Indian Institute of Technology, Kanpur

MTH392A Report

FACIAL EXPRESSION RECOGNITION

Author
Keshav Ranjan

Supervisor
Dr. Amit Mitra

11th April 2023

CERTIFICATE

This is to certify that the project entitled "Facial Expression Recognition" submitted by Keshav Ranjan (Roll no.: 200508) as a part of MTH392A course offered by the Indian Institute of Technology, Kanpur, is a Bonafide record of the work done by him under my guidance and supervision from 5th January 2023 to 10th April 2023.

Amit

Dr. Amit Mitra
Professor,
Dept. of Mathematics & Statistics
Indian Institute of Technology Kanpur

Annexure-II

DECLARATION

I/We hereby declare that the work presented in the project report entitled FACIAL EXPRESSION RECOGNITION contains my own ideas in my own words. At places, where ideas and words are borrowed from other sources, proper references, as applicable, have been cited. To the best of our knowledge this work does not emanate from or resemble other work created by person(s) other than mentioned herein.

Name and Signature Keshav Ranjan Keshav Ranjan

Date: 11/04/2023

Acknowledgement

I am very grateful to Indian Institute of Technology Kanpur and my project supervisor **Dr. Amit Mitra** for giving me this opportunity to pursue undergraduate research through **MTH392A** course. The experience which I got from this project is invaluable for my future endeavours and I'll be deeply indebted to **Dr. Amit Mitra** for mentoring me in this research project. I am thankful to him for taking out his time and giving me helpful feedback while writing the technical reports. This project was my first research experience and it made me realise my passion for **Statistics**.

Abstract

Facial expressions play a vital role in non-verbal communication between humans, and the development of automatic facial expression recognition is crucial for human-machine interactions. However, traditional machine learning methods for facial expression recognition often yield poor results due to the complexity of feature extraction.

In this research, we propose a solution that leverages recent advances in deep learning, particularly deep Convolutional Neural Networks (CNNs), to accurately interpret the semantic information in facial expressions without the need for hand-crafted feature descriptors. We also explore different loss functions and training techniques to improve the CNNs' classification power.

Our experimental results demonstrate that the proposed networks outperform state-of-the-art methods on both the FERC-2013 data-set provided on the Kaggle facial expression recognition competition and the Extended Cohn-Kanade(CK+) data-set. Furthermore, we show that our CNNs have a significantly reduced number of parameters compared to the winning model of the competition[7], making them well-suited for real-time systems that require faster performance.

In summary, our research advances automatic facial expression recognition and its potential impact on human-machine interactions. The use of deep CNNs offers a promising solution to the challenges of feature extraction and classification in facial expression recognition.

Keywords- Facial Expression Recognition, Convolutional Neural Network, Deep Learning

Contents

Chapter 1

INTRODUCTION

Facial expressions serve as a visual representation of a person's emotional state, cognitive processes, intentions, personality, and potential mental health issues, and play a vital role in interpersonal communication. The study of facial expressions has been ongoing for a considerable time, and there have been significant advancements in recent decades. However, accurately identifying facial expressions remains a challenging task due to their intricate and diverse nature. The complexity and diversity of facial expressions make it difficult to achieve high accuracy rates, even with technological advancements.[1]

Humans use various nonverbal cues, such as gestures, facial expressions, and body language, to convey their intentions and emotions during communication. Among these cues, facial expressions play a significant role and are widely used for communication purposes. Facial expression recognition systems are gaining popularity due to their potential applications in fields like lie-detection, medical diagnosis, human-computer interfaces and psychological analysis. Automated facial expression recognition can enable robots to anticipate and appropriately respond to human emotions, thereby enhancing their social interaction abilities[7]. The success of such systems depends on their ability to accurately detect and extract facial expressions from images.

1.1 Motivation

The Facial Expression Recognition System has the potential to be used in various domains, such as healthcare, education, customer service, automated cars, and music experiences. Recognizing and responding to human emotions is a significant step towards improving our daily lives. For example, in automated cars, the system can recognize passengers' emotions and create a more comfortable and personalized experience, increasing trust in and acceptance of autonomous driving technology. Similarly, a personalized music system that predicts our emotions and selects appropriate songs to match our mood can have a significant impact on our daily lives.

1.2 Problem Statement

The current **state-of-the-art** in facial expression recognition focuses on the recognition of the **six** basic expressions **(angry, disgust, fear, happy, sad, surprise)** and one **neutral** expression. However, it has been observed that these expressions are insufficient to describe all facial expressions and that expressions can be better categorized based on facial actions. The primary challenge in this field is detecting the face and recognizing the facial expression accurately. To achieve this, it is vital to pay attention to primary components such as face configuration, orientation, and location where the face is set. Moreover, the choice of the **loss function** used to make the prediction plays a crucial role in the accuracy of facial expression recognition. Therefore, the task is to compare the **performance of different loss functions** in recognizing facial expressions, including the six basic expressions and other expressions categorized based on facial actions, and to determine which **loss function** performs the best.

1.3 Objectives

- Develop a facial expression recognition system using two data-sets.
- Compare the performance of different loss functions on both data-sets.

- Evaluate the performance of the system using accuracy, loss on each data-set.
- Investigate the effect of different hyper-parameters on the system's performance on each data-set.

Chapter 2

Literature Review

Facial expression recognition (FER) is a field of computer vision and has significant applications in various areas of human-computer interaction, such as gaming, virtual reality, and robotics. It involves detecting and classifying the emotional state of a person based on their facial expressions. In recent years, deep learning techniques, particularly convolutional neural networks (CNNs), have demonstrated remarkable improvements in FER accuracy. CNNs are a type of neural network that is widely used in image recognition tasks due to their ability to learn high-level features from raw image data.
One of the papers by Md. Mohsin Kabir et al proposed a CNN-LSTM approach to FER, which involves training a CNN to extract features from facial images and then feeding them into an LSTM network to learn temporal dependencies. The authors used the CK+ dataset, which contains facial images labeled with seven basic emotions. The proposed approach achieved an accuracy of 86.87%, outperforming other state-of-the-art methods.[1]
Another paper by Dinh Viet Sang et al proposed a CNN-based approach to FER, which involves training a deep CNN to extract high-level features from facial images. The proposed approach achieved an accuracy of **97.16**% on CK+ dataset and **69.4**% on FER2013 dataset outperforming other state-of-the-art methods at that time, won the competition conducted by KAGGLE on FER2013 dataset in 2013 ,and also from many models at present.[2][7]
Another study by Zhang et al proposed a facial expression recognition model based on a combination of deep residual networks (ResNets) and attention mechanisms. The proposed model achieved an accuracy of **94.9**% on the CK+ dataset.[3] A Comparative Study of CNNs and RNNs by Omkar M. Parkhi, et al compares the performance of CNNs and RNNs on the FER2013 dataset. They experimented with various architectures and hyperparameters for both CNNs and RNNs and provide insights into the strengths and weaknesses of each approach[4]. One paper by Sulaiman Muhammad presents a real-time emotion-based music player that uses CNN architectures to recognize emotions from facial expressions. The authors used the FER2013 dataset to train their model and achieved a recognition accuracy of 65.67%.[5]
Hence, My research is focused on finding new ways to improve the accuracy of convolutional neural network (CNN) models without using pre-processing techniques like data augmentation. Specifically, I am working on model (which has similarity with VGG16 architecture) to reduce the number of parameters required for training while still maintaining or improving performance on different datasets. One important aspect of my work is to investigate the role of the loss function in optimizing the model. I understand that the choice of loss function can greatly affect the ability of the model to accurately classify images, so I plan to experiment with different loss functions to see which ones work best for my model. Overall, my goal is to contribute to the field of computer vision by developing a more efficient and accurate CNN model. By reducing the number of parameters required for training and improving the model's performance on various datasets, I hope to help researchers and practitioners build better image recognition systems that can be used in a variety of applications.

Chapter 3

Background

3.1 Convulation Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep neural network designed for processing spatial data, such as images. They achieve this by using **convolutional kernels** to extract features from the input image, such as edges, corners, and textures. A typical CNN architecture consists of several layers, each with a specific function. The first layer is usually a convolutional layer, followed by a pooling layer to reduce dimensionality and improve robustness. The final layer is typically a fully connected layer, which uses the extracted features to make a prediction. Additional layers, such as dropout and normalization layers, can be added to prevent **overfitting** and improve stability. **Hyperparameters**, such as the number of layers and the size of filters, control the learning process and must be tuned to achieve optimal performance. CNNs have proven to be highly effective in image recognition, speech recognition, natural language processing, and other tasks that involve pattern recognition. Lets discuss about some **important parameters in** details:

3.1.1 Convolution for Images:

Convulational Layers:-

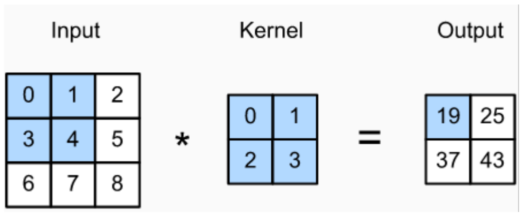
•A convolutional layer cross-correlates the input and kernel and adds a scalar bias to produce an output.
•The two parameters of a convolutional layer are the kernel and the scalar bias.
•filters parameter defines the number of filters to use, which determine the number of features the layer can learn.
•kernel_size parameter defines the size of the filter, which determines the size of the receptive field of the filter.

Cross-Correlation Operation:-

•The cross-correlation operation is used in the convolutional layer of a CNN to learn and extract features from the input data. By applying multiple filters with different learned weights, The cross-correlation of a filter F with an input image I at spatial coordinates (x, y) is defined as:

$$(F * I)(x, y) = \sum_i \sum_j F(i, j) \cdot I(x + i, y + j)$$

where F is the filter, I is the input image, and (x, y) are the spatial coordinates of the output feature map.
Note: the symbol * represents the cross-correlation operation in this context. Given an input of size $n_h \times n_w$ and a kernel size of $k_h \times k_w$, its output size is given by $(n_h - k_h + 1, n_w - k_w + 1)$.
Applying cross-correlation operation on a input of size 3×3 and kernel size 2×2 giving output size 2×2.



Padding and Strides:-

•Padding maintains the spatial dimensions of the input after convolution by adding zeros around the input.In general, if we add a total of p_h rows of padding (roughly half on top and half on bottom) and a total of p_w columns of padding (roughly half on the left and half on the right), the output size will be

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

•Stride determines the amount of shift between the filter and the input, affecting the resolution and size of the output feature maps.
In general, when the stride for the height is s_h and the stride for the width is s_w and if we add a total of p_h rows of padding (roughly half on top and half on bottom) and a total of p_w columns of padding (roughly half on the left and half on the right), the output shape will be

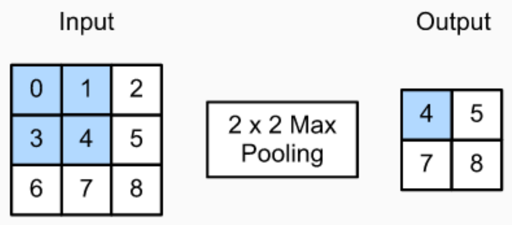
$$\text{Output shape} = \left\lfloor \frac{n_h - k_h + p_h}{s_h} \right\rfloor + 1 \times \left\lfloor \frac{n_w - k_w + p_w}{s_w} \right\rfloor + 1$$

•Choosing appropriate padding and stride values is a trade-off between computational efficiency and model performance, and can depend on the specific task, input data, and CNN architecture.

3.1.2 Pooling Layers:

Max-Pooling Layers:-

•MaxPool2D layer performs down-sampling by taking the maximum value of each pool, reducing the spatial dimensions of the feature map while preserving the most important features.
•pool_size parameter defines the size of the pooling window and strides parameter defines the stride of the pooling window.
•Similarly, AvgPool2D layer performs down-sampling by taking the maximum value of each pool.



Max-pooling with a pooling window shape of 2×2. The shaded portions are the first output element as well as the input tensor elements used for the output computation: $\max(0, 1, 3, 4) = 4$.

3.1.3 Activation Layers:

•Activation layer applies an activation function to the output of each neuron, adding non-linearity to the model.
•Here are some activation layers used in our project:-

Sigmoid:-

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

In this equation, x is the input to the function, and e is the mathematical constant known as Euler's number (approximately equal to 2.71828). The function maps any input value to a range between 0 and 1, which can be interpreted as a probability.

Relu(Rectified Linear Unit):-

$$\mathrm{ReLU}(x) = \max(0, x)$$

In this formula, x is the input to the ReLU function, and the max function outputs the maximum of 0 and x. This means that any negative input value will be set to 0, while positive input values are unchanged. The ReLU function is applied element-wise to the output of each neuron in a CNN.

Softmax:-

•The softmax function is a mathematical function commonly used for multi-class classification problems.
•It maps a vector of real-valued numbers to a probability distribution over the classes. Here's the formula for the softmax function:

$$\mathrm{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

In this formula, $\mathbf{x}$ is a vector of real-valued numbers, and i ranges from 1 to K, the number of classes. The softmax function outputs a vector of probabilities, where each element represents the probability of the input belonging to a particular class.

The numerator in the formula is the exponential function of the i-th element of the input vector, while the denominator is the sum of the exponential functions of all elements in the input vector. The resulting vector has values between 0 and 1, and the sum of all values equals 1, which makes it a valid probability distribution.

3.1.4 Dropout Layers:

•Dropout layer randomly drops out some fraction of the units (neurons) in the layer during training, reducing the overfitting of the model.
•Rate parameter defines the fraction of the input units to drop.
Let $X = (x_1, x_2, \ldots, x_n)$ be the input to a layer with n units. During training, we apply dropout to X as follows:

We randomly choose a set of units to keep active, with probability p of keeping a unit and $1 - p$ of setting it to zero.

We create a mask M of the same shape as X, where each element m_i is drawn from a Bernoulli distribution with parameter p (i.e., $m_i = 1$ with probability p and $m_i = 0$ with probability $1 - p$).

We apply the mask M elementwise to X, so that the output of the dropout layer is given by:

$$Y = M \odot X$$

where $\odot$ denotes elementwise multiplication.

During testing, we do not apply dropout and simply pass the input through the layer unchanged.

Note that the dropout rate p is a hyperparameter that must be chosen by the user. Typically, values of p between 0.2 and 0.5 are used in practice.

3.1.5 Dense Layers:-

• Dense layer is a fully connected layer where every neuron in the previous layer is connected to every neuron in the current layer.
•units parameter defines the number of neurons in the layer.
•activation parameter defines the activation function used in the layer.

3.2 Batch Normalisation:

•BatchNormalization layer normalizes the input to each neuron, making the model more robust and reducing overfitting.
•It helps to stabilize the learning process and speeds up the training process.
Batch normalization for fully connected layers:

Given a batch of inputs $X = x_1, x_2, ..., x_m$, where each $x_i \in \mathbb{R}^d$ is a vector of d features, batch normalization applies the following transformations:

- Calculate the mean and variance of the batch:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

- Normalize the features using the mean and variance:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

  where ϵ is a small constant added for numerical stability.

- Scale and shift the normalized features using learned parameters:

$$y_i = \gamma \hat{x}_i + \beta$$

  where γ and β are learned scaling and shifting parameters, respectively.

Batch normalization for convolutional layers:-
Given a batch of inputs $X = x_1, x_2, ..., x_m$, where each $x_i \in \mathbb{R}^{h\times w\times c}$ is a feature map of height h, width w, and c channels, batch normalization applies the following transformations:

- Calculate the mean and variance of each feature map over the batch:

$$\mu_{B,j} = \frac{1}{mhw}\sum_{i=1}^{m}\sum_{k=1}^{c} x_{ijk}\ \sigma_{B,j}^2 \qquad = \frac{1}{mhw}\sum_{i=1}^{m}\sum_{k=1}^{c}(x_{ijk} - \mu_{B,j})^2$$

  where j indexes the feature maps and k indexes the channels.

- Normalize the features using the mean and variance for each feature map:

$$\hat{x}_{ijk} = \frac{x_{ijk} - \mu_{B,j}}{\sqrt{\sigma_{B,j}^2 + \epsilon}}$$

  where ϵ is a small constant added for numerical stability.

- Scale and shift the normalized features using learned parameters:

$$y_{ijk} = \gamma_j \hat{x}_{ijk} + \beta_j$$

  where γ_j and β_j are learned scaling and shifting parameters for each feature map, respectively.

In both cases, batch normalization can help to improve the stability and accuracy of the model by reducing the internal covariate shift, which is the tendency of the distribution of activations to shift during training.

3.3 Regularization:

•Kernel_regularizer parameter applies L2 regularization to the kernel weights of the convolutional layers to prevent overfitting.
•It adds a penalty term to the loss function, which encourages the model to learn simpler and smoother representations.

3.3.1 L2 Regularization:-

The L2 regularization penalty term for a weight matrix w in a layer is given by:

$$L_{reg} = \frac{\lambda}{2}\sum_{i=1}^{n} w_i^2 \tag{3.1}$$

where L_{reg} is the regularization term, λ is the regularization strength (a hyperparameter that controls the amount of regularization applied), n is the number of weights in the model, and w_i is the ith weight. The regularization term is added to the original loss function, so the total loss becomes:

$$L_{total} = L_{original} + L_{reg} \tag{3.2}$$

where $L_{original}$ is the original loss function (e.g., cross-entropy loss, l2-multiclass svm, etc.).

During training, the weights are updated using gradient descent, but with an additional term that incorporates the gradient of the regularization term:

$$\Delta w_i = -\eta \left(\frac{\partial L_{total}}{\partial w_i} + \lambda w_i \right) \tag{3.3}$$

where Δw_i is the update to the ith weight, η is the learning rate (a hyperparameter that controls the step size of the gradient descent algorithm), and $\frac{\partial L_{total}}{\partial w_i}$ is the gradient of the total loss function with respect to the ith weight. The second term in the parentheses is the gradient of the regularization term, which penalizes large values of w_i.

In summary, L2 regularization is a technique for reducing overfitting in machine learning models by adding a penalty term to the loss function that encourages the weights of the model to be small. This penalty term is proportional to the square of the L2 norm of the weights, and is controlled by a hyperparameter λ.

3.4 Loss Functions:

3.4.1 Cross-Entropy Loss Function:-

The cross-entropy loss function for a CNN model can be written as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \log(\hat{y}_{i,j})$$

where L is the loss function, N is the number of examples in the training set, C is the number of classes, $y_{i,j}$ is the true label of the ith example for class j, and $\hat{y}_{i,j}$ is the predicted probability of the ith example for class j.

The predicted probability $\hat{y}_{i,j}$ is calculated using the softmax function:

$$\hat{y}_{i,j} = \frac{\exp(z_{i,j})}{\sum_{k=1}^{C} \exp(z_{i,k})}$$

where $z_{i,j}$ is the logit (pre-softmax) value for the ith example and the jth class.

The logit values $z_{i,j}$ are calculated by passing the input data x_i through the layers of the CNN model:

$$z_{i,j} = f_j(x_i)$$

where f_j is the output of the final fully connected layer for class j.

During training, the weights and biases of the CNN model are updated using gradient descent. The gradient of the loss function with respect to the weights and biases is calculated using backpropagation:

$$\frac{\partial L}{\partial w_{ij}} = -\frac{1}{N} \sum_{i=1}^{N} (y_{i,j} - \hat{y}_{i,j})\, x_i$$

$$\frac{\partial L}{\partial b_j} = -\frac{1}{N} \sum_{i=1}^{N} (y_{i,j} - \hat{y}_{i,j})$$

where w_{ij} is the weight connecting the ith input neuron to the jth output neuron, b_j is the bias term for the jth output neuron, and x_i is the input data for the ith example.

In summary, the cross-entropy loss function is a commonly used loss function for multi-class classification problems. It measures the dissimilarity between the true label and predicted probability for each example in the training set, and is used to update the weights and biases of a CNN model during training.

3.4.2 l2-multiclass SVM loss function:-

$$L(w) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq y_i}\max\left(0, s_j - s_{y_i} + \Delta\right)^2$$

where:

- w is the weight vector of the model
- N is the number of training examples
- y_i is the true label of the i-th example
- s_j is the score of the j-th class for the i-th example
- Δ is the margin parameter, which controls the amount of separation between the correct score and the incorrect scores

This is the standard form of the L2 SVM loss function for multiclass classification, where $j \neq y_i$ means that we sum over all classes except the correct class. Note that this equation is a bit different from the code implementation I described earlier, but they are both equivalent formulations of the L2 SVM loss function.

3.5 Adam Optimiser:

The Adam optimizer maintains two moving averages of the gradient: the first moment m_t (mean) and the second moment v_t (uncentered variance). It computes the learning rate for each parameter based on these moving averages and updates the parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}t} + \epsilon}\hat{m}t \tag{3.4}$$

where θ_t is the vector of parameters at time step t, α is the learning rate, $\hat{m}_t$ is the estimate of the first moment of the gradient at time step t, $\hat{v}_t$ is the estimate of the second moment of the gradient at time step t, and ϵ is a small constant added to the denominator to prevent division by zero.

The estimates of the first and second moments are computed as follows:

$$\hat{m}t = \beta_1\hat{m}t - 1 + (1-\beta_1)g_t \tag{3.5}$$

$$\hat{v}t = \beta_2\hat{v}t - 1 + (1-\beta_2)g_t^2 \tag{3.6}$$

where g_t is the gradient at time step t and β_1 and β_2 are decay rates for the first and second moments, respectively.

The Adam optimizer also uses bias correction to adjust the estimates of the first and second moments at the beginning of training when the moving averages are biased towards zero. The bias-corrected estimates are calculated as follows:

$$\hat{m}^t = \frac{\hat{m}_t}{1-\beta_1^t} \tag{3.7}$$

$$\hat{v}^t = \frac{\hat{v}_t}{1-\beta_2^t} \tag{3.8}$$

The hyperparameters for the Adam optimizer are the learning rate α, the decay rates β_1 and β_2, and the small constant ϵ. These hyperparameters can be tuned to optimize performance on a specific task.

Chapter 4

Proposed Work and Evaluations

The experiments were conducted on a laptop with the following specifications: AMD Ryzen 7 4800H Processor (8 cores, 16 threads, 2.9 GHz base clock, 4.2 GHz boost clock), 16 GB DDR4 RAM, Nvidia GeForce GTX 1650Ti graphics card, running Windows 11 Home 64-bit operating system. The experiments were conducted using Jupyter Notebook, version 6.1.4, and Python programming language, version 3.8.5. with the help of TensorFlow, version 2.11.0.

4.1 Data Collection

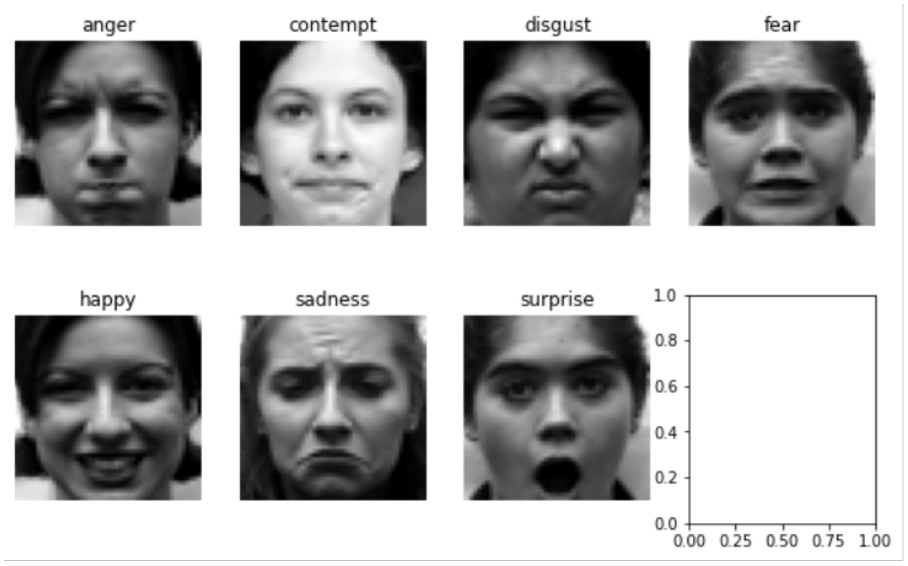
4.1.1 FER2013 Dataset

The FER2013 dataset(publically avalaible) is a popular benchmark dataset for facial expression recognition research. It consists of 35,887 grayscale images of faces with 48x48 pixel resolution. The FER2013 dataset was created by researchers at the Google Brain team and was originally intended for use in the Kaggle facial expression recognition challenge. Kaggle has divided into 28,709 training images, 3589 public test images and 3589 private test images. The dataset was compiled from a variety of sources, including the internet and public databases, and is considered to be a challenging dataset due to the variability in image quality, lighting conditions, and facial expressions. Each image contains a human face that is not posed (in the wild). The images are divided into seven classes representing different facial expressions: anger, disgust, fear, happiness, sadness, surprise, and neutral.
One of the notable characteristics of the FER2013 dataset is its class imbalance. The majority of images in the dataset are labeled as neutral, while the other classes have fewer examples. This makes it challenging for models to accurately recognize the less frequent expressions. Another challenge is that some of the images in the dataset have low quality or are difficult to interpret due to occlusions, variations in lighting and facial features.

FER2013 dataset

4.1.2 Extended Cohn-Kanade Dataset

To collect the CK+ dataset(publically available), researchers used a high-quality camera to capture the images of subjects displaying seven different emotions - anger, contempt, disgust, fear, happiness, sadness, and surprise. The images were captured under controlled lighting conditions and with consistent head poses. The dataset consists of a total of 930 images, with each emotion having a varying number of images. The dataset also includes annotations for the location of facial landmarks and the intensity of the emotion displayed in each image, which makes it a valuable resource for training and evaluating facial expression recognition models.



CK+ dataset

4.2 Proposed Model Architecture

4.2.1 First Model:

The First Propsed model is a convolutional neural network (CNN) designed for image classification on grayscale images with a size of 48x48 pixels. It consists of three sets of convolutional and batch normalization layers with ReLU activation, followed by max pooling and dropout layers. The model has 32, 64, and 128 filters in the convolutional layers respectively, each with a 3x3 kernel size. The max pooling layers have a pool size of 2x2. The purpose of using batch normalization and dropout layers is to prevent overfitting, while the convolutional and max pooling layers learn the features of the input images. The model also includes a flatten layer, which converts the output of the convolutional layers into a one-dimensional vector. This vector is then fed into a dense layer with 1024 units and ReLU activation, followed by batch normalization and dropout layers. Another dense layer with ReLU activation and dropout is then added, followed by a final dense layer with softmax activation that outputs the predicted probabilities for each of the 7 classes. Overall, the model is designed to efficiently extract useful features from input images and make accurate predictions on the image classes. Nevertheless, if one use the multi-class SVM loss during the training process, the softmax layer can be skipped.(Table 4.1)

4.2.2 Second Model:

The Second Proposed model is a convolutional neural network (CNN) designed for image classification on grayscale images with a size of 48x48 pixels. It consists of four sets of convolutional and batch normalization layers with ReLU activation, followed by max pooling and dropout layers. The model has 32, 64, 128, and 512 filters in the convolutional layers respectively, with kernel sizes of 3 or 5, and with padding to maintain the spatial dimensions of the feature maps. The max pooling layers have a pool size of 2x2. The purpose of using batch normalization and dropout layers is to prevent overfitting, while the convolutional and max pooling layers learn the features of the input images. The model also includes a flatten layer, which converts the output of the convolutional layers into a one-dimensional vector. This vector is then fed into two dense layers with 256 and 512 units, respectively, both with ReLU activation, followed by batch normalization and dropout layers. Another dense layer with softmax activation is then added, which outputs the predicted probabilities for each of the 7 classes. Overall, the model is designed to efficiently extract useful features from input images and make accurate predictions on the image classes, with the regularization techniques employed to prevent overfitting. Nevertheless, if one use the multi-class SVM loss during the training process, the softmax layer can be skipped.(Table 4.2)

Table 4.1: **First Model Architecture**

Layer (type)	**Output Shape**	**Param #**
conv2d	(None, 46, 46, 32)	320
conv2d_1	(None, 44, 44, 32)	9,248
batch_normalization	(None, 44, 44, 32)	128
max_pooling2d	(None, 22, 22, 32)	0
dropout	(None, 22, 22, 32)	0
conv2d_2	(None, 20, 20, 64)	18,496
batch_normalization_1	(None, 20, 20, 64)	256
conv2d_3	(None, 18, 18, 64)	36,928
batch_normalization_2	(None, 18, 18, 64)	256
max_pooling2d_1	(None, 9, 9, 64)	0
dropout_1	(None, 9, 9, 64)	0
conv2d_4	(None, 7, 7, 128)	73,856
batch_normalization_3	(None, 7, 7, 128)	512
conv2d_5	(None, 5, 5, 128)	147,584
batch_normalization_4	(None, 5, 5, 128)	512
max_pooling2d_2	(None, 2, 2, 128)	0
dropout_2	(None, 2, 2, 128)	0
flatten	(None, 512)	0
dense	(None, 1024)	525,312
batch_normalization_5	(None, 1024)	4,096
dropout_3	(None, 1024)	0
dense_1	(None, 7)	71,655

Table 4.2: Second Model Architecture

Layer (type)	**Output Shape**	**Param #**	**Activation**
Conv2D	(None, 46, 46, 32)	320	ReLU
Conv2D	(None, 46, 46, 64)	18496	ReLU
BatchNorm	(None, 46, 46, 64)	256	-
MaxPool2D	(None, 23, 23, 64)	-	-
Dropout	(None, 23, 23, 64)	-	-
Conv2D	(None, 23, 23, 128)	204928	ReLU
BatchNorm	(None, 23, 23, 128)	512	-
MaxPool2D	(None, 11, 11, 128)	-	-
Dropout	(None, 11, 11, 128)	-	-
Conv2D	(None, 11, 11, 512)	590336	ReLU
BatchNorm	(None, 11, 11, 512)	2048	-
MaxPool2D	(None, 5, 5, 512)	-	-
Dropout	(None, 5, 5, 512)	-	-
Conv2D	(None, 5, 5, 512)	2359808	ReLU
BatchNorm	(None, 5, 5, 512)	2048	-
MaxPool2D	(None, 2, 2, 512)	-	-
Dropout	(None, 2, 2, 512)	-	-
Flatten	(None, 2048)	-	-
Dense	(None, 256)	524544	ReLU
BatchNorm	(None, 256)	1024	-
Dropout	(None, 256)	-	-
Dense	(None, 512)	131584	ReLU
BatchNorm	(None, 512)	2048	-
Dropout	(None, 512)	-	-
Dense	(None, 7)	3591	Softmax

4.3 Data Pre-processing

Data preprocessing is a critical step in any machine learning project. It involves preparing the raw data in a way that is suitable for use with machine learning algorithms. This typically includes steps such as cleaning the data, handling missing values, and transforming the data into a suitable format.
In this project, i used two datasets: the FER2013 dataset and the CK+ dataset. Each dataset required different preprocessing steps to prepare it for use in our machine learning models.

4.3.1 FER2013 Dataset

The data preprocessing steps for this dataset include:

- **Rescaling**: Each pixel value of the images is scaled down to a value between 0 and 1 by dividing each pixel value by 255.
- **Image Augmentation**: To increase the diversity of the training dataset, we applied various image augmentation techniques such as width_shift_range upto 10%, height_shift_range upto 10%, and horizontal_flip to the training images. This technique helps the model to learn more robust and generalized features from the dataset.
- **Grayscale conversion**: Since our dataset is in grayscale, we set the color_mode argument of the flow_from_directory method to grayscale, which converts the input images to grayscale.
- **Splitting the dataset**: We split the dataset into training and validation sets using the validation_split argument of the ImageDataGenerator method.
- **Resizing**: The input images are resized to (48, 48) using the target_size argument of the flow_from_directory method.
- **Batch size and class mode**: Finally, we set the batch_size and class_mode arguments of the flow_from_directory method to 64 and categorical, respectively. The class_mode categorical indicates that we have more than two classes in our dataset.

4.3.2 CK+ Dataset

The data preprocessing steps for this dataset include:

- **Loading the Images**: The images are loaded from the specified directory using the `os` library. The images are then resized to (48,48) using the `cv2.resize` method, and added to a list called `img_data_list`.
- **Normalization**: The pixel values of the images in `img_data_list` are then converted to float32 type and normalized by dividing each pixel value by 255.
- **Labeling**: The labels for the images are created as a NumPy array of ones with the same number of rows as the length of `img_data_list`, and then each label is assigned an integer value corresponding to the emotion it represents. The labels are then converted to categorical format using `np_utils.to_categorical`.
- **Shuffle and Split**: The data is shuffled using `shuffle` from the `sklearn.utils` library and split into training and testing sets using `train_test_split` from the `sklearn.model_selection` library.

4.4 Training

During the training phase, we used two different loss functions, namely cross-entropy loss and L2 MultiSVM loss, to optimize our models. Both loss functions were observed to perform comparably well in terms of accuracy and convergence rate. We also experimented with two different datasets, namely the FER2013 and CK+ datasets, to train our models.

In order to optimize the models, we used the Adam optimizer with a batch size of 64 and a momentum of 0.9. To prevent overfitting, we applied dropout regularization with a probability of 0.25. The biases of all layers were initialized to zero, and the weights were randomly initialized from a Gaussian distribution with a zero mean and a standard deviation given by $\delta = \sqrt{\frac{2}{2n_{input}}}$, Where n_{input} is the number of weights of each neuron[7]. For convolutional layers, n_{input} is calculated as the filter size multiplied by the filter size multiplied by the depth of the previous layer. For the fully connected layer,n_{input} is the number of neurons in the previous layer.[1]

For each iteration, we randomly selected a batch of 64 augmented images(7 in case of ck+) of size 48x48 from the training data and fed them to the network for training. After each epoch, the training data was randomly shuffled to avoid any bias. We trained each model for 50 to 100 epochs and monitored their performance on a validation set to avoid overfitting.[7]

We named our two proposed models "Model 1" and "Model 2", and trained each of them with both loss functions and both datasets. The training phase resulted in models with decent accuracy and good convergence rate, which we evaluated on the test set to measure their real-world performance. Overall, the training phase was a crucial step in developing our models and achieving state-of-the-art performance on facial expression recognition.

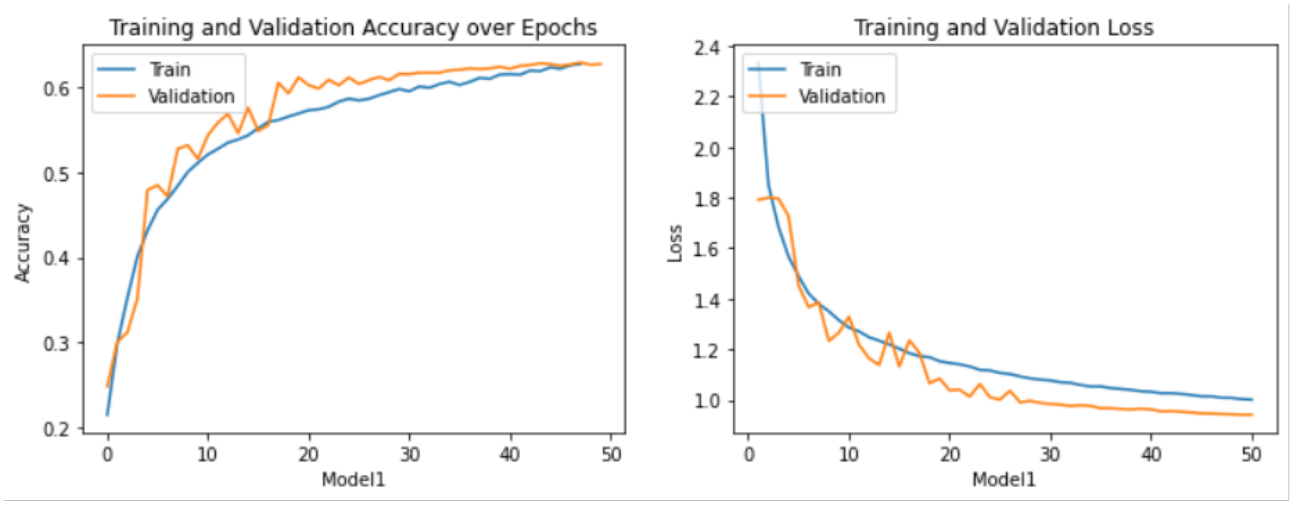


Fig 4.4.1: Model 1 on FER2013 using Cross-Entropy loss function

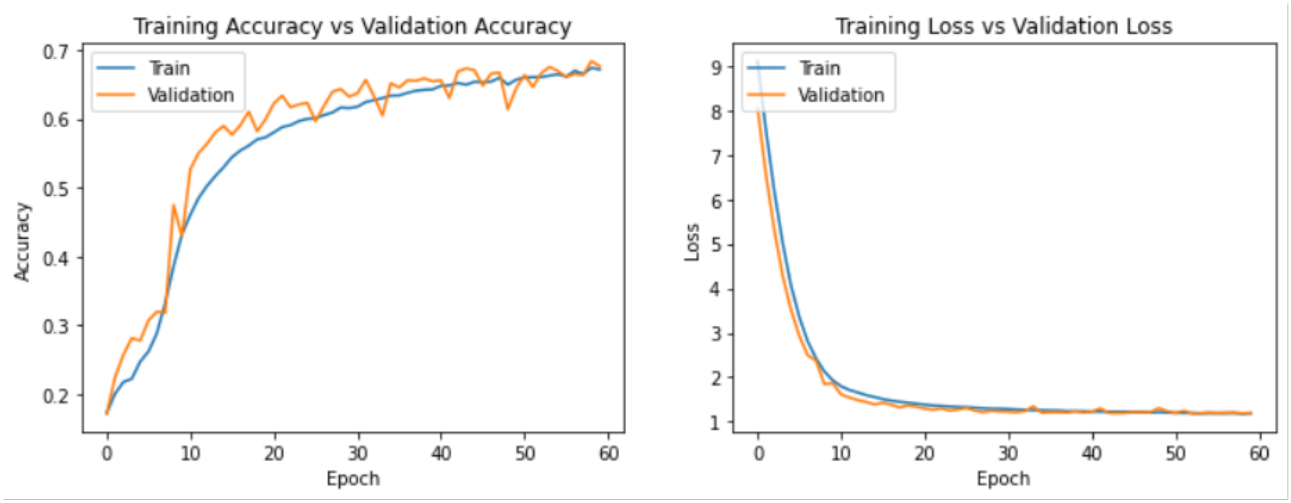


Fig 4.4.2: Model 2 on FER2013 using Cross-Entropy loss function

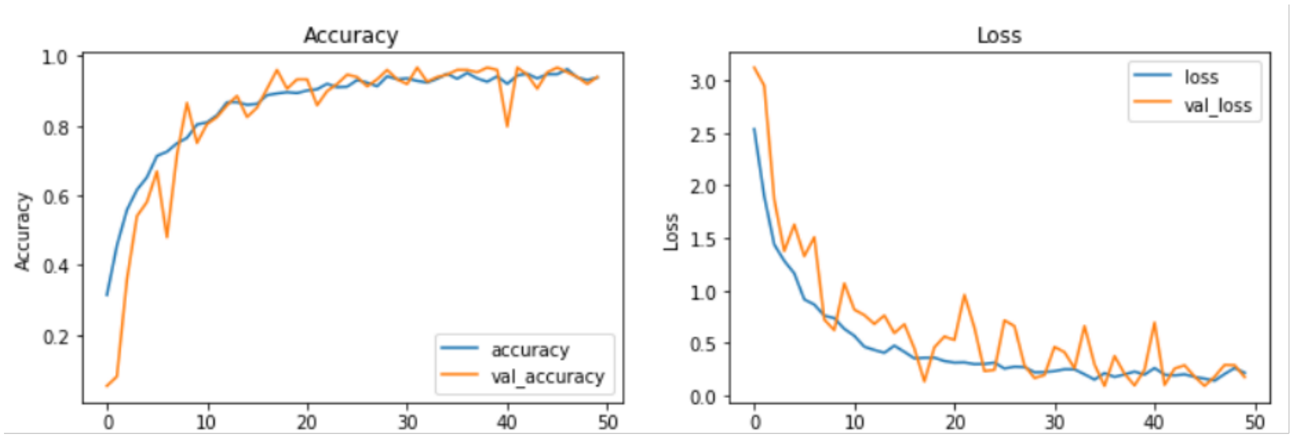


Fig 4.4.3: Model 1 on CK+ using Cross-Entropy loss function

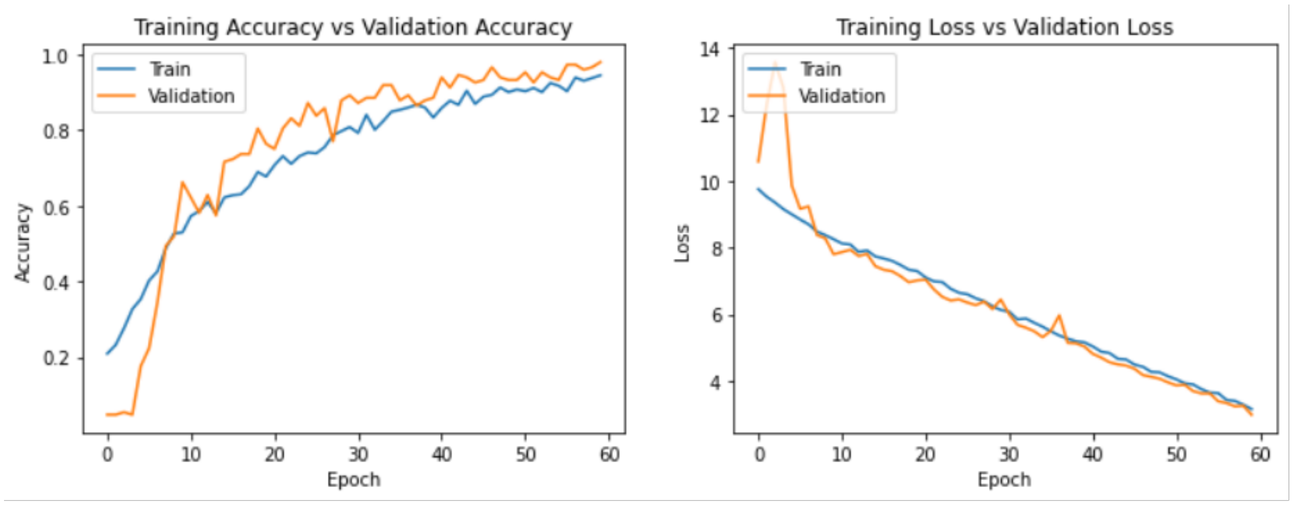


Fig 4.4.4: Model 2 on CK+ using Cross-Entropy loss function

4.5 Evaluation and Results

4.5.1 Effects of Data augmentation

The performance of Model 2 was evaluated with and without data augmentation on the Fer2013 dataset. Looking at the Table 4.3, the results show that data augmentation significantly improves the accuracy of the model. Specifically, the validation accuracy increased from 63.59% to 67.60%, and the test accuracy increased from 64.56% to 68.66%. These improvements suggest that data augmentation is an effective technique for improving the accuracy of machine learning models. I will be using data-augmentation by default from now for FER2013 dataset(no use of augmentation done for CK+ dataset)

Table 4.3: Affect of Data Augmentation on Model 2 Performance

Model	Data Augmentation	Validation Accuracy	Test Accuracy
2	No	63.59%	64.56%
2	Yes	67.60%	68.66%

4.5.2 Affects due to loss functions

Table 4.4 displays the accuracy on test sets obtained by applying two different loss functions, cross-entropy and L2 multi-class SVM, to Model 2 architectures. The results indicate that the cross-entropy loss function achieves better accuracy on both datasets, implying that it is more suitable for facial expression recognition tasks.

Table 4.4: Affect of loss functions

Model	Data Set	Loss Function	Test Accuracy
2	FER2013	Cross-Entropy + 'Softmax'	67.60%
2	CK+	Cross-Entropy+'Softmax'	97.97%
2	FER2013	L2-Multiclass SVM	65.64%
2	CK+	L2-Multiclass SVM	93.55%

4.5.3 Comparision Between Different Models

We are taking the cross-entropy loss function by default from now on. The accuracy of the proposed architectures on the FERC-2013 and CK+ datasets is shown in Table 4.5. The results reveal that there is a gradual increase in accuracy from Model 1 to Model 2. This increase in accuracy can be attributed to the increasing depth and complexity of the model from Model 1 to Model 2. With this increase in depth, the model becomes more capable of fitting the dataset, leading to improved accuracy. We tried to add a **Random-Forest Classifer(Random forest** is a machine learning algorithm that combines the predictions of multiple decision trees to improve accuracy and reduce overfitting. It works by randomly selecting subsets of the training data and features for each tree, and then taking a majority vote of their predictions. Random forest classifiers can handle high-dimensional feature spaces, non-linear relationships, and **noisy data.)** with my **Model 2**, but it **drastically decreases** the accuracy, so, we came to the conclusion that Random forest classifier is **not preferable** for **Multi- Classification Problems**. • My model has less parameters than the model made by Dinh Viet Sang et al[7] and its performance is almost same so, my model is better.

Table 4.5: Affect of complex Models

Model	Data Set	Test Accuracy	Parameters
1	FER2013	62.16%	0.88M
1	CK+	93.92%	0.88M
2	FER2013	67.60%	3.8M
2	CK+	97.97%	3.8M

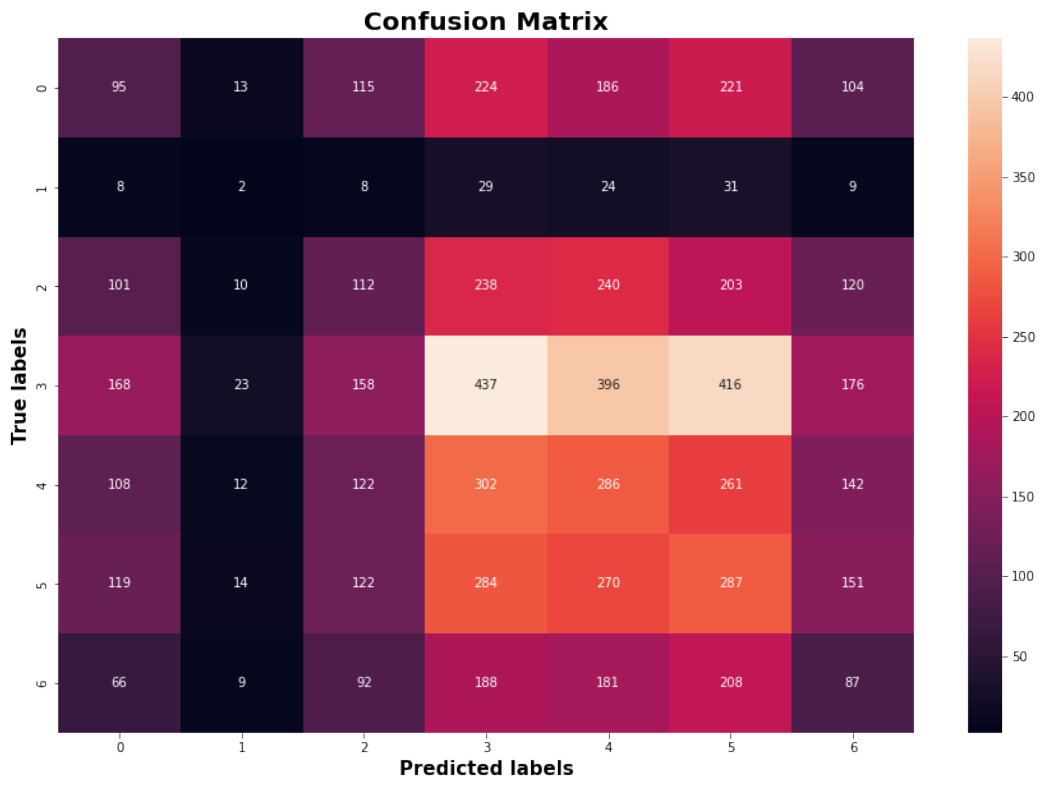
4.5.4 Confusion Matrix

The **confusion matrix** is a performance evaluation tool that allows us to assess the accuracy of a classification model by comparing its predictions with the actual labels of a dataset. In this case, the **confusion matrix of Model 2 on the FER2013 dataset**(Figure 1) and **confusion matrix of Model 2 on the ck+ dataset**(Figure 2) and provides us with valuable insights into how well our model performs on each emotion category.

The rows of the confusion matrix correspond to the true values or labels, while the columns represent the predictions made by the model. The values in each cell indicate the number of images that were correctly classified (true positives and true negatives) or misclassified (false positives and false negatives) by the model.

The analysis of **the confusion matrix of FER2013** reveals that the **Disgust class** has the lowest accuracy of all the emotion categories, indicating that our model struggles to correctly identify this emotion. On the other hand, the **Happy class** has the highest accuracy, suggesting that our model is well-suited for detecting this emotion.Another interesting observation is that our model 2 misclassifies **19**% of the images labeled as Fear and predicts them as Sad. This misclassification is consistent with human's mispredictions on the same images, indicating that our model may be learning similar patterns as humans when recognizing these two emotions.(See Appendix)

After the analysis of **the confusion matrix of ck+(Figure 2)**, we can see that our model 2 has performed an amazing job of classifying the emotions in the CK+ dataset. Each image is being classified mostly correctly, which is a remarkable achievement.The diagonal of the confusion matrix indicates the number of correctly classified instances for each emotion. From the confusion matrix, we can see that the model 2 correctly classified **100**% of the instances of Happiness, Sadness, Fear and Surprise. Moreover, it has achieved high accuracy on other emotion classes as well, with more than **98**% of the instances being classified correctly.(See Appendix)



where 0-Angry, 1-Disgust,2-Fear,3-Happy,4-'Neutral',5-'Sad',6-'Surprise'
Figure 1 : confusion matrix of Model 2 on the FER2013 dataset

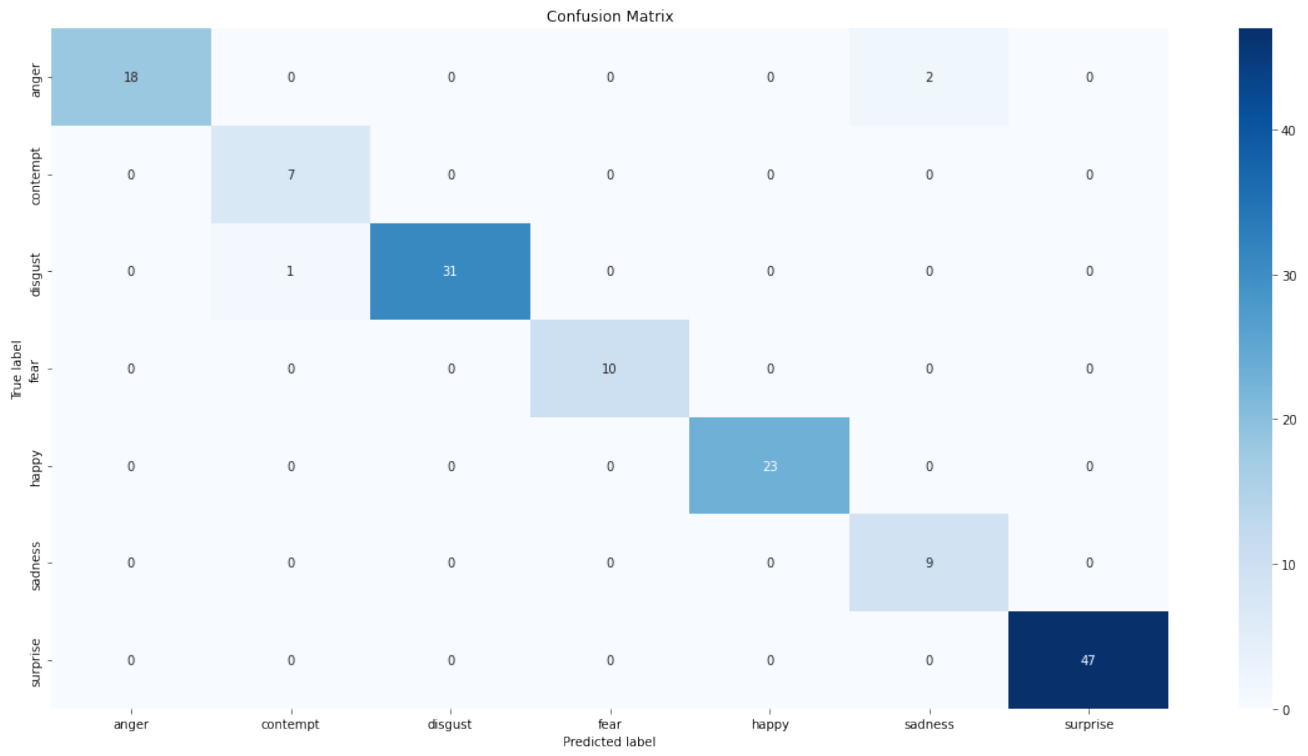
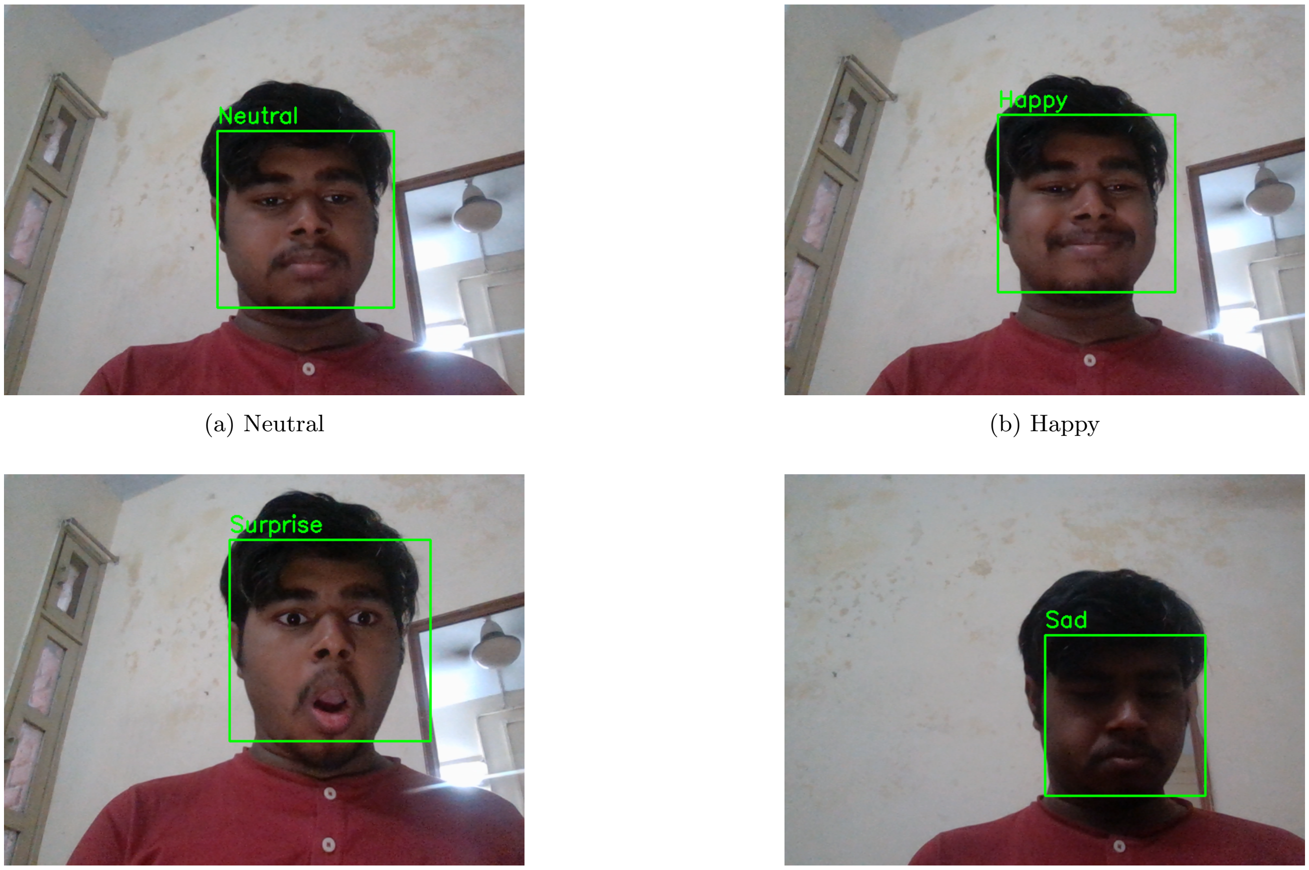


Figure 2 : confusion matrix of Model 2 on the ck+ dataset

4.5.5 Real-Time performance

Real-time performance is a critical aspect of a webcam-based expression recognition model and the Ultimate goal is to provide smooth functioning of model in real life. The model must be able to make accurate predictions in real-time to provide a seamless user experience. Real-time performance can be measured in terms of the speed at which the model processes incoming video frames, as well as the overall frame rate of the webcam.

Based on our evaluation of the webcam-based expression recognition model, it seems that the model is performing well in terms of real-time performance. It has an average frame rate of 30 ms per frame, which is good for real-time applications. This means that the model is able to process each video frame in less than one second, which is within the time it takes for the webcam to capture each frame. However, we have identified some issues with the model's accuracy for certain expressions. Specifically, the model is fluctuating in its predictions for the Sad, Anger, and Fear expressions, indicating that it may need further optimization or additional training data to improve its accuracy for these expressions.Additionally, the model is less predictable for the Disgust expression, which may be due to the limited training data for this expression.



(a) Neutral (b) Happy

(c) Surprise (d) Sad

Figure 4.1: Figure: Some live prediction by my model 2

Chapter 5

Conclusion and Future Works

In conclusion, this research successfully achieved its objectives of developing a facial expression recognition system using two data-sets, comparing the performance of different loss functions on both data-sets, evaluating the system's performance using accuracy and loss on each data-set, and investigating the effect of different hyper-parameters on the system's performance on each data-set. The study found that cross-entropy loss is preferable to L2-Multiclass SVM loss for facial expression recognition, and the proposed deep CNN architectures with small filters showed promising results, outperforming the winning model of the Kaggle competition while reducing the complexity and number of parameters required for the model. Additionally, the study highlights the importance of selecting appropriate hyper-parameters for achieving high accuracy in facial expression recognition tasks. Overall, the research contributes to the development of effective facial expression recognition systems and provides insights into the performance of different loss functions and hyper-parameters on the system's accuracy and loss on multiple data-sets.

As a part of Future Work, I would like to evaluate the system on larger and more diverse data-sets to assess its performance and robustness in different scenarios. This will enable me to identify potential weaknesses and limitations of the system and refine its architecture and training process accordingly. I also planned to explore other deep learning techniques, such as convolutional neural networks with attention mechanisms or recurrent neural networks, to improve the system's accuracy and interpretability. This will enable me to compare the proposed system's performance to other state-of-the-art models in the field and potentially identify novel ways to enhance its effectiveness. I plan to incorporate additional features like voice recognition, or physiological signals to further enhance the system's performance and ability to detect subtle changes in facial expressions and Also, to make a complete model for providing ethinicity of a person.

References

1. Md. Mohsin Kabir et al. “A CNN-LSTM approach to facial expression recognition.” 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE).

2. Dinh Viet Sang et al. “Facial expression recognition using deep convolutional neural networks.” 2016 3rd National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS).

3. Zhang et al. “Facial expression recognition based on ResNet and attention mechanism.” Journal of Electronic Imaging, vol. 29, no. 4, 2020.

4. Omkar M. Parkhi, et al. “A comparative study of CNNs and RNNs for facial expression recognition.” 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI).

5. Sulaiman Muhammad. “Real-time emotion-based music player using facial expression recognition.” 2018 International Conference on Computer and Information Sciences (ICCIS).

6. Jonitta Meryl C, Dharshini K, Sujitha Juliet D Akila Rosy J and Sneha Sara Jacob. “Deep Learning based Facial Expression Recognition for Psychological Health Analysis.” International Journal of Computer Applications, vol. 182, no. 49, pp. 23-27, 2018.

7. Dinh Viet Sang, Nguyen Van Dat, and Do Phan Thuan. ”Facial Expression Recognition Using Deep Convolutional Neural Networks.” 2017 9th International Conference on Knowledge and Systems Engineering (KSE).

8. Wenyi Zhao et al. ”Facial expression recognition based on convolutional neural network and feature fusion.” IEEE Access, vol. 8, pp. 22747-22757, 2020.

9. Khoa Luu et al. ”Facial expression recognition using convolutional neural networks and ensemble methods.” 2019 16th International Conference on Control, Automation, Robotics and Vision (ICARCV).

10. Kamal Gupta et al. ”Facial expression recognition using deep convolutional neural network: A review.” 2018 IEEE 5th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON).

11. Michael Valdenaire et al. ”Facial expression recognition using convolutional neural networks: A survey.” Proceedings of the 2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR).

12. Zhenzhong Lan et al. ”Towards pose-invariant face recognition using pairwise network with pose attention.” IEEE Transactions on Multimedia, vol. 22, no. 6, pp. 1491-1502, 2020.

13. Ying Zhang et al. ”Facial expression recognition based on residual attention networks.” Multimedia Tools and Applications, vol. 79, no. 33, pp. 23947-23963, 2020.

14. find more information about deep learning at https://d2l.ai/index.html.

Appendix

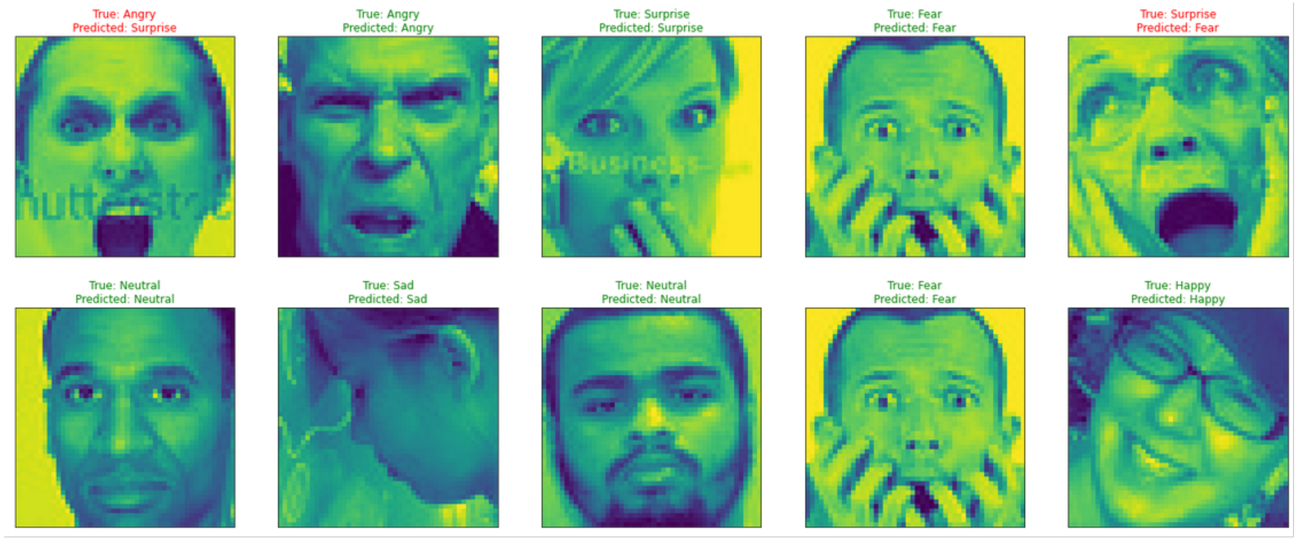


Figure 1 : prediction of test images by model 2 on fer2013 dataset

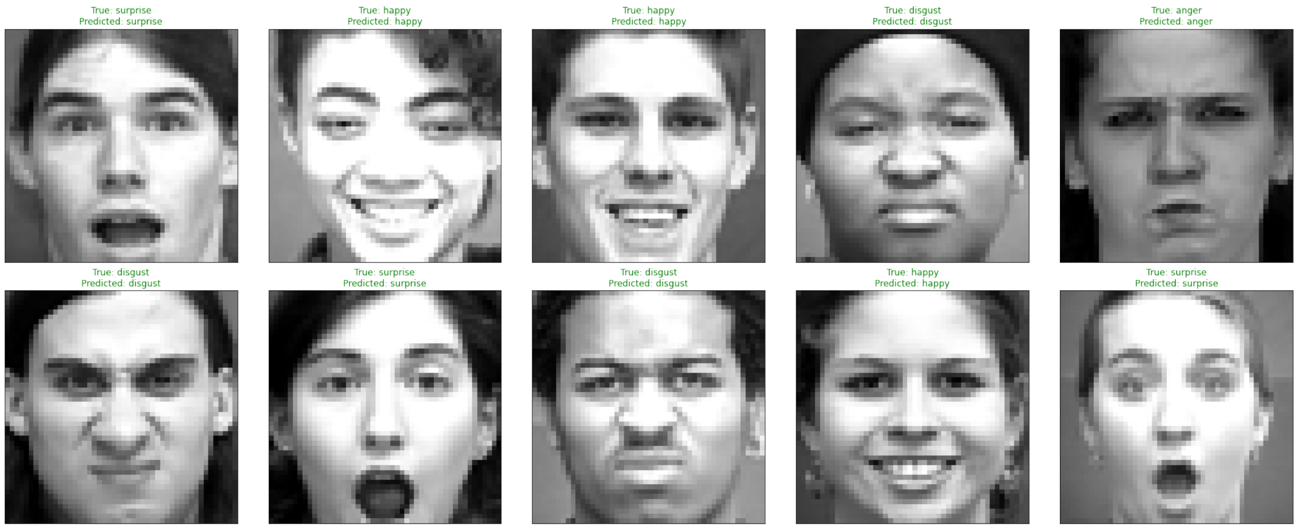


Figure 1 : prediction of test images by model 2 on CK+ dataset