EC6447E DEEP LEARNING IN COMPUTER VISION

PROJECT REPORT

FACIAL EMOTION DETECTOR USING CONVOLUTION NEURAL NETWORK (CNN)

Group Members			
S. No.	**Name**	**Roll No.**	**Email ID**
1.	ANJAL S ANAND	B200895EC	anjal_b200895ec@nitc.ac.in
2.	ANTONY JOY	B200912EC	antony_b200912ec@nitc.ac.in
3.	DAN MANI BINU	B200915EC	dan_b200915ec@nitc.ac.in

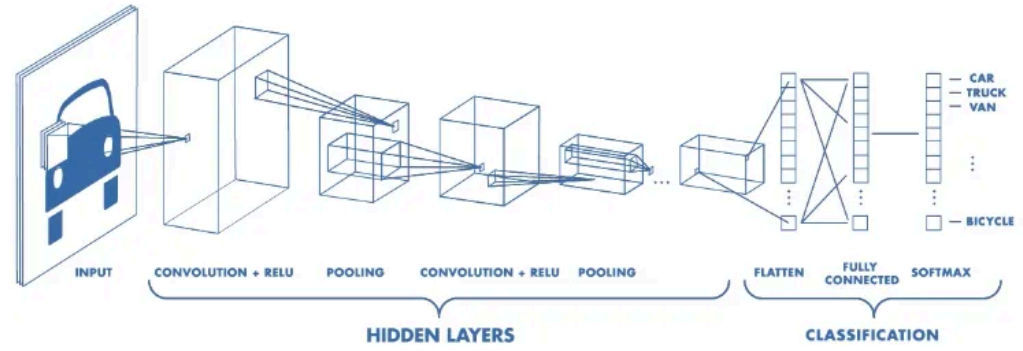
AIM

The aim of this project is to develop and train a Convolutional Neural Network (CNN) model for facial emotion detection using the FER2013 dataset. The primary objective is to achieve high accuracy in recognizing seven different emotions, namely neutral, happy, sad, surprise, fear, disgust, and anger, from facial expressions. To accomplish this, we will design and implement our own CNN architecture, utilising OpenCV for real-time image processing. Our approach involves training the model on a large dataset of labelled images, monitoring its performance during training, and evaluating its accuracy and robustness on a separate test set. We will also analyse various statistical metrics such as accuracy and error per epoch to optimise the hyperparameters and enhance the overall performance of the model. Ultimately, the developed system will be able to accurately recognize emotions in real-time, which has numerous applications in fields like human-computer interaction, affective computing, and mental health diagnosis.

ALGORITHMS DESCRIPTION

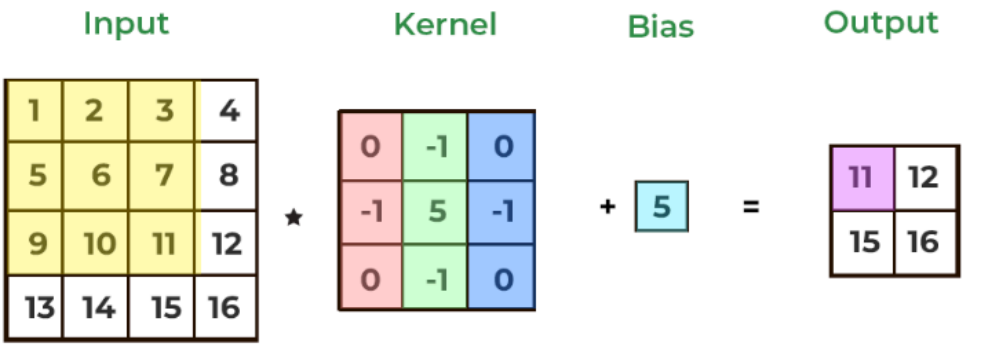
CONVOLUTIONAL NEURAL NETWORKS (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning models particularly suited for tasks like image recognition, object detection, and image classification. They leverage the concept of convolution, where filters are applied to input images to extract relevant features. Their architecture is inspired by the visual cortex of animals, where neurons respond to specific regions of the visual field. CNNs are adept at automatically and adaptively learning spatial hierarchies of features from input data.



Convolutional layers

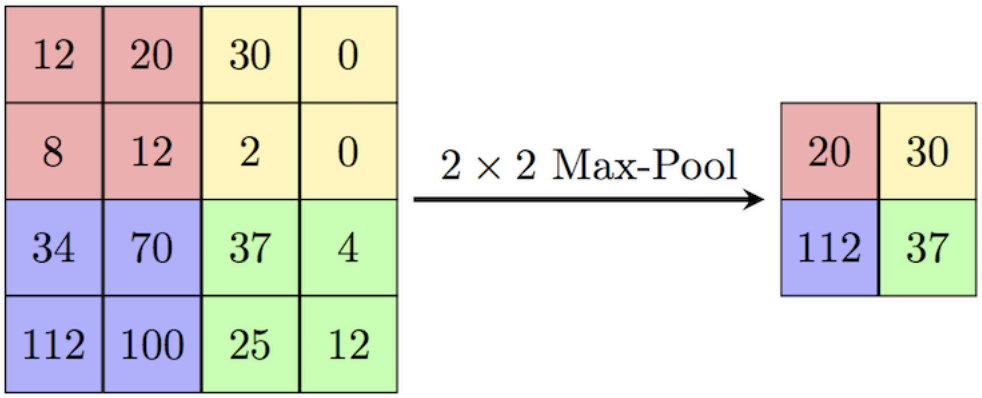
The main layers of a CNN include convolutional layers, pooling layers, and fully connected layers. Convolutional layers form the backbone of CNNs, where filters (also known as kernels) are applied to the input data to extract various features. These filters slide across the input image, performing element-wise multiplication and aggregation to generate feature maps, which highlight relevant patterns and structures. Through the training process, the network learns to detect meaningful features at different spatial locations.



1 X 0 + 5 X -1 + 9 X 0 + 2 X -1 + 6 X 5 + 10 X -1 + 3 X 0 + 7 X -1 + 11 X 0 + 5 = 11

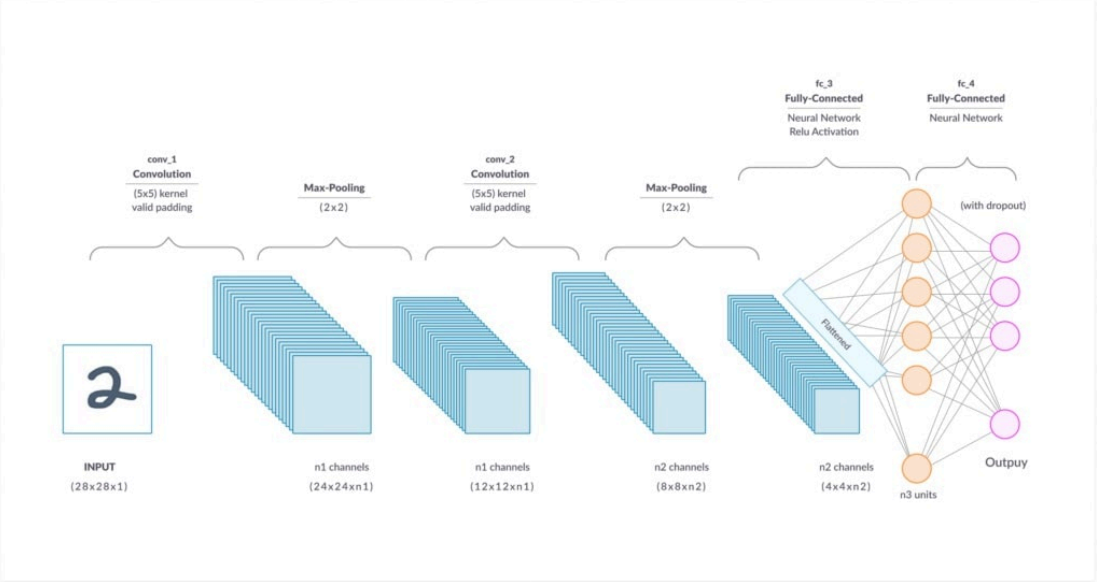
Pooling layers

Pooling layers, often interleaved between convolutional layers, serve to downsample the feature maps, reducing their spatial dimensions while retaining the most salient information. Common pooling operations include max-pooling and average-pooling, which respectively retain the maximum or average value within each pooling region. Pooling helps to make the learned features more invariant to translations and distortions in the input data, improving the model's robustness and reducing computational complexity.



Fully connected layers

Fully connected layers, typically located towards the end of the network, integrate the high-level features learned by convolutional and pooling layers for classification or regression tasks. These layers connect every neuron in one layer to every neuron in the subsequent layer, allowing the network to learn complex decision boundaries and relationships between features. During training, the network adjusts the parameters of these fully connected layers through backpropagation, optimizing them to minimize the loss function and improve performance on the task at hand.

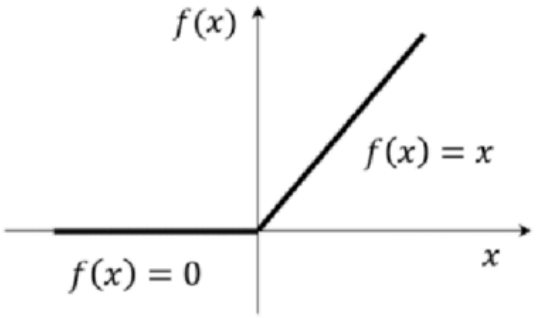


CNNs excel in learning hierarchical representations of visual data. Lower layers of the network learn basic features like edges, textures, and simple shapes, while higher layers learn increasingly complex patterns and object representations by combining features

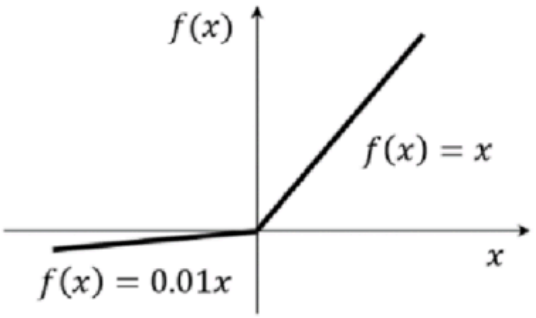
learned from lower layers. This hierarchical feature learning enables CNNs to capture intricate spatial relationships and abstract concepts present in images, facilitating tasks such as object recognition, image segmentation, and scene understanding.

Leaky ReLU

Leaky ReLU (Rectified Linear Unit) is an activation function used in neural networks to introduce a small slope to negative values, allowing some information to pass through even for negative inputs, unlike traditional ReLU which completely blocks negative inputs. This small slope helps alleviate the "dying ReLU" problem where neurons can become inactive during training.. Its application in CNNs contributes to more robust and effective feature learning, leading to better generalization and model performance, particularly in tasks like image classification, object detection, and segmentation.



ReLU activation function



LeakyReLU activation function

Dropout

Dropout is a regularization technique used in neural networks to prevent overfitting and improve the generalization of the model. It works by randomly dropping out (i.e., setting to zero) a fraction of the input units (neurons) during training. This encourages the network to learn more robust features by preventing co-adaptation of neurons.Dropout has found wide application in various types of neural networks, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and feedforward neural networks, and it helps improve the model's ability to generalise to unseen data, leading to better performance and more robust models.

Batch normalization

Batch normalization is a technique used in neural networks to normalize the input of each layer by adjusting and scaling the activations, which improves training speed, stability, and performance. It addresses issues like internal covariate shift by stabilizing activations, allowing networks to converge faster and making them less sensitive to weight initialization. Batch normalization acts as a form of regularization, adding noise to activations to prevent

overfitting. It finds widespread application in various neural network architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and deep learning models for computer vision, natural language processing, and other tasks, contributing to more efficient and stable training processes and improved model performance.

FER2013 DATASET

The dataset used in this project is the FER2013 dataset, which is a widely used benchmark dataset for facial expression recognition. The dataset contains 35,874 images of faces expressing one of seven emotions - neutral, happy, sad, surprise, fear, disgust, and anger. The dataset includes images of 105 individuals (56 females and 49 males), with varying ages, poses, and illumination conditions. Each individual has been photographed under four different lighting conditions, resulting in a total of 420 images per person. The images are in grey scale and have a resolution of 48x48 pixels. The FER2013 dataset is split into a training set of 27,674 images and a test set of 8,200 images. This allows for a thorough evaluation of the model's performance on unseen data, ensuring that the model is not overfitting to the training data. The FER2013 dataset is annotated with emotion labels for each image, indicating the emotion expressed by the individual in the image. The annotations were performed manually by human raters, ensuring high accuracy and consistency in the labelling process. Overall, the FER2013 dataset provided a suitable platform for developing and evaluating our facial expression recognition model. Its large size, diversity, and manual annotation made it an ideal choice for training and testing our model.

CNN architecture

REFERENCE MODEL:

We considered the following model as our baseline. Upon training it with the FER2013 dataset, the validation accuracy of the model was determined to be 58 percent. Subsequently, we attempted to enhance the model's performance by incorporating additional layers and augmenting dropout rates to introduce more randomness.

```
from keras.optimizers import Adam,SGD,RMSprop
no_of_classes = 7
model = Sequential()

#1st CNN layer
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#2nd CNN layer
model.add(Conv2D(128,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#3rd CNN layer
model.add(Conv2D(512,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#4th CNN layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(no_of_classes, activation='softmax'))
opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
214/214 [==============================] - 29s 136ms/step - loss: 0.9372 - accuracy: 0.6409 - val_loss: 1.1958 - val_accuracy: 0.5586
Epoch 10/15
214/214 [==============================] - 30s 140ms/step - loss: 0.8963 - accuracy: 0.6586 - val_loss: 1.2049 - val_accuracy: 0.5638
Epoch 11/15
214/214 [==============================] - 30s 142ms/step - loss: 0.8545 - accuracy: 0.6728 - val_loss: 1.0832 - val_accuracy: 0.5916
Epoch 12/15
214/214 [==============================] - 30s 138ms/step - loss: 0.8111 - accuracy: 0.6925 - val_loss: 1.2282 - val_accuracy: 0.5538
Epoch 13/15
214/214 [==============================] - 29s 136ms/step - loss: 0.7697 - accuracy: 0.7083 - val_loss: 1.2004 - val_accuracy: 0.5541
Epoch 14/15
214/214 [==============================] - 30s 140ms/step - loss: 0.7332 - accuracy: 0.7235 - val_loss: 1.1660 - val_accuracy: 0.5868
Epoch 15/15
214/214 [==============================] - 28s 132ms/step - loss: 0.6813 - accuracy: 0.7468 - val_loss: 1.2056 - val_accuracy: 0.5828
```

OUR MODEL:

```
no_of_classes = 7

model = Sequential()

#1st CNN layer
model.add(Conv2D(32,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.3))

#2nd CNN layer
model.add(Conv2D(64,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.3))

#3rd CNN layer
model.add(Conv2D(128,(3,3),padding = 'same'))
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout (0.3))

#4th CNN layer
model.add(Conv2D(256,(5,5), padding='same'))
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

#5th CNN layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(LeakyReLU(alpha = 0.01))
model.add(Dropout(0.3))

model.add(Flatten())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))

# Fully connected  2nd layer
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(no_of_classes, activation='softmax'))

#Optimizer
opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

1. **Input Layer**: Input images are of size 48x48 pixels with 1 channel (grayscale).

2. **1st CNN Layer**: Convolutional layer with 32 filters of size 3x3, followed by Batch Normalisation, Leaky ReLU activation (alpha=0.01), max-pooling (2x2), and dropout (30%).

3. **2nd CNN Layer**: Convolutional layer with 64 filters of size 5x5, followed by Batch Normalisation, Leaky ReLU activation (alpha=0.01), max-pooling (2x2), and dropout (30%).

4. **3rd CNN Layer**: Convolutional layer with 128 filters of size 3x3, followed by Leaky ReLU activation (alpha=0.01) , max-pooling (2x2), and dropout (30%).

5. **4th CNN Layer**: Convolutional layer with 256 filters of size 5x5, followed by Leaky ReLU activation (alpha=0.01), max-pooling (2x2), and dropout (30%).

6. **5th CNN Laye**r: Convolutional layer with 512 filters of size 3x3, followed by Leaky ReLU activation (alpha=0.01), and dropout (30%).

7. **Flatten Layer:** Flatten layer to convert the 2D feature maps into a 1D vector.

8. **Fully Connected 1st Laye**r: Dense layer with 256 neurons, followed by Batch Normalisation, ReLU activation.

9. **Fully Connected 2nd Laye**r: Dense layer with 512 neurons, followed by ReLU activation, and dropout (30%).

10. **Output Layer**: Dense layer with 7 neurons (number of classes), using softmax activation for multi-class classification.

11. **Optimizer**: Adam optimizer with a learning rate of 0.0001.

12. **Loss Function**: Categorical Cross Entropy loss function used for all layers.

The model utilizes CNNs, which are well-suited for image recognition tasks like emotion detection. CNNs can effectively extract features from images, such as edges, shapes, and textures, which are crucial for recognizing facial expressions associated with different emotions.

The model employs multiple convolutional layers with increasing filter sizes (3x3, 5x5, 3x3, 5x5) and depths (32, 64, 128, 256, 512 filters). This allows the model to progressively extract low-level to high-level features from the images, capturing both basic facial features and their intricate combinations relevant to emotion recognition.

Leaky ReLU activation is used in most convolutional layers, helping to address the vanishing gradient problem and improve training efficiency.

ReLU activation in the fully connected layers introduces non-linearity, allowing the model to learn complex relationships between extracted features and emotions. Batch Normalization helps stabilise the training process and improve convergence. Dropout prevents overfitting by randomly dropping a certain percentage of neurons during training, encouraging the model to learn more robust features.

Adam optimizer is a popular choice for its efficiency and effectiveness in optimising the model's parameters. Categorical Cross Entropy is a suitable loss function for multi-class classification tasks like emotion detection.

Code:

Importing Libraries

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os

# Importing Deep Learning Libraries

from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalization,Activation,M
from keras.models import Model,Sequential
from keras.optimizers import Adam,SGD
```

Displaying Sample images

```
folder_path = "../input/face-expression-recognition-dataset/images/"
expression = 'happy'
picture_size = 48
plt.figure(figsize= (12,12))
for i in range(1, 10, 1):
    plt.subplot(3,3,i)
    img = load_img(folder_path+"train/"+expression+"/"+
                   os.listdir(folder_path + "train/" + expression)[i], target_size=(picture_size, picture_size))
    plt.imshow(img)
plt.show()
```

Training and Validation Data Generator

```
batch_size  = 128

datagen_train  = ImageDataGenerator()
datagen_val = ImageDataGenerator()

train_set = datagen_train.flow_from_directory(folder_path+"train",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=True)


test_set = datagen_val.flow_from_directory(folder_path+"validation",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=False)
```

```
Found 28821 images belonging to 7 classes.
Found 7066 images belonging to 7 classes.
```

This code initialises data generators for training and validation data using Keras's ImageDataGenerator class. Images are loaded directly from directories using the flow_from_directory() method, with preprocessing and augmentation options specified. For both training and validation sets, the directory paths, target image size, colour mode (grayscale), batch size (128), class mode (categorical), and shuffling settings are defined. The train_set and test_set are then created as iterators, facilitating the generation of batches of training and validation data, respectively, suitable for model training and evaluation.

Model Building:

```
no_of_classes = 7

model = Sequential()

#1st CNN layer
model.add(Conv2D(32,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.3))

#2nd CNN layer
model.add(Conv2D(64,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.3))

#3rd CNN layer
model.add(Conv2D(128,(3,3),padding = 'same'))
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout (0.3))

#4th CNN layer
model.add(Conv2D(256,(5,5), padding='same'))
model.add(LeakyReLU(alpha = 0.01))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

#5th CNN layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(LeakyReLU(alpha = 0.01))
model.add(Dropout(0.3))

model.add(Flatten())

#Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))

# Fully connected  2nd layer
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(no_of_classes, activation='softmax'))

#Optimizer
opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

This code defines a Convolutional Neural Network (CNN) architecture for this project on FER2013 dataset. it consists of five convolutional layers with Leaky ReLU activation and dropout for feature extraction, followed by two fully connected layers for classification. Batch normalisation is applied to stabilise and accelerate training. The model is compiled using the Adam optimizer with a learning rate of 0.0001 and categorical cross entropy loss function. The architecture is designed to learn hierarchical features from input images and classify them into one of seven emotion classes.

Model Training

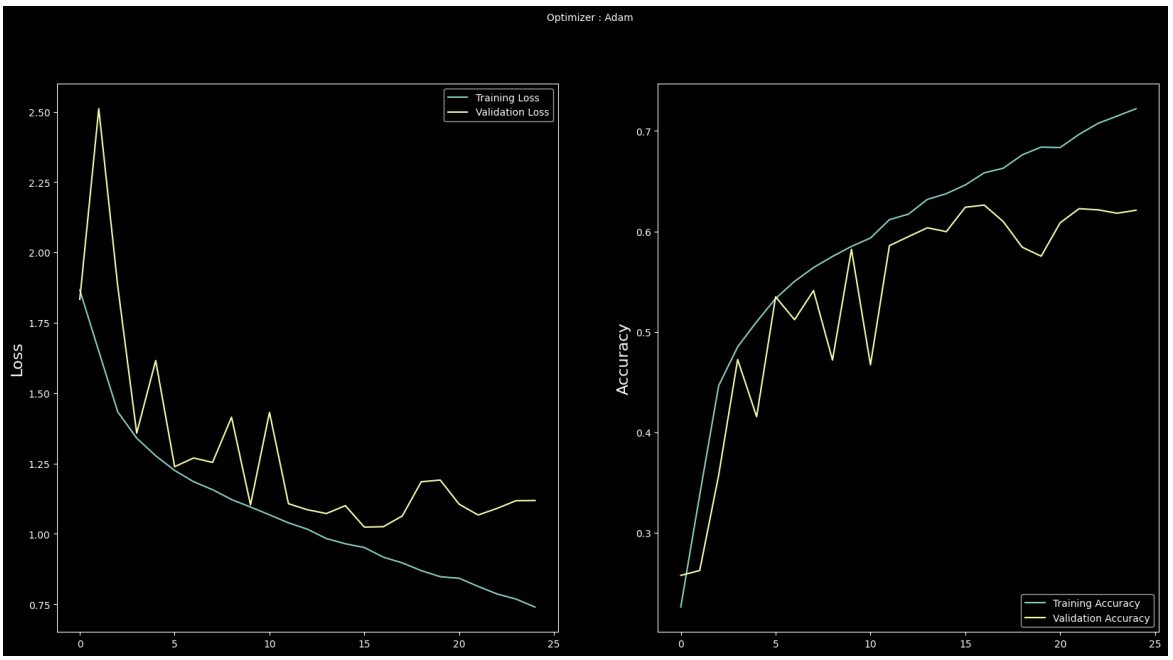
```
epochs =25
history = model.fit_generator(generator=train_set,
                              steps_per_epoch=train_set.n//train_set.batch_size,
                              epochs=epochs,
                              validation_data = test_set,
                              validation_steps = test_set.n//test_set.batch_size
                              )
```

```
Epoch 20/25
225/225 [==============================] - 60s 266ms/step - loss: 0.8480 - accuracy: 0.6839 - val_loss: 1.1914 - val_accuracy:
0.5753
Epoch 21/25
225/225 [==============================] - 54s 242ms/step - loss: 0.8423 - accuracy: 0.6834 - val_loss: 1.1054 - val_accuracy:
0.6085
Epoch 22/25
225/225 [==============================] - 54s 242ms/step - loss: 0.8134 - accuracy: 0.6966 - val_loss: 1.0672 - val_accuracy:
0.6226
Epoch 23/25
225/225 [==============================] - 53s 236ms/step - loss: 0.7866 - accuracy: 0.7075 - val_loss: 1.0907 - val_accuracy:
0.6214
Epoch 24/25
225/225 [==============================] - 54s 242ms/step - loss: 0.7683 - accuracy: 0.7147 - val_loss: 1.1181 - val_accuracy:
0.6180
Epoch 25/25
225/225 [==============================] - 53s 236ms/step - loss: 0.7400 - accuracy: 0.7221 - val_loss: 1.1187 - val_accuracy:
0.6210
```

This code trains the defined Convolutional Neural Network (CNN) model using the training data generator `train_set` for 25 epochs . The training progress is monitored, and the model's performance is evaluated using the validation data generator `test_set`. The `fit_generator` function iterates over the training data generator for the specified number of epochs, updating the model's parameters to minimise the categorical cross entropy loss. The training history, including loss and accuracy metrics for both training and validation sets across epochs, is stored in the `history` object for further analysis and visualisation.

RESULTS:

Accuracy and Loss Curves

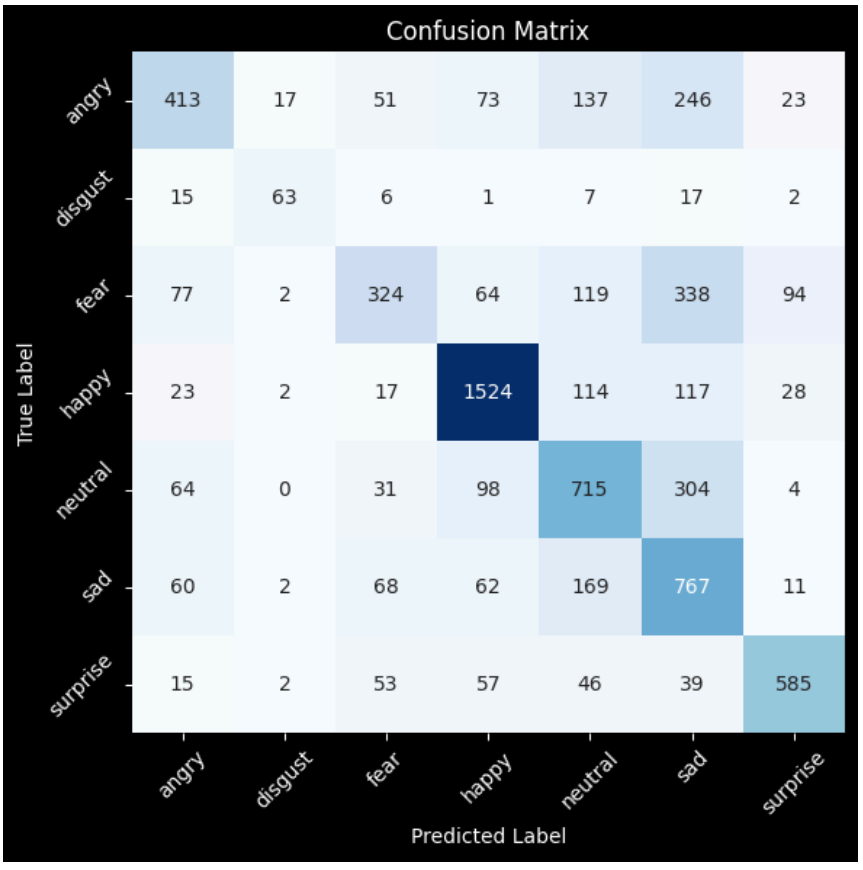


The training algorithm introduces randomness, like shuffling the data order and using dropout layers. This randomness can lead to slight variations in the model's behavior even when trained on the same data, potentially causing some "zig-zags" in the training curve.

The training was terminated after 25 epochs to prevent overfitting. This decision was made because the validation loss, which indicates the model's performance on unseen data, started to increase slightly.

Confusion Matrix

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Predict labels for the test set
y_pred = model.predict(test_set)
y_pred_classes = np.argmax(y_pred, axis=1)
true_classes = test_set.classes

# Calculate confusion matrix
conf_matrix = confusion_matrix(true_classes, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(6,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=test_set.class_indices.keys(),
            yticklabels=test_set.class_indices.keys())
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.tight_layout()
plt.show()
```



While the model performs well overall, the confusion matrix highlights particularly high accuracy in detecting happy, neutral, surprised, and sad emotions compared to others.

These emotions might have more distinct and easily recognizable facial expressions compared to others. For example, happiness often involves a raised mouth corner and smiling eyes, while sadness involves furrowed brows and downturned mouth corners. These features might be easier for the model to learn and classify accurately.

CK+ Dataset:

The CK+ (Extended Cohn-Kanade) dataset is a widely utilized benchmark dataset in the realm of facial expression recognition, comprising expressions such as anger, disgust, fear, happiness, sadness, surprise, and a neutral expression, resulting in a total of seven classes. It consists of 788 images for training and 193 images for testing. However, compared to larger datasets like FER2013, CK+ is relatively small. This limitation may hinder its performance in real-time face recognition applications, as the smaller dataset may not capture the diversity and complexity of real-world scenarios as comprehensively as larger datasets do.

Model Building:

Since the CK+ dataset is significantly smaller than FER2013, a less deep model architecture is likely to be more effective for emotion detection on this dataset. To address this, a new model with the following definition has been created:

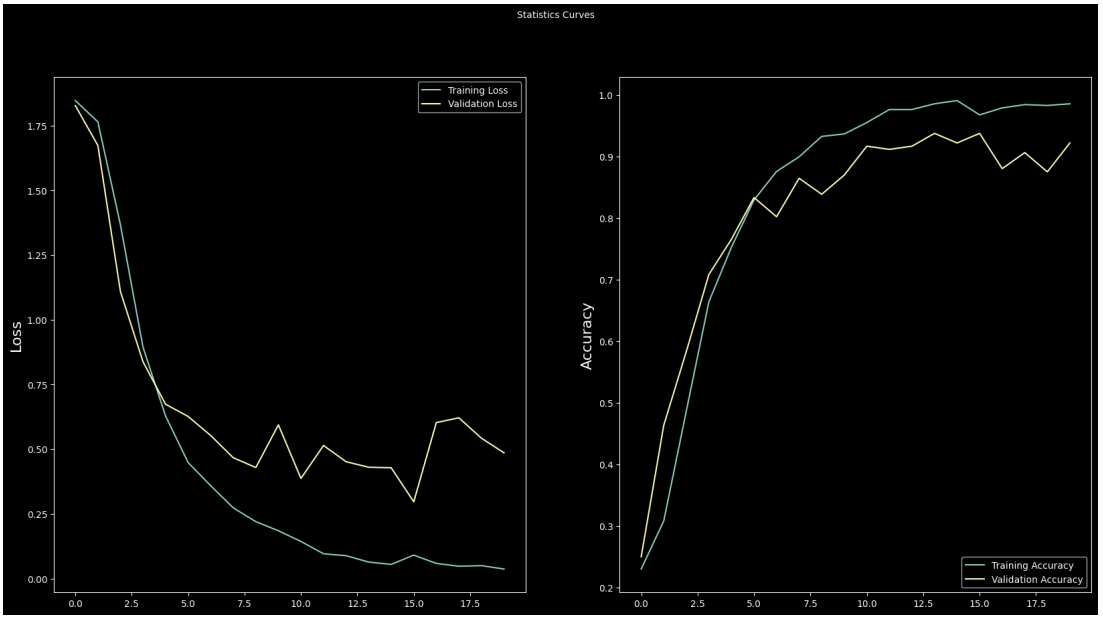
```
num_classes=7
# Define the CNN model
model = Sequential()
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(img_width, img_height, 1)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (5,5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Dropout(0.25))
model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))

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

```python
epochs=20
history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)
```

```
Epoch 15/20
24/24 [==============================] - 2s 84ms/step - loss: 0.0546 - accuracy: 0.9907 - val_loss: 0.4280 - val_accuracy: 0.9219
Epoch 16/20
24/24 [==============================] - 2s 84ms/step - loss: 0.0905 - accuracy: 0.9674 - val_loss: 0.2964 - val_accuracy: 0.9375
Epoch 17/20
24/24 [==============================] - 2s 84ms/step - loss: 0.0584 - accuracy: 0.9788 - val_loss: 0.6025 - val_accuracy: 0.8802
Epoch 18/20
24/24 [==============================] - 2s 84ms/step - loss: 0.0475 - accuracy: 0.9841 - val_loss: 0.6210 - val_accuracy: 0.9062
Epoch 19/20
24/24 [==============================] - 2s 84ms/step - loss: 0.0498 - accuracy: 0.9828 - val_loss: 0.5422 - val_accuracy: 0.8750
Epoch 20/20
24/24 [==============================] - 2s 83ms/step - loss: 0.0368 - accuracy: 0.9854 - val_loss: 0.4857 - val_accuracy: 0.9219
```

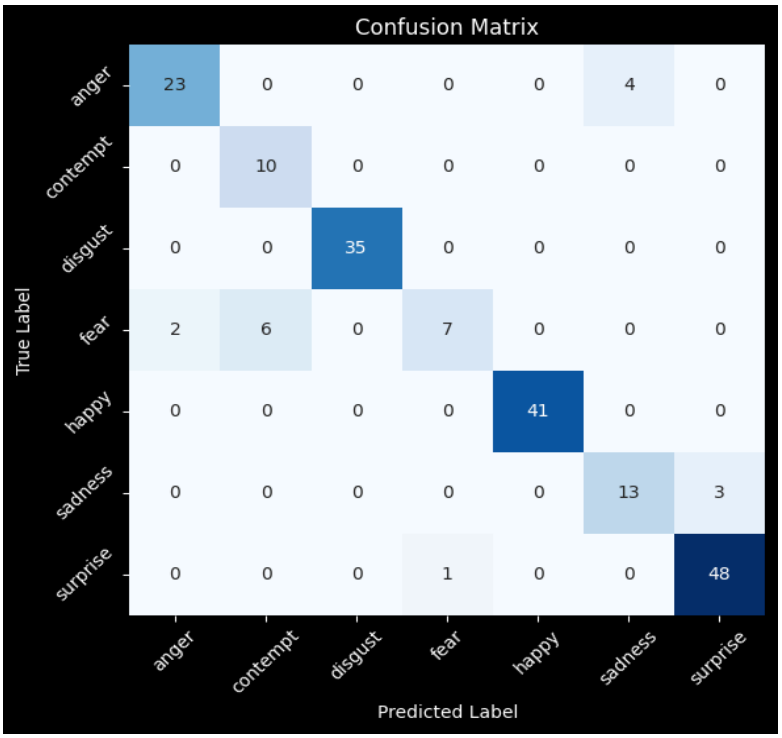
RESULTS:

Accuracy and Error Curves



Despite being less complex and trained on a smaller dataset, the new model achieves higher accuracy and exhibits smoother learning curves. However, the original model might be more suitable for practical applications due to its broader applicability.

Confusion Matrix:



Similar to the previous model, the second model exhibits superior accuracy for detecting happy, neutral, surprise, and sad emotions. This further strengthens the hypothesis that these emotions might be characterized by more distinct and easily identifiable facial expressions compared to others.

Weights of all kernels of the first convolution layer:

```
[4] from tensorflow.keras.models import load_model
    model = load_model('/content/model.h5')

    first_layer_weights = model.layers[0].get_weights()
    first_layer_weights
```

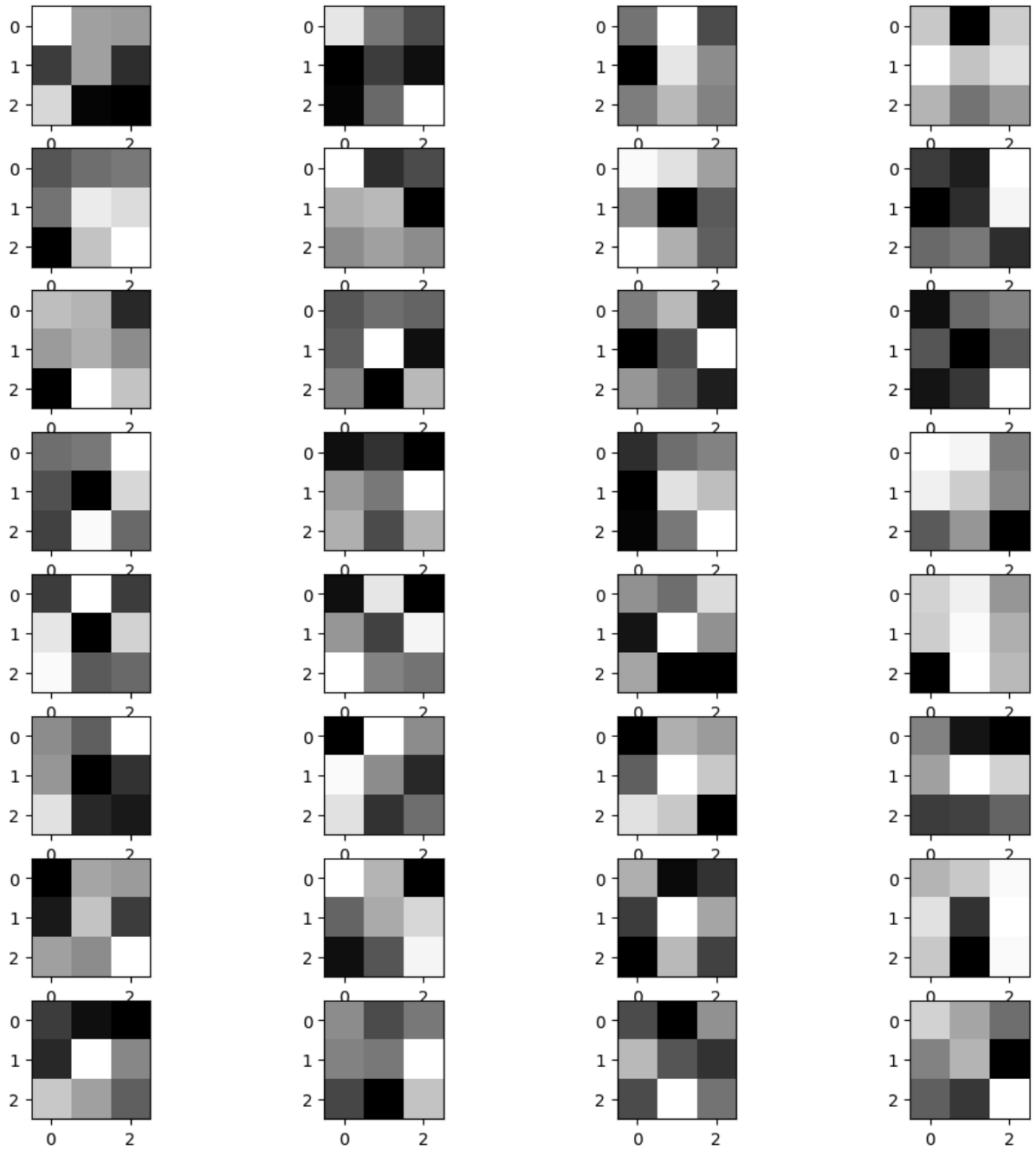
```
[array([[[[ 0.73992157,  0.7559084 , -0.1293708 ,  0.35178536,
           -0.37904197,  0.94729304,  0.5620772 , -0.172637  ,
            0.18389308, -0.11945254,  0.12083405, -0.36506167,
           -0.1941992 , -0.4668567 , -0.34868053,  0.5118035 ,
           -0.42885372, -0.57404286, -0.48118994,  0.1403573 ,
            0.2276827 , -0.78806156, -0.57271516,  0.09912644,
           -0.9226562 ,  0.07324341,  0.37675673, -0.00216397,
           -0.02999357,  0.08492657, -0.20394923,  0.54957485]],

         [[ 0.20877646,  0.13953397,  0.6682493 , -0.8412908 ,
           -0.2460221 , -0.62528694,  0.42319936, -0.2959723 ,
            0.14762104,  0.00971394,  0.5163661 ,  0.1149314 ,
           -0.10564312, -0.30538067, -0.05512764,  0.47591424,
            0.60692495,  0.44340363, -0.6806576 ,  0.31835434,
           -0.01658345,  0.688999  ,  0.12722354, -0.48647475,
            0.17995146, -0.27465487, -0.5354761 ,  0.14142251,
           -0.2532412 , -0.3089904 , -0.6343348 ,  0.2117281 ]],

         [[ 0.17454316, -0.10162611, -0.3628458 ,  0.40508407,
           -0.19348945, -0.39522627,  0.0150208 ,  0.6214655 ,
           -0.4612914 , -0.04663997, -0.5239273 ,  0.2674465 ,
            0.98339224, -0.5450213 ,  0.04895294, -0.18956648,
           -0.42710292, -0.6453331 , -0.03720325, -0.1960374 ,
            0.8046403 ,  0.02002923,  0.05390206, -0.606255  ,
            0.13795742, -1.1513212 , -0.30301014,  0.4501469 ,
           -0.34709558, -0.04376019,  0.20700857, -0.1796689 ]]],


        [[[-0.35245633, -0.5314821 , -0.7953906 ,  0.68433005,
           -0.20800887,  0.343576  , -0.09013428, -0.42043832,
            0.04099418, -0.0850286 , -0.7218444 ,  0.0068861 ,
           -0.41462278,  0.20244323, -0.5764549 ,  0.4371731 ,
            0.48794207,  0.08320745, -1.1779866 ,  0.12220801,
            0.2760029 ,  0.6792153 , -0.19747566,  0.2570521 ,
```

```
print("Shape of weights of the first layer:", first_layer_weights[0].shape)
print("Number of filters in the first layer:", first_layer_weights[0].shape[3])
```

```
Shape of weights of the first layer: (3, 3, 1, 32)
Number of filters in the first layer: 32
```

Grayscale Representation of kernels

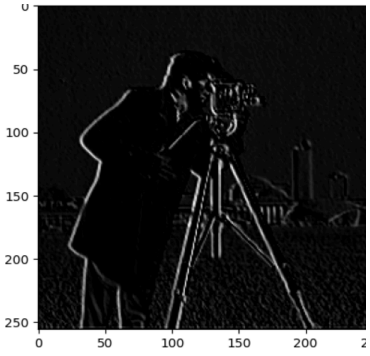
```
import matplotlib.pyplot as plt
plt.figure(figsize= (12,12))
for i in range(32):
    plt.subplot(8,4,i+1)
    plt.imshow(first_layer_weights[0][:, :, 0, i], cmap='gray')
plt.show()
```

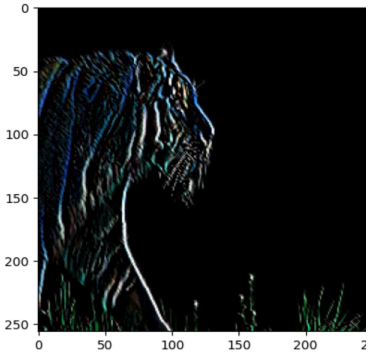


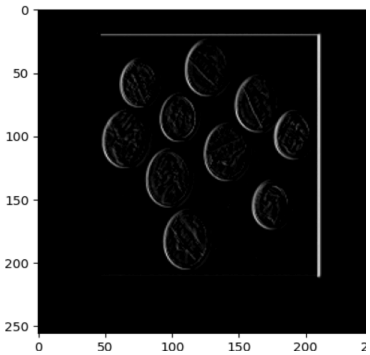
Passing random image through kernel (edge detectors)

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
image = cv2.imread('/content/coin.png')
image = cv2.resize(image, (256, 256))
for i in range(32):
  kernal=first_layer_weights[0][:, :, 0, i]
  # Perform 2D convolution with the random filter
  filtered_image = cv2.filter2D(image, -1, kernal)
  plt.imshow(filtered_image, cmap='gray')
  plt.show()
```

Upon applying the first 32 filters of the model to images of a cameraman, tiger, and coins.jpg, the edges within these images became clearly visible. This suggests that these initial filters are likely responsible for detecting basic image features like edges.

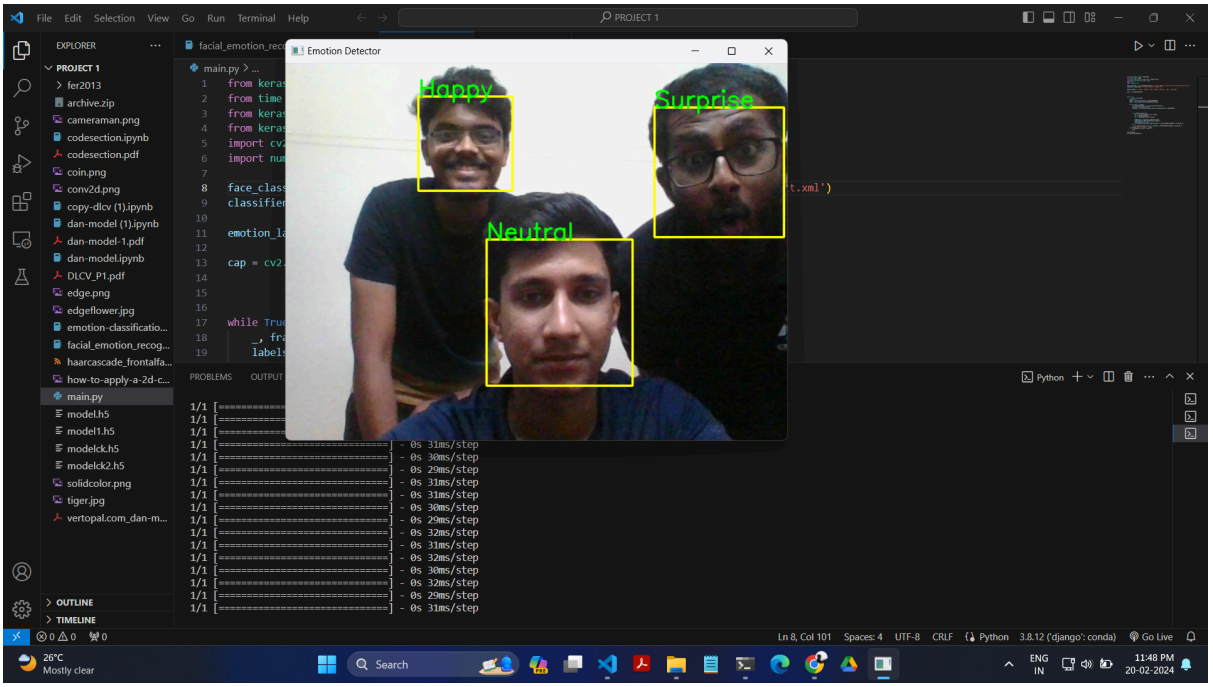


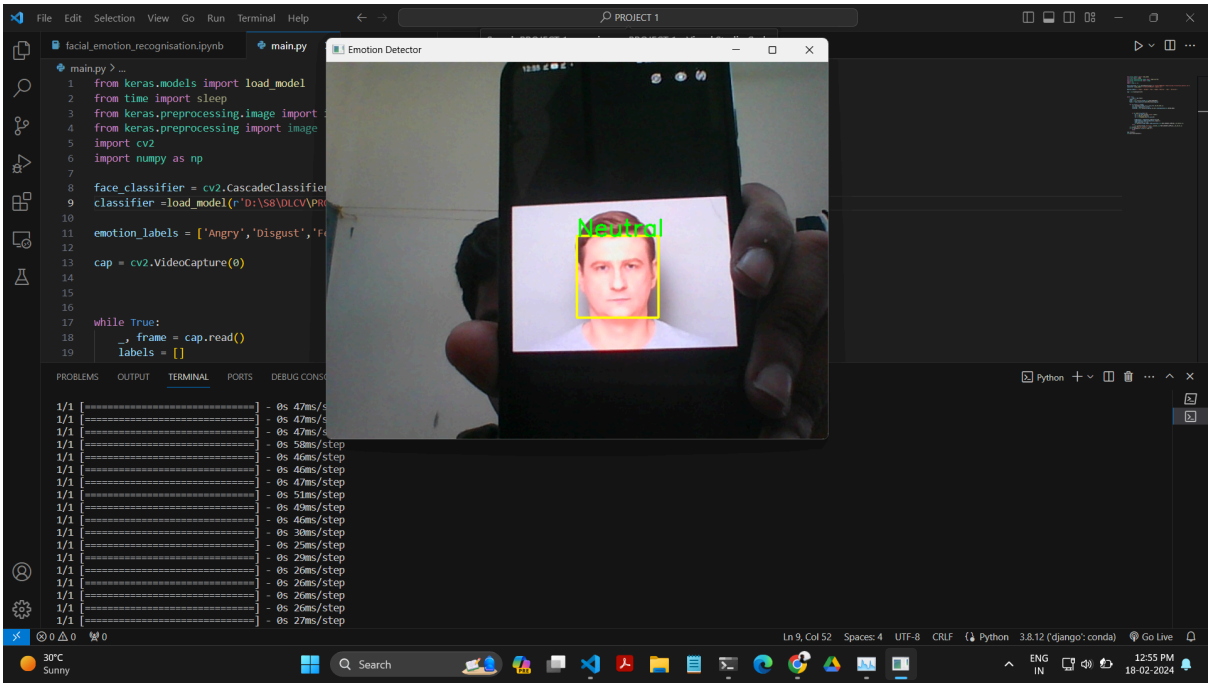


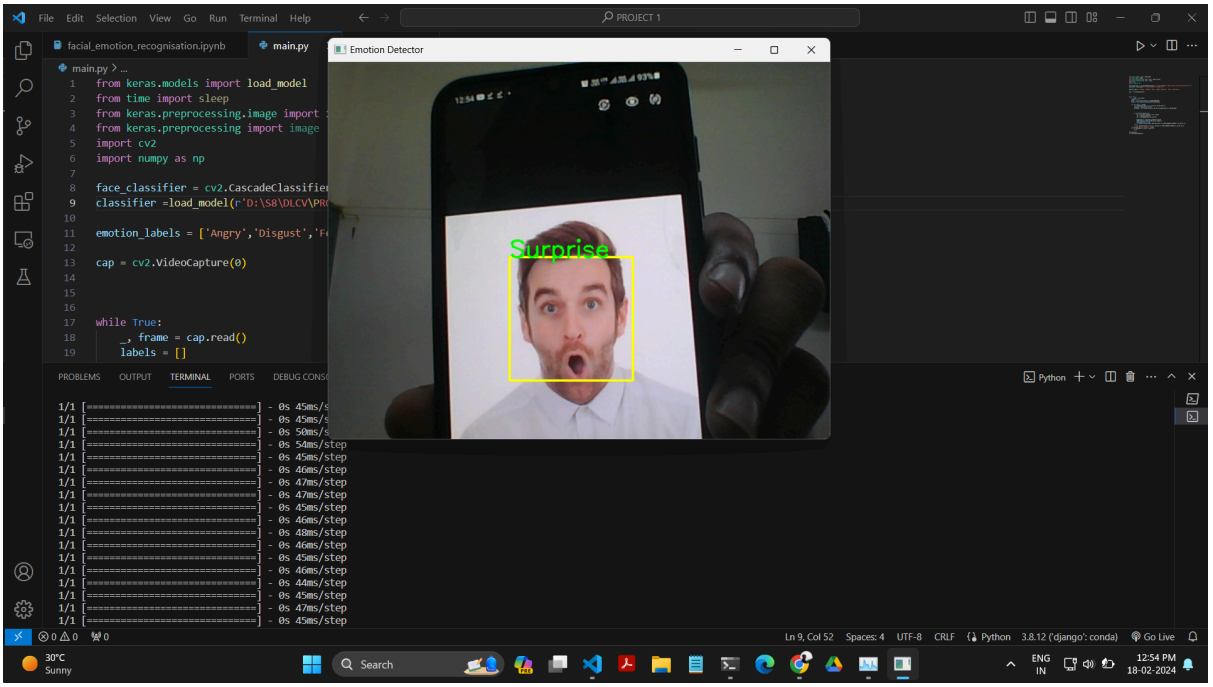


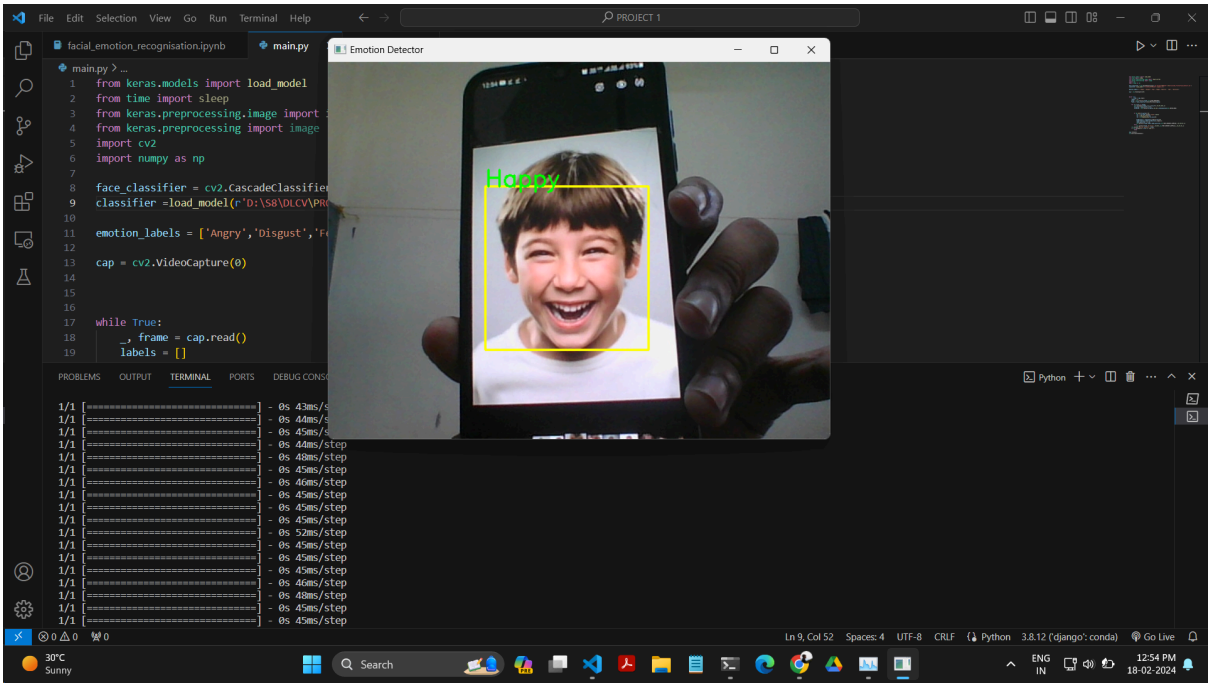
Results:

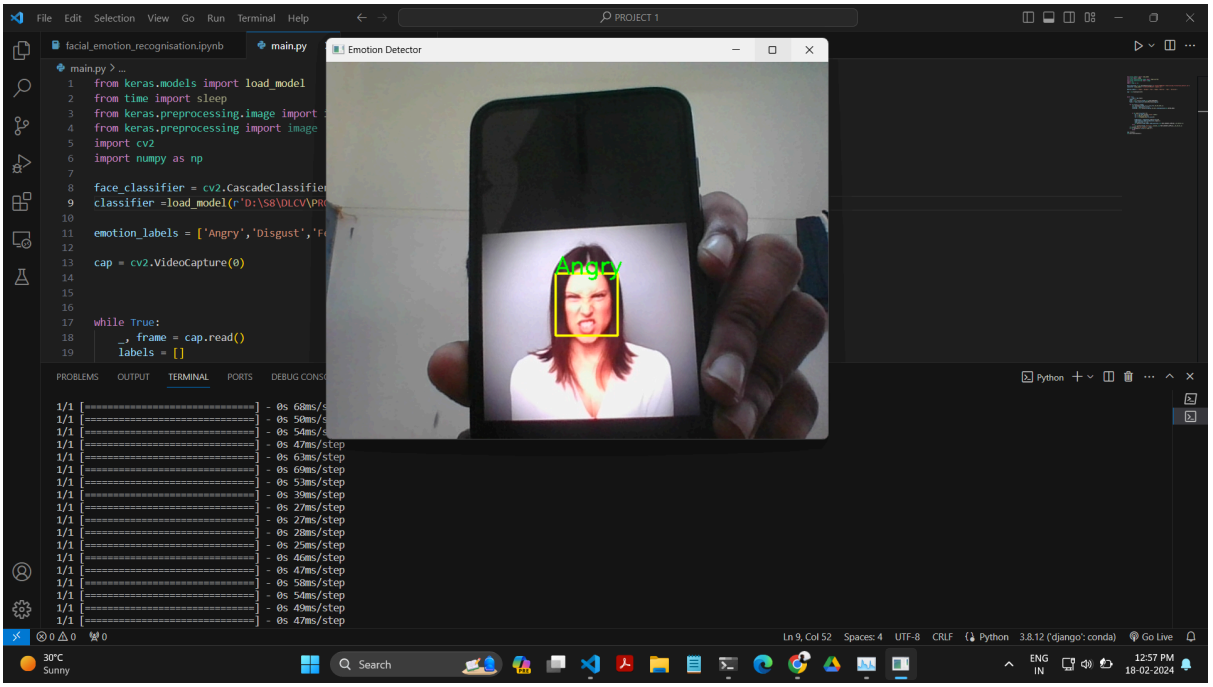
Real time Facial Emotion Recognisation using openCV:






Emotion Detector
Surprise
facial_emotion_recognisation.ipynb
main.py
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import
from keras.preprocessing import image
import cv2
import numpy as np
face_classifier = cv2.CascadeClassifie
classifier =load_model(r'D:\S8\DLCV\PR
emotion_labels = ['Angry','Disgust','F
cap = cv2.VideoCapture(0)
while True:
_, frame = cap.read()
labels = []
PROBLEMS
OUTPUT
TERMINAL
PORTS
1/1 [==========] - 0s 45ms/s
1/1 [==========] - 0s 45ms/s
1/1 [==========] - 0s 50ms/s
1/1 [==========] - 0s 54ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 46ms/step
1/1 [==========] - 0s 47ms/step
1/1 [==========] - 0s 47ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 46ms/step
1/1 [==========] - 0s 48ms/step
1/1 [==========] - 0s 46ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 46ms/step
1/1 [==========] - 0s 44ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 47ms/step
1/1 [==========] - 0s 45ms/step
Ln 9, Col 52
Spaces: 4
UTF-8
CRLF
Python
3.8.12 ('django': conda)
Go Live
30°C
Sunny
Search
ENG
IN
12:54 PM
18-02-2024


Emotion Detector
Happy
facial_emotion_recognisation.ipynb
main.py
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import
from keras.preprocessing import image
import cv2
import numpy as np
face_classifier = cv2.CascadeClassifie
classifier =load_model(r'D:\S8\DLCV\PR
emotion_labels = ['Angry','Disgust','F
cap = cv2.VideoCapture(0)
while True:
_, frame = cap.read()
labels = []
PROBLEMS
OUTPUT
TERMINAL
PORTS
1/1 [==========] - 0s 43ms/s
1/1 [==========] - 0s 44ms/s
1/1 [==========] - 0s 45ms/s
1/1 [==========] - 0s 44ms/step
1/1 [==========] - 0s 48ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 46ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 52ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 46ms/step
1/1 [==========] - 0s 48ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 45ms/step
Ln 9, Col 52
Spaces: 4
UTF-8
CRLF
Python
3.8.12 ('django': conda)
Go Live
30°C
Sunny
Search
ENG
IN
12:54 PM
18-02-2024


Emotion Detector
Angry
main.py
facial_emotion_recognisation.ipynb
PROJECT 1
PROBLEMS
OUTPUT
TERMINAL
PORTS
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 47ms/step
30°C
Sunny
Search
ENG
IN
12:57 PM
18-02-2024