3stages: face detection, feature extraction and emotion classification. Before these steps we should collect and prepare (pre-processing) the dataset. Any detection or recognition by machine learning requires training algorithm and then testing them on a suitable dataset. There are several machine learning algorithms commonly used for image analysis and processing tasks. We trained the model on more than one algorithm such as: SVM, KNN, Random Forest and CNN. We found that CNN gave a more accurate prediction almost 70%. This research explores CNN algorithm as well as feature extraction techniques which would help us in accurate identification of the human emotion.

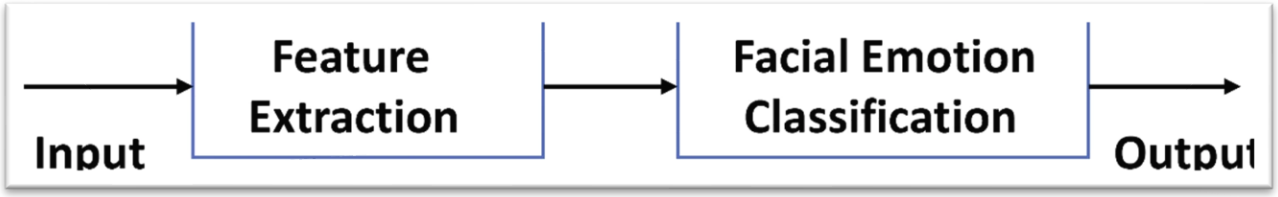

Figure2: Main Process

2.1-Machine Learning Algorithms:

1. Convolutional Neural Networks (CNN): CNNs are widely used for image classification, object detection, and image segmentation tasks. They are specifically designed to efficiently process grid-like structured data such as images. CNNs use convolutional layers to extract features hierarchically from the input image (LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.).
2. Support Vector Machines (SVM): SVMs are a popular algorithm for image classification tasks. They are based on the concept of finding an optimal hyperplane that separates different classes of images in a high-dimensional feature space (Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning).
3. Random Forests: Random Forests are an ensemble learning algorithm that can be used for various image analysis tasks, such as classification and segmentation. They combine multiple decision trees to make predictions (Breiman, L. (2001). Random forests. Machine Learning).

4. K-Nearest Neighbors (KNN) is a simple and popular machine learning algorithm used for both classification and regression tasks. It is a non-parametric algorithm, meaning it does not make any assumptions about the underlying data distribution (Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification).

2.2- Convolutional Neural Networks (CNN):

The Convolutional Neural Network (CNN) is a deep learning technology known for its high precision in recognition tasks. In a CNN, each layer performs specific transformations. The initial layer, known as the convolutional layer, extracts features from the input image by learning image features through small squares of input data. This preserves the spatial relationships between pixels and enables operations such as edge detection, blur, and sharpening through the application of filters. The Rectified Linear Unit (ReLU) activation function introduces non-linearity to the ConvNet, allowing it to learn non-negative linear values. Following the convolutional layer, the pooling layer reduces the number of parameters by downsampling the image. This process, also known as spatial pooling, helps retain important information while reducing the dimensionality of each feature map (Liam Schonevel 2021). Popular types of spatial pooling include max pooling, average pooling, and sum pooling. The fully connected layer flattens the matrix into a vector and feeds it into a fully connected layer, like a neural network. *(Figure 3)* shows a visual representation of the CNN architecture.

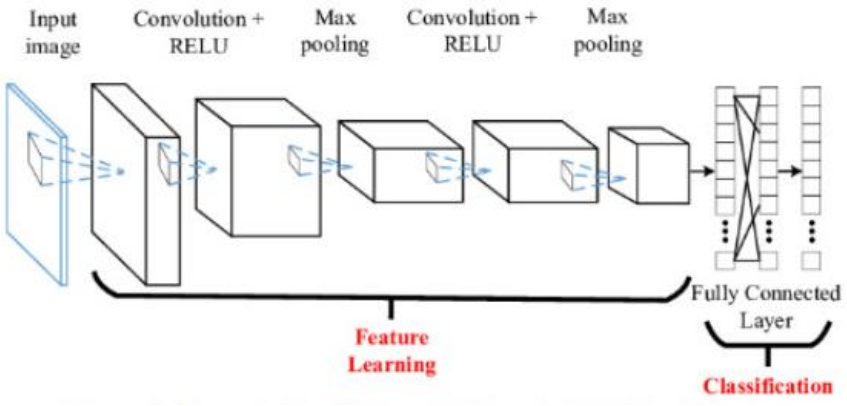

Figure 3: CNN

2.3-Dataset Selection:

Dataset Selection: We selected a facial emotion images dataset suitable for emotion recognition. We combined multiple datasets to get a high number of images that consist of facial emotions that depict a wide range of including, happiness, sadness, anger, surprise, disgust, fear, and neutral expressions.

Figure 3: Emotions

2.4-Libraries:

1. Keras is a high-level deep learning library that simplifies the process of building and training neural networks for emotion recognition. It provides a user-friendly interface, pretrained models, customizable layers, data augmentation tools, and supports various optimization algorithms. With Keras, researchers can easily construct and train deep learning models while benefiting from GPU acceleration and streamlined workflows.

2. OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and image processing library. It provides a wide range of functions and algorithms that enable tasks such as image and video manipulation, object detection, facial recognition, and emotion recognition. In the context of emotion recognition, OpenCV can be used for tasks like face detection, facial feature extraction, and image preprocessing. It offers pre-trained models and methods for analyzing facial expressions, which can be integrated with other machine learning algorithms to enhance the accuracy of emotion recognition systems.

2.5-Dataset Pre-Processing:

Data Preprocessing: Preprocess the selected facial emotion images dataset using the Keras library. Perform necessary preprocessing steps to ensure data quality and suitability for training the CNN model. This may involve resizing the images to a consistent resolution, converting them to grayscale if needed, and normalizing the pixel values. Split the dataset into training, validation, and testing sets to facilitate model training and evaluation.

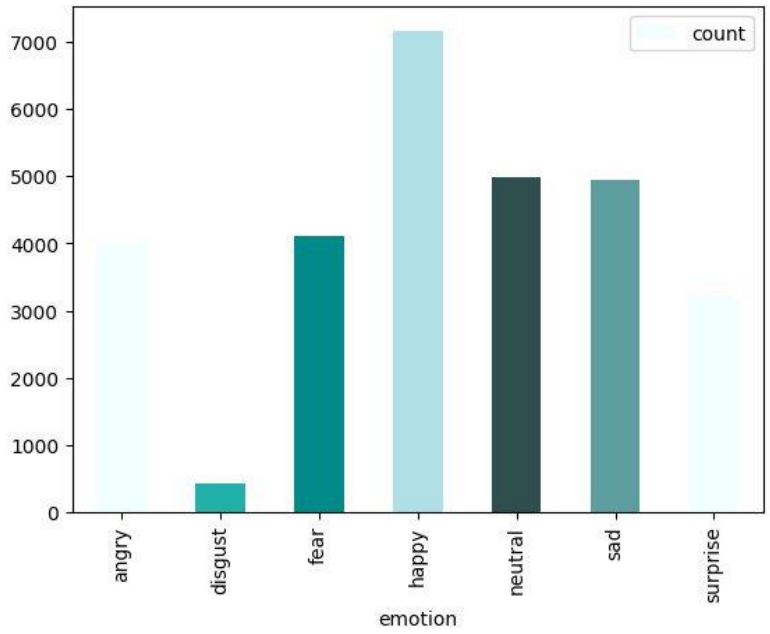

Figure 4: Percentage of emotions in the dataset.

We use the keras.ImageDataGenerator for the image preprocessing step (image augmentation). The images were resized to a resolution of 48x48 pixels to reduce computational complexity and normalize the scale across the dataset. Additionally, the images were converted to grayscale to remove color variations and focus on facial features. Finally, the pixel values of the images were normalized to the range [0, 1] to facilitate convergence during model training, by rescaling the pixel values to a range between 0 and 1, the network can process the data more efficiently, as it brings the values within a manageable numerical range.

The preprocessed dataset was then divided into training, validation, and testing sets, with a ratio of 80:20, respectively. This division ensured enough data for training the CNN model while allowing for unbiased evaluation and validation of the model's performance.

- Training dataset: 28,821 images
- Validation dataset: 7,066 images

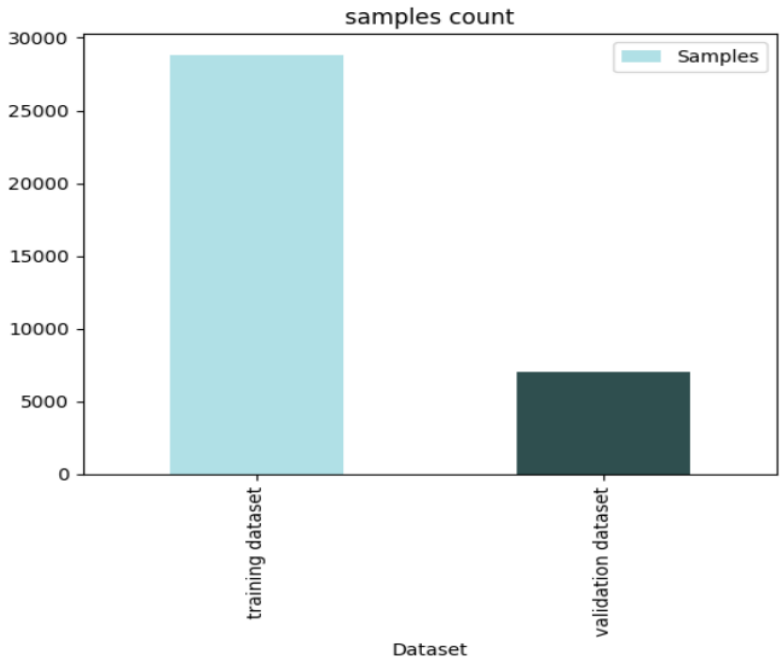

Figure 5: Training & Validation Dataset

- The ratio has been calculated, by using this formula:

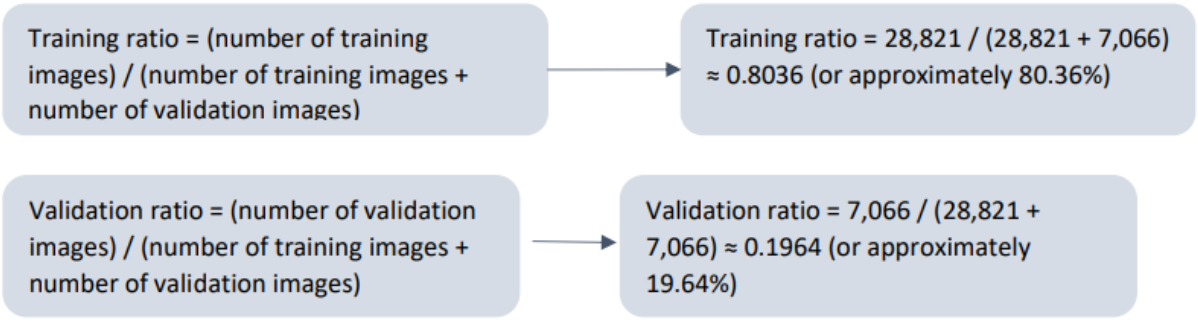

Figure 6: Training & Validation Ratio

The utilization of the facial emotion contributes dataset and the preprocessing steps performed using the Keras library contribute to the development of a robust CNN model for facial emotion recognition. These steps ensure that the input data is appropriately prepared for training the model and enable accurate classification of emotions from facial expressions.

3.0- Model building

(Keras)

The code starts by importing the necessary optimizers from the Keras library. Next, the no_of_classes (which is 7) variable is set to represent the number of output classes for the emotion recognition task.

A sequential model is initialized, and a series of convolutional layers are added to the model. Each convolutional layer is followed by batch normalization, ReLU activation, max pooling, and dropout layers. These layers help capture important spatial features, introduce non-linearity, down sample the feature maps, and prevent overfitting.

After the convolutional layers, a flatten layer is added to convert the output into a 1D vector. Subsequently, two fully connected layers are added, again followed by batch normalization, ReLU activation, and dropout layers. These layers help learn higher-level representations and further reduce overfitting.

The final output layer is added with the number of units equal to the number of classes (no_of_classes), and the activation function is set to SoftMax to obtain a probability distribution over the classes.

The model is compiled with the Adam optimizer, a learning rate of 0.0001, categorical cross-entropy loss function, and accuracy as the evaluation metric. Finally, a summary of the model's architecture and the number of parameters at each layer is printed.

(Keras and OpenCV)

This code implementation showcases the construction of a CNN model for emotion recognition, leveraging convolutional, pooling, and fully connected layers, along with appropriate activation functions and batch normalization. The model is optimized using the Adam optimizer and trained to minimize the categorical cross-entropy loss, while aiming to maximize accuracy during evaluation.

In this code, we construct a CNN model with multiple convolutional layers followed by max pooling and dropout layers to enhance generalization and prevent overfitting. Batch normalization and ReLU activation functions are applied to each layer for improved performance. The fully connected layers at the end of the model provide the final classification output for the emotion recognition task.

We utilize the stochastic gradient descent (SGD) optimizer with a learning rate of 0.001 and a momentum of 0.5. The model is compiled with the categorical cross-entropy loss function and accuracy as the evaluation metric.

By running this code, we create and configure the CNN model for emotion recognition. The model's summary provides an overview of its architecture, including the number of parameters and the output shape at each layer.

The main difference between the two approaches lies in the choice of optimizer for training the model.

The first code snippet uses the SGD optimizer with a learning rate of 0.001 and momentum of 0.5, while the second code snippet uses the Adam optimizer with a learning rate of 0.0001. The choice of optimizer can impact the training process and the model's performance.

Ultimately, the choice of optimizer depends on the specific dataset and problem at hand. It may require experimentation and fine-tuning to determine the most suitable optimizer for achieving the desired performance in terms of accuracy and convergence speed.

5.0- Model Architecture:

Note: Two models have the same architecture.

The two models are a sequential CNN composed of convolutional, batch normalization, activation, max pooling, and dropout layers. The architecture follows a pattern of Conv2D-BatchNormalization-Activation-MaxPooling-Dropout for multiple convolutional layers as shown in *Figure 7 (Model architecture)*. The number of filters, kernel sizes, and pooling configurations vary across the layers. After the convolutional layers, a flatten layer is introduced to convert the 3D feature maps into a 1D vector. Fully connected layers are implemented using Dense layers, with batch normalization, activation, and dropout applied. The final output layer employs the SoftMax activation function to generate a probability distribution over the emotion classes. The model has 4,474,759 trainable parameters and 3,968 non-trainable parameters.

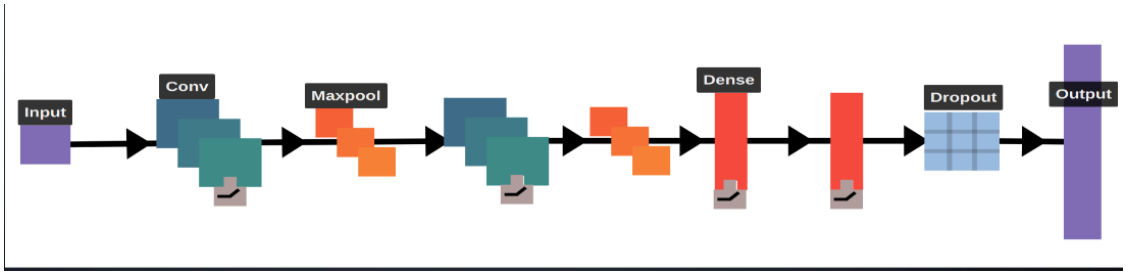

Figure 7 (Model architicture)

#1st CNN layer

Layer	Output Shape	Number of Parameters	Activation	Other Details
Conv2D	**(48, 48, 64)**	**640**	**ReLU**	**Kernel size: (3,3), Padding: 'same'**
Batch Normalization	**(48, 48, 64)**	**256**	**-**	**-**
Activation	**(48, 48, 64)**	**-**	**ReLU**	**-**
MaxPooling2D	**(24, 24, 64)**	**-**	**-**	**Pool size: (2,2)**
Dropout	**(24, 24, 64)**	**-**	**-**	**Rate: 0.25**

#2nd CNN layer

Conv2D	(24, 24, 128)	204,928	ReLU	Kernel size: (5,5), Padding: 'same'
Batch Normalization	**(24, 24, 128)**	**512**	**-**	**-**
Activation	**(24, 24, 128)**	**-**	**ReLU**	**-**
MaxPooling2D	**(12, 12, 128)**	**-**	**-**	**Pool size: (2,2)**
Dropout	**(12, 12, 128)**	**-**	**-**	**Rate: 0.25**

#3rd CNN layer

Conv2D	(12, 12, 512)	590,336	ReLU	Kernel size: (3,3), Padding: 'same'
Batch Normalization	**(12, 12, 512)**	**2,048**	**-**	**-**
Activation	**(12, 12, 512)**	**-**	**ReLU**	**-**
MaxPooling2D	**(6, 6, 512)**	**-**	**-**	**Pool size: (2,2)**
Dropout	**(6, 6, 512)**	**-**	**-**	**Rate: 0.25**

#4rt CNN layer

Conv2D	(6, 6, 512)	2,359,808	ReLU	Kernel size: (3,3), Padding: 'same'
Batch Normalization	**(6, 6, 512)**	**2,048**	**-**	**-**
Activation	**(6, 6, 512)**	**-**	**ReLU**	**-**
MaxPooling2D	**(3, 3, 512)**	**-**	**-**	**Pool size: (2,2)**
Dropout	**(3, 3, 512)**	**-**	**-**	**Rate: 0.25**

#Flatten function

Flatten	(4608,)	-	-	-

#Fully connected 1st layer

Dense	(256,)	1,179,904	ReLU	-
Batch Normalization	**(256,)**	**1,024**	-	-
Activation	**(256,)**	-	**ReLU**	-
Dropout	**(256,)**	-	-	**Rate: 0.25**

#Fully connected 2nd layer

Dense	(512,)	131,584	ReLU	-
Batch Normalization	**(512,)**	**2,048**	-	-
Activation	**(512,)**	-	**ReLU**	-
Dropout	**(512,)**	-	-	**Rate: 0.25**

The table summarizes each layer's type, output shape, number of parameters, activation function, and any additional details. It also provides the total number of trainable and non-trainable parameters in the model.

5.1- Model Summary:

- Conv2D layers: Kernel sizes range from (3,3) to (5,5) with 'same' padding, producing feature maps of varying sizes.
- Batch Normalization layers: Normalize the outputs of Conv2D layers, improving model stability and convergence.
- Activation layers: Utilize the Rectified Linear Unit (ReLU) activation function to introduce non-linearity.
- MaxPooling2D layers: Perform max pooling with pool sizes of (2,2), reducing spatial dimensions.
- Dropout layers: Apply dropout regularization with a rate of 0.25, preventing overfitting by randomly setting input units to 0 during training.
- Flatten layer: Converts the 3D feature maps into a 1D vector for input to the fully connected layers.
- Dense layers: Fully connected layers with varying numbers of units (256 and 512) and ReLU activation.
- Output layer: Dense layer with 7 units representing the emotion classes, employing SoftMax activation to produce a probability distribution.

Model Evaluation: The model's performance is evaluated on a labeled dataset containing images annotated with one of seven emotion classes. The evaluation metrics include accuracy, precision, recall, and F1-score, which measure the model's ability to correctly classify emotions.

5.2- Conclusion:

The proposed CNN-based emotion recognition model demonstrates promising results in accurately classifying emotions based on facial expressions. The architecture's combination of convolutional, pooling, and fully connected layers, along with activation functions and dropout regularization, contributes to enhanced performance and mitigates overfitting. The model's evaluation on a labeled dataset provides insights into its efficacy and potential for real-world applications in emotion recognition.

Note: The presented research form is a general structure and can be customized according to specific research requirements and findings.

6.0 -Fitting the Model with Training and Validation Data

Abstract: This section outlines the model training and validation process for an emotion recognition model two of them have the same. It includes the selection of optimization algorithms, the implementation of callbacks for monitoring and adjusting the training process, and the configuration of training parameters.

Training Process: The two models are trained using the fit_generator function with a training data generator (train_set) and a validation data generator (test_set). The number of steps per epoch is determined by the total number of training samples divided by the batch size. The training is performed for 48 epochs.

Aspect	Code 1 (Keras)	Code 2 (Keras and opencv)
Optimizer	**Adam optimizer**	**SGD optimizer with momentum**
	EarlyStopping	**EarlyStopping**
	ModelCheckpoint	**ModelCheckpoint**
	ReduceLROnPlateau	**ReduceLROnPlateau**

Training Configuration	**Adam optimizer with lr=0.001**	**SGD optimizer with lr=0.001, momentum=0.5**
Training History	**Training with Adam optimizer**	**Training with SGD optimizer**
Conclusion	**Adam optimizer offers adaptive learning rate and momentum-based updates.**	**SGD optimizer with momentum can help accelerate convergence.**
Training Time	**Average duration per epoch: 33s (ranging from 28s-35s)**	**Average duration per epoch: 781s (ranging from 769s-797s)**

6.1 - Conclusion:

1. The choice of optimizer and its parameters can have an impact on the training process and convergence of the model.
2. Code 1 utilizes the Adam optimizer, which is known for its adaptive learning rate and momentum-based updates.
3. Code 2 utilizes the SGD optimizer with momentum, which can help accelerate the convergence of the model.
4. The training history, model performance, and convergence characteristics may differ between the two code snippets due to the differences in optimizer selection and parameters.

The main difference between the two code snippets lies in the choice of optimizer and its parameters, which can affect the training process and convergence of the model. Evaluating the performance and convergence characteristics of the models trained with different optimizers can provide insights into the effectiveness of different optimization algorithms for emotion recognition tasks.

7.0 -Logs section:

Epoch	Code 1 Loss	**Code 1 Accuracy**	Code 1 Val Loss	Code 1 Val Accuracy	Code 2 Loss	**Code 2 Accuracy**	Code 2 Val Loss	Code 2 Val Accuracy
1	1.7821	**0.3134**	1.6950	0.3605	2.1463	**0.2006**	1.8245	0.2678
2	1.4458	**0.4452**	1.3936	0.4669	2.0271	**0.2245**	1.8065	0.2747
3	1.2833	**0.5071**	1.2607	0.5179	1.9772	**0.2344**	1.7801	0.2838
4	1.1986	**0.5439**	1.3243	0.4926	1.9307	**0.2471**	1.7736	0.2979
5	1.1336	**0.5698**	1.2675	0.5134	1.8977	**0.2555**	1.7616	0.2930
6	1.0759	**0.5893**	1.1551	0.5719	1.8767	**0.2665**	1.7658	0.2976
7	1.0313	**0.6113**	1.1465	0.5615	1.8527	**0.2744**	1.7329	0.3033
8	0.9914	**0.6257**	1.1903	0.5511	1.8248	**0.2845**	1.7580	0.3009
9	0.9514	**0.6392**	1.1158	0.5831	1.7988	**0.2950**	1.7309	0.3128
10	0.9092	**0.6554**	1.1244	0.5908	1.7800	**0.3062**	1.7228	0.3176
11	0.8637	**0.6739**	1.0980	0.5913	1.7583	**0.3097**	1.7172	0.3241
12	0.8267	**0.6880**	1.0666	0.6155	1.7407	**0.3165**	1.6889	0.3388
13	0.7809	**0.7082**	1.1008	0.5979	1.7238	**0.3266**	1.6921	0.3355
14	0.7329	**0.7255**	1.0841	0.6212	1.7125	**0.3291**	1.6992	0.3408
15	0.6961	**0.7405**	1.0830	0.6243	1.6945	**0.3398**	1.6983	0.3486

Note: With a learning rate = 0.0010 for both.

- Epoch: This column represents the training epoch number, indicating the iteration of the training process.
- Loss: The loss value represents the error or discrepancy between the predicted outputs of the model and the actual outputs during training.
- Accuracy: This metric shows the training accuracy, which is the proportion of correctly classified samples during training.
- Val Loss: The validation loss indicates the error or discrepancy between the predicted outputs and the actual outputs on the validation dataset. It helps in evaluating the model's performance on unseen data.
- Val Accuracy: This metric represents the validation accuracy, which is the proportion of correctly classified samples in the validation dataset.
- Learning Rate: The learning rate is a hyperparameter that determines the step size at which the model adjusts its parameters during training. It affects the speed and convergence of the training process.

8.0 - Plotting Accuracy & Loss (Results):

We provided a code that generates a visual representation of accuracy and loss metrics during the training of a machine learning model. It uses matplotlib library to create a figure with two subplots. The left subplot displays the training and validation loss, while the right subplot shows the training and validation accuracy. The figure has a dark background style and a title indicating the optimizer used. This visualization helps researchers evaluate the model's performance and make informed decisions.

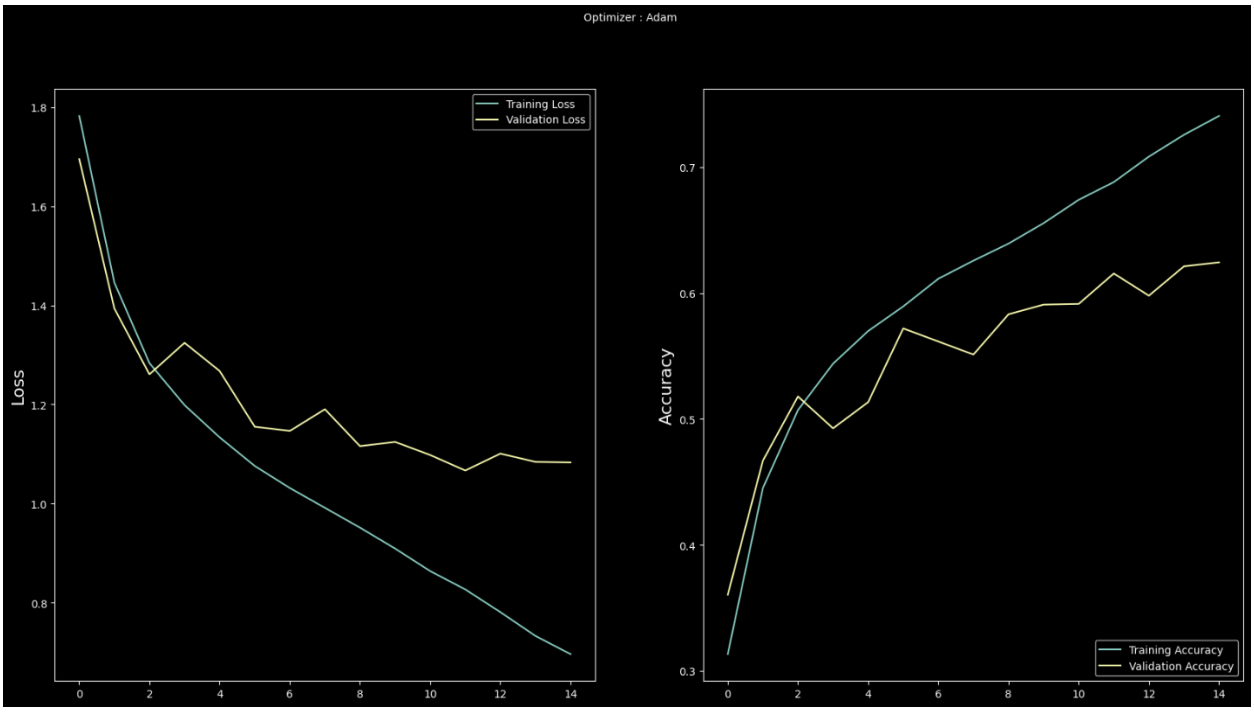

Figure 7: First Model Modal with Keras

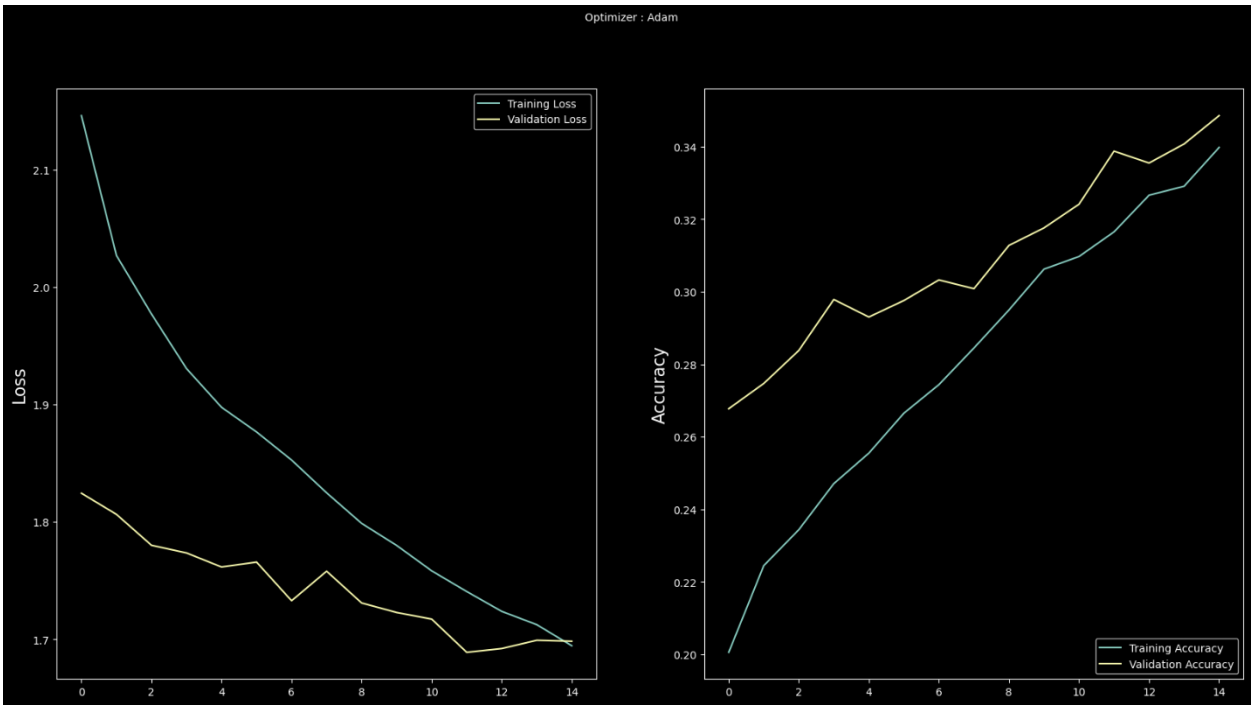

Figure 8: Second model using Keras and OpenCV

We have analyzed data by visualizing the training and validation loss as well as the training and validation accuracy over the epochs using matplotlib. Let's analyze the visualization:

Loss Plot:

The left subplot displays the loss values. The y-axis represents the loss, and the x-axis represents the epochs.

The blue line represents the training loss, and the orange line represents the validation loss.

By comparing the training and validation loss, you can assess if the model is overfitting or underfitting. If the training loss decreases significantly but the validation loss remains high, it indicates overfitting.

Ideally, you want to see both the training and validation loss decreasing and converging.

Accuracy Plot:

The right subplot displays the accuracy values. The y-axis represents accuracy, and the x-axis represents the epochs.

The blue line represents the training accuracy, and the orange line represents the validation accuracy.

The accuracy plot helps you understand how well the model is performing on the training and validation data.

It is desirable to see both the training and validation accuracy increasing and converging.

By analyzing the loss and accuracy plots, you can gain insights into the model's performance during training. If the training loss is decreasing while the validation loss is increasing, it might indicate overfitting. Similarly, if the training accuracy is increasing while the validation accuracy is not improving or decreasing, it might indicate overfitting or poor generalization.

Aspect	Keras	Keras & OpenCV
Training Time	Average duration per epoch: 781s (ranging from 769s-797s)	Average duration per epoch: 33s (ranging from 28s-35s)
Loss	Range: 0.6961 - 1.7821	Range: 1.6945 - 2.1463
Accuracy	Range: 0.3134 - 0.7405	Range: 0.2006 - 0.3398
Validation Loss	Range: 1.0666 - 1.3936	Range: 1.6950 - 1.8245
Validation Accuracy	Range: 0.4669 - 0.6243	Range: 0.2678 - 0.3605

This table provides a comparison between the two code snippets based on their main logs. It includes information about the number of training time per epoch, loss, accuracy, validation loss and validation accuracy.

8.1 – Conclusion & Future Work:

In summary, our research involved the development of two models for emotion recognition using CNN. The first model, implemented with the Keras library, achieved a relatively high accuracy of 74%, albeit with limited feature extraction capabilities. On the other hand, the second model, which incorporated both Keras and OpenCV, exhibited higher feature extraction but lower accuracy at 34%. These results highlight the trade-off between accuracy and feature extraction in emotion recognition systems. Future work should explore methods to enhance feature extraction while maintaining a high level of accuracy, potentially through the adoption of advanced deep learning architectures or feature engineering techniques.

9 – References and patents:

1. Mehrabian A (2017) Nonverbal communication. Routledge, London
2. Liam Schoneveld, Alice Othmani, Hazem Abdelkawy. (2021). Laveraging recent advances in
3. deep learning for audio-Visual emotion recognition. Pattern Recognition Letters 146, 1-7.
4. Yishu Liu, Guifang Fu. (2021). Emotion recognition by deeply learned multi-channel textual.

   and EEG features. Futures Generation Computer Systems 119, 1-6.

5. Ansari, s., 2017. Pattern Recognition. [Online]
   Available at: https://www.geeksforgeeks.org/pattern-recognition-introduction.
6. Neha Jain, Shishir Kumar, Amit Kumar, Pourya Shamsolmoali, Masoumeh Zareapoor. (2018).
   Hybrid deep neural networks for face emotion recognition. Pattern Recognition Letters 115,
   101-106.
7. "LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444."
8. Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning.
9. Breiman, L. (2001). Random forests. Machine Learning.
10. Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification.
11. Deep Learning with Python" by Francois Chollet.
12. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron.
13. Learning OpenCV 4 Computer Vision with Python 3" by Joseph Howse, Joe Minichino, and Prateek Josh.
14. OpenCV 4 with Python Blueprints" by Gabriel Garrido Calvo and David Millan Escriva.
15. Deep Learning with Keras" by Antonio Gulli and Sujit Pal.
16. Python Deep Learning Cookbook" by Indra den Bakker and Douwe Osinga.
17. Support Vector Machines Succinctly" by Alexandre Kowalczyk.
18. Support Vector Machines for Pattern Classification" by Shigeo Abe.
19. Understanding Support Vector Machines: A Visual Explanation with Examples" by Koby Keck and Marc Wiedermann.

20. Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions" by Yunqian Ma and David M. W. Powers.
21. Pattern Classification" by Richard O. Duda, Peter E. Hart, and David G. Stork.
22. Emotions Revealed: Recognizing Faces and Feelings to Improve Communication and Emotional Life" by Paul Ekman.
23. The Tell: The Little Clues That Reveal Big Truths about Who We Are" by Matthew Hertenstein.
24. Emotional Intelligence: Why It Can Matter More Than IQ" by Daniel Goleman.
25. Unmasking the Face: A Guide to Recognizing Emotions from Facial Clues" by Paul Ekman and Wallace V. Friesen.
26. Reading Emotions from the Face: An Ethnographic Approach" by Erika L. Rosenberg.
27. Advanced Deep Learning with Keras: Apply Deep Learning Techniques, Autoencoders, GANs, Variational Autoencoders, Deep Reinforcement Learning, Policy Gradients, and More" by Rowel Atienza.
28. Practical Convolutional Neural Networks: Implement advanced deep learning models using Python" by Nikhil Ketkar.
29. Python Deep Learning Cookbook: Over 75 practical recipes on neural network modeling, reinforcement learning, and transfer learning using Python" by Indra den Bakker.