Image Classification by using CNN

Jaladurgam Navya
Department of Computer Science
Kent State University,Kent,

jnavya@kent.edu

***Abstract*— This study focuses on the development and optimization of a Convolutional Neural Network (CNN) model for facial expression recognition using a dataset consisting of 35,887 images. The dataset was collected specifically for this project and includes various facial expressions categorized into different 7 classes which are sad, happy, surprise ,neutral, fear, disgust, angry and second research on a unique dataset created using Teachable Machine, which includes three specific categories: "Touch," "No hands," and "No Touch with Hands" classes in the dataset.**

For these two datasets a Convolutional Neural Network (CNN) model to analyze this dataset. The study evaluates the model's performance by analyzing overall accuracy, constructing confusion matrices, and plotting training/validation curves to visualize accuracy and loss trends.

Keywords—Convolutional neural network, accuracy, training, validation. Max polling .

I. INTRODUCTION

CNNs is particularly good at understanding and analyzing visual information, such as images and videos. They are inspired by how the human brain processes visual data, specifically the part of the brain responsible for vision.

The first dataset comprises 35,887 images specifically collected for facial expression recognition. In the second part of the research, we investigate a unique dataset created using Teachable Machine and for each of these three classes, collected over 300 image examples using interactive interface.

This second dataset provides a unique challenge compared to the first pre-built facial expression dataset. It allows investigating how well CNNs can learn to distinguish these hand gesture classes from training data collected in an interactive, real-time manner.

The primary application of CNNs is facial expression recognition, which is a subfield of computer vision and affective computing. CNNs are well-suited for this task because of their ability to automatically learn and extract complex visual features from image data, including facial features and expressions.

In this study the following evaluation metrics and visualizations for the CNN models trained on the two datasets, confusion matrix, training, and validation curves.

III. BACKGROUND

CNNs are a type of deep neural network architecture specifically designed for efficiently processing and analyzing visual data. In CNN model having a stack of filters with different patterns on them. These filters are used to analyze an image by sliding them across the image, looking for specific patterns or features like edges, curves, or textures. The filters are designed to detect these patterns at different orientations and scales.

As the filters move across the image, they create a new representation of the image, highlighting the areas where the patterns were found. This new representation is then passed through additional layers of filters, each layer looking for more complex patterns and features. To allow the network to learn more complex, non-linear relationships which are very common in visual data, we need to introduce a non-linear component called an activation function. The activation function acts like a gate or a switch. It takes the input from the previous layer and decides how much of that input should be passed on to the next layer based on a non-linear rule. One of the most popular activation functions used in CNNs is called ReLU. Using ReLU for this study.

Batch normalization:

Batch normalization is an important technique that is used in modern deep neural networks like CNN. It is an extra layer that can be added into the network to normalize the inputs going into each following layer. Normalizing the inputs provides several key advantages when training the network. During training the inputs to each layer can change quite a bit as the parameters of the previous layers keep updating. This shifting of input distributions to the layers is called the internal covariate shift problem. Batch normalization helps control this problem by normalizing the inputs to each layer.

Maxpooling:

Maxpooling is used in the model. If the large image that is being analyzed by image network. After the first new layers process this image , we get a map that highlights the important

Features in the images, like edges ,shapes, etc. Maxpooling is a

max pooling is a way to simplify and summarize this feature map. It works by dividing the map into small rectangular sections.

Jupyter notebook: Jupter notebook is an open-source web application that allows you to create, visualizations, data cleaning, data transformation. In this notebook code can write in cells and execute.

Emotion Recognition Data

Emotion Recognition dataset contains faces with expressions. This dataset contains seven different expressions.
Such as sad, happy, fear, disgust ,surprise, neutral, and angry.
This dataset is collected from the internet. Using this dataset we trained a CNN model.

Don't Touch Your Face
We collected this dataset from our class students. This dataset contains three classes , Class 1 Touch, Class 2 No Hands and class 3 No Touch with hands. This data is collected from our class .Every student in the took a 300 above images for every class. The images are arranged and given by the professor.
With this data we trained a CNN model and evaluated the model's performance using metrics like overall accuracy, confusion matrix, training/validation curves of accuracy, and loss.

IV. *OVERVIEW*

Over view Model A
Emotion Recognition Data
The necessary Python modules are imported, including NumPy, Seaborn, Keras and Matplotlib. CNN model architecture suitable for classifying facial expressions from 48x48 pixel grayscale images. The model includes multiple convolutional layers to extract features from the images, followed by fully connected layers to perform the classification task. The use of batch normalization, dropout, and appropriate activation functions helps in training the model effectively.
This model architecture has 4 convolutional layers, 2 fully connected layers. By using this model and 50 epochs got 64.55% accuracy .

Overview Model B
Don't Touch Your Face
Imported the necessary modules and libraries, including Conv2D, MaxPooling2D, Dense, Dropout, Flatten, BatchNormalization, Activation from Keras, Adam optimizer, and ImageDataGenerator for data augmentation.
This CNN architecture is designed for image classification tasks, with an input shape of 48x48x3 the RGB images. The convolutional layers extract features from the input images, and the max-pooling layers down sample the feature maps. The fully connected layers at the end classify the extracted features into the desired number of classes .Used flatten layer. A dense layer with 1024 units and the ReLU activation function, followed by 50% dropout. By using this model and 30 epochs got 99% accuracy.

V. METHODLOGY

Before building a model, I used jupyter notebook in the local system, we need to install python and anaconda into our computer. Alternatively, if you prefer not to configure your local system, you can leverage the free Google Colab Platform. I have written code in jupyter notebook.

STAGE – 1 Data Preprocessing - Model A

a) Facial expression recognition Dataset

Downloaded dataset from Kaggle and extracted. In that dataset contains the train and validation folders. In these two folders having 7 classes of expressions. Retrieving the current working directory with this code.

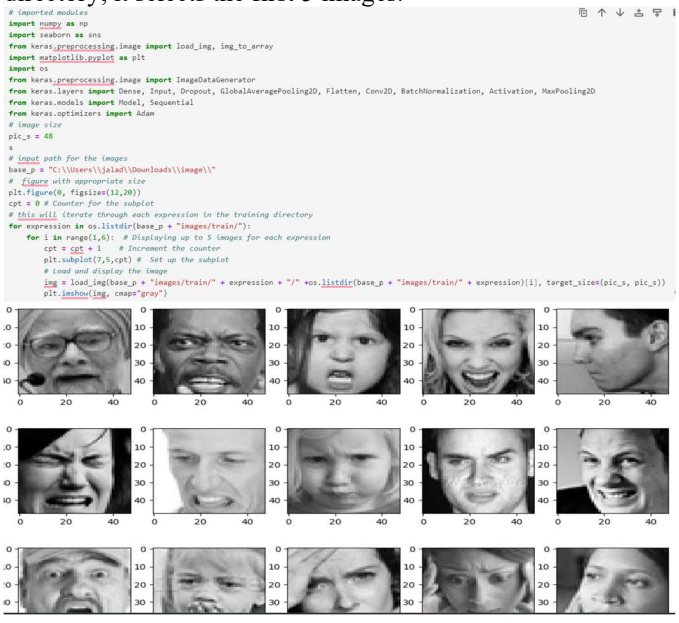
```
import os
# Retrieving the current working directory
os.getcwd()
```

```
'C:\\Users\\jalad\\Downloads'
```

Fig:1 Current working directory

Imported the required modules and sets the image size to 48x48 pixels and specifies the input path for the image dataset.
It creates a figure with a size of 12x20 inches to display the sample images. Initializes a counter variable cpt to keep track of the subplot position. It iterates over the directories in the train folder, where each directory represents a different facial expression e.g., happy, sad, angry. For each expression directory, it selects the first 5 images.

```
# imported modules
import numpy as np
import seaborn as sns
from keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt
import os
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Input, Dropout, GlobalAveragePooling2D, Flatten, Conv2D, BatchNormalization, Activation, MaxPooling2D
from keras.models import Model, Sequential
from keras.optimizers import Adam
# image size
pic_s = 48
s
# input path for the images
base_p = "C:\\Users\\jalad\\Downloads\\image\\"
#  figure with appropriate size
plt.figure(0, figsizes(12,20))
cpt = 0 # Counter for the subplot
# this will iterate through each expression in the training directory
for expression in os.listdir(base_p + "images/train/"):
    for i in range(1,6):  # Displaying up to 5 images for each expression
        cpt = cpt + 1    # Increment the counter
        plt.subplot(7,5,cpt) #  Set up the subplot
        # Load and display the image
        img = load_img(base_p + "images/train/" + expression + "/" +os.listdir(base_p + "images/train/" + expression)[i], target_size(pic_s, pic_s))
        plt.imshow(img, cmap="gray")
```



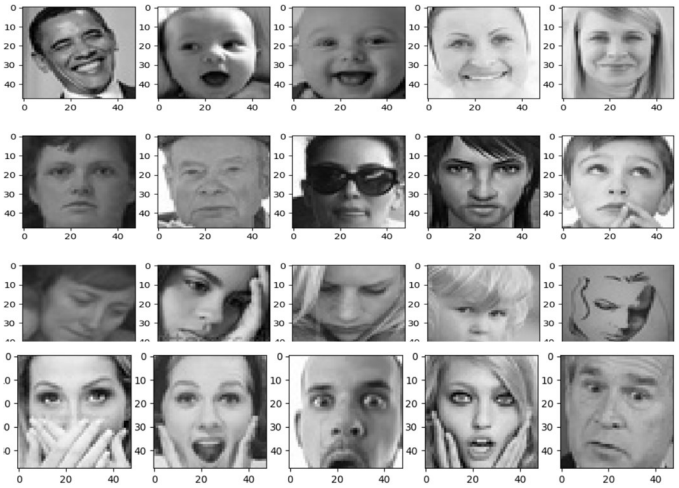


Fig 2: images with 48*48 pixels.

Data generators for the training and validation datasets using the Keras ImageDataGenerator , batch size for training to 128. It initializes two instances of the ImageDataGenerator class: data_train and data_validation. These generators will be used for preprocessing and augmenting the training and validation data. Got 28,821 images in the specified training directory. Found a total of 7,066 images in the specified validation directory.

```
# the batch size for training
batch_size = 128
# ImageDataGenerator for training and validation data initializing
data_train = ImageDataGenerator()
data_validation = ImageDataGenerator()


# data generators for the training set
train_v= data_train.flow_from_directory('C:\\Users\\jalad\\Downloads\\image\\images\\train', # path for the train folder
                                        target_size=(pic_s,pic_s), # target image size
                                        color_mode="grayscale", # Converting images to grayscale
                                        batch_size=batch_size, # Setting batch size
                                        class_mode='categorical', # Using categorical labels
                                        shuffle=True) # Shuffling the data during training
# data generators for the validation set
validation_v = data_validation.flow_from_directory('C:\\Users\\jalad\\Downloads\\image\\images\\validation', # path for the train folder
                                        target_size=(pic_s,pic_s), # target image size
                                        color_mode="grayscale", # Converting images to grayscale
                                        batch_size=batch_size,  # Setting batch size
                                        class_mode='categorical', # Using categorical labels
                                        shuffle=False) #  not shuffling the data for validation
```

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

Fig 3: images in train and validation folders.

STAGE – 1 Data Preprocessing - Model B
Don't Touch Your Face

First, retrieving the directory of the dataset and this dataset given by the professor in that contains the Subject folders fromSubject_1 to Subejct_55.In these each folder there are 3 classes . To prepare the data for training, images from these Subject folders were reorganized and placed into three separate classes or directories within a new folder called "final dataset".

```
# Define the base directory where all folders are located
base_directory = r'C:\Users\jalad\Downloads\Dataset_Project_1'

# Define the destination folders where you want to copy images for each class
destination_folders = {
    "Class 1 Touch": r'C:\Users\jalad\Downloads\final dataset\Class 1 Touch',
    "Class 2 No hands": r'C:\Users\jalad\Downloads\final dataset\Class 2 No hands',
    "Class 3 No Touch w Hands": r'C:\Users\jalad\Downloads\final dataset\Class 3 No Touch w Hands'
}

# Initialize a counter to keep track of the index for each copied file
index = 1

# Iterate over all folders in the base directory
for subject_folder in os.listdir(base_directory):
    subject_path = os.path.join(base_directory, subject_folder)

    # Check if the current item is a directory
    if os.path.isdir(subject_path):
        # Iterate over the destination folders
        for class_folder, destination_folder in destination_folders.items():

            class_folder_path = os.path.join(subject_path, class_folder)
            if os.path.exists(class_folder_path) and os.path.isdir(class_folder_path):
                print(f"Copying images from {class_folder_path} to {destination_folder}")

                for file_name in os.listdir(class_folder_path):
                    source_file_path = os.path.join(class_folder_path, file_name)
                    destination_file_name = f"image{index}.jpg"
                    destination_file_path = os.path.join(destination_folder, destination_file_name)
                    shutil.copyfile(source_file_path, destination_file_path)
                    print(f"Copied {file_name} to {destination_file_name}")
                    index += 1
            else:
                print(f"No '{class_folder}' folder found in {subject_folder}")
    else:
        print(f"{subject_folder} is not a directory")

print("Copying completed.")
```

Splitting a dataset of images into training and testing sets. Defines the base path where the dataset is located and creates separate directories for the training and testing sets if they don't already exist. A list of all image files in the dataset, along with the class folder for each image. Shuffles the list of images randomly. Splitting the list of images into training and testing folders in the ratio of 80 for training and 20 for testing. In entire dataset counts the images in each class this information will print. The copy_images function for both the training and testing folders, which copy the images to their respective folders. Counts the number of images in each class for the training and testing sets separately and prints this information.

```
test_folder = os.path.join(base_path, "Test")

# Create train and test folders if they don't exist
os.makedirs(train_folder, exist_ok=True)
os.makedirs(test_folder, exist_ok=True)

# Get a list of all image files
all_images = []
for class_folder in os.listdir(base_path):
    class_path = os.path.join(base_path, class_folder)
    if os.path.isdir(class_path):
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            all_images.append((img_path, class_folder))

# Shuffle the list of images
random.shuffle(all_images)

# Define ratios for train and test sets
train_ratio = 0.8
test_ratio = 0.2

# Split the list of images into train and test sets
num_images = len(all_images)
num_train = int(train_ratio * num_images)
num_test = num_images - num_train

train_set = all_images[:num_train]
test_set = all_images[num_train:]

# Function to copy images from source to destination
def copy_images(img_set, dest_folder):
    class_counts = {}s
class_counts = {}
for img_path, class_folder in all_images:
    class_counts[class_folder] = class_counts.get(class_folder, 0) + 1

# Print the number of images in each class
print("Number of images in each class:")
for class_name, count in class_counts.items():
    print(f"{class_name}: {count} images")

# Prepare train and test data
train_counts = copy_images(train_set, train_folder)
test_counts = copy_images(test_set, test_folder)

# Print the number of images in each class for train and test sets
print("\nTrain set:")
for class_name, count in train_counts.items():
    print(f"{class_name}: {count} images")

print("\nTest set:")
for class_name, count in test_counts.items():
    print(f"{class_name}: {count} images")
```

Fig 4:This is the output for above code

```
Number of images in each class:
Class 3 No Touch w Hands: 18646 images
Class 2 No hands: 18195 images
Class 1 Touch: 18166 images

Train set:
Class 3 No Touch w Hands: 14930 images
Class 2 No hands: 14571 images
Class 1 Touch: 14504 images

Test set:
Class 1 Touch: 3662 images
Class 2 No hands: 3624 images
Class 3 No Touch w Hands: 3716 images
```

The purpose of the code below is to provide a visual inspection of a sample of images from the training dataset. By displaying the first three images from each class in a 3x3 grid, it allows you to quickly check if the images are loaded correctly and if the dataset is as expected.

```
pic_s = 100

base_p = "C:\\Users\\jalad\\Downloads\\final dataset\\"

plt.figure(0, figsizes(12, 20))
cpt = 0

for expression in os.listdir(base_p + "/train/"):
    for i in range(1, 4):  # Iterate over the first 3 images
        cpt = cpt + 1
        plt.subplot(3, 3, cpt)  # Create a 3x3 grid
        img = load_img(base_p + "/train/" + expression + "/" + os.listdir(base_p + "/train/" + expression)[i], target_sizes(pic_s, pic_s))
        plt.imshow(img, cmaps="gray")
```

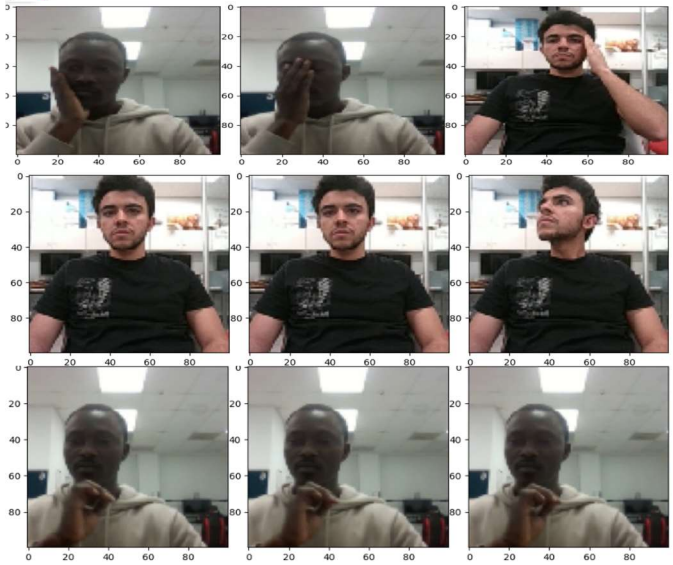


Fig 5: Images of three different classes

Loads the training data and testing data from the specified directory. Specified the target size(112,112) to which all images will be resized. Data will be loaded in batches of 64 samples at a time.

```
]: # batches of augmented data from the training directory
   train_generl= train_datagen.flow_from_directory(
       'C:\\Users\\jalad\\Downloads\\final dataset\\Train',  # Path for the training directory
       target_size=(112, 112), # image size
       batch_size=64,   # batch size
       color_mode="rgb", # color of the image
       class_mode='categorical', # type of label returned
       shuffle=True)


   test_generl= test_datagen.flow_from_directory(
           'C:\\Users\\jalad\\Downloads\\final dataset\\Test',  # Path for the testing directory
           target_size=(112, 112), # image size
           batch_size=64, # batch size
           color_mode="rgb", # color of the image
           class_mode='categorical') # type of label returned
```

```
Found 44005 images belonging to 3 classes.
Found 11002 images belonging to 3 classes.
```

STAGE -2- Building the model A
Facial Expression Recognition

Trains a convolutional neural network model on the facial expression recognition dataset using the Keras library. The initial learning rate for the model to 0.0001. Adam optimizer with the defined exponential decay learning rate schedule . The model is trained for a specified number of epochs, and the trained model architecture and weights are saved .

```
classes = 7

# Initialising the CNN
model8 = Sequential()

# 1 - Convolution
model8.add(Conv2D(64,(3,3), padding='same', input_shape=(48, 48,1)))
model8.add(BatchNormalization())
model8.add(Activation('relu'))
model8.add(MaxPooling2D(pool_size=(2, 2)))
model8.add(Dropout(0.25))

# 2nd Convolution layer
model8.add(Conv2D(128,(5,5), padding='same'))
model8.add(BatchNormalization())
model8.add(Activation('relu'))
model8.add(MaxPooling2D(pool_size=(2, 2)))
model8.add(Dropout(0.25))

# 3rd Convolution layer
model8.add(Conv2D(512,(3,3), padding='same'))
model8.add(BatchNormalization())
model8.add(Activation('relu'))
model8.add(MaxPooling2D(pool_size=(2, 2)))
model8.add(Dropout(0.25))

# 4th Convolution layer
model8.add(Conv2D(512,(3,3), padding='same'))
model8.add(BatchNormalization())
model8.add(Activation('relu'))
model8.add(MaxPooling2D(pool_size=(2, 2)))
model8.add(Dropout(0.25))


model8.add(Flatten())
# Fully connected layer 1st layer
model8.add(Dense(256))
model8.add(BatchNormalization())
model8.add(Activation('relu'))
model8.add(Dropout(0.25))

# Fully connected layer 2nd layer
model8.add(Dense(512))
model8.add(BatchNormalization())
model8.add(Activation('relu'))
model8.add(Dropout(0.25))

model8.add(Dense(classes, activation='softmax'))

opt = Adam(learning_rate=0.0001)
model8.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

Fig:6 Model for emotion recognition

STAGE -2- Building the model B
Don't Touch Your Face

The first layer has 64 filters, 3*3 kernel size and same padding. The input shape is (112,112,3).The BatchNormalization layer and a ReLU activation function. Dropout layer with a rate of 0.25 is added to reduce overfitting. Feature maps are then flattened using Flatten layer. The model is compiled with the categorical cross-entropy loss function, Adam optimizer, and accuracy metric.

```
model_cnn1 = Sequential()

# Increase the number of filters in the first convolutional layer
model_cnn1.add(Conv2D(64, kernel_size=(3, 3), padding='same', strides=(1, 1), input_shape=(112, 112, 3)))
model_cnn1.add(BatchNormalization())
model_cnn1.add(Activation('relu'))

# Add another convolutional layer before the max-pooling layer
model_cnn1.add(Conv2D(64, kernel_size=(3, 3), padding='same', strides=(1, 1)))
model_cnn1.add(BatchNormalization())
model_cnn1.add(Activation('relu'))
model_cnn1.add(MaxPooling2D(pool_size=(2, 2)))
model_cnn1.add(Dropout(0.25))

# Increase the number of filters in the subsequent convolutional layers
model_cnn1.add(Conv2D(128, kernel_size=(3, 3), padding='same', strides=(1, 1)))
model_cnn1.add(BatchNormalization())
model_cnn1.add(Activation('relu'))
model_cnn1.add(MaxPooling2D(pool_size=(2, 2)))

model_cnn1.add(Conv2D(256, kernel_size=(3, 3), padding='same', strides=(1, 1)))
model_cnn1.add(BatchNormalization())
model_cnn1.add(Activation('relu'))
model_cnn1.add(MaxPooling2D(pool_size=(2, 2)))
model_cnn1.add(Dropout(0.25))

model_cnn1.add(Flatten())
model_cnn1.add(Dense(1024, activation='relu'))
model_cnn1.add(Dropout(0.5))
model_cnn1.add(Dense(3, activation='softmax'))

model_cnn1.summary()
```

Fig :7 Model for don't touch your face

STAGE -3- Training the model A

Facial Expression Recognition

Training model for 50 epochs using ModelCheckpoint callback to save the best model weights based on the validation accuracy. *Got 0.6454 validation accuracy.*

```
epochs = 50

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_acc', verbose=1, mode='max')
callbacks = [checkpoint]

history = model8.fit(x=train_v,
                     steps_per_epoch=train_v.n//train_v.batch_size,
                     epochs=epochs,
                     validation_data = validation_v,
                     validation_steps = validation_v.n//validation_v.batch_size,
                     callbacks=callbacks
                     )
```

Fig:8 Training the model

```
225/225 [==============================] - ETA: 0s - loss: 0.6886 - accuracy: 0.7437
Epoch 48: saving model to model_weights.h5
225/225 [==============================] - 111s 494ms/step - loss: 0.6886 - accuracy: 0.7437 - val_loss: 1.0692 - val_accuracy: 0.6416
Epoch 49/50
225/225 [==============================] - ETA: 0s - loss: 0.6757 - accuracy: 0.7457
Epoch 49: saving model to model_weights.h5
225/225 [==============================] - 112s 497ms/step - loss: 0.6757 - accuracy: 0.7457 - val_loss: 1.0607 - val_accuracy: 0.6392
Epoch 50/50
225/225 [==============================] - ETA: 0s - loss: 0.6567 - accuracy: 0.7562
Epoch 50: saving model to model_weights.h5
225/225 [==============================] - 112s 499ms/step - loss: 0.6567 - accuracy: 0.7562 - val_loss: 1.0419 - val_accuracy: 0.6447
```

Fig 9: Accuracy

STAGE -3- Training the model B

Don't touch your face

Training model for 30 epochs using ModelCheckpoint callback to save the best model weights based on the validation accuracy.*The weights are saved in* model_cb.h5 file.

```
epochs = 30 # number of epochs to train

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint("model_cb.h5", monitor='val_acc', verbose=1, mode='max') # # ModelCheckpoint object to save model weights dur
callbackz= [checkpoint] # Generating a callback list that exclusively includes the ModelCheckpoint callback
# Training model8 with the training dataset and validating its performance with the validation dataset.
#The validation_data parameter indicates the dataset used for evaluating the model's performance after each epoch during training.

# The validation_steps parameter indicates the quantity of batches to be processed from the validation dataset in every epoch.
# the callbacks parameter specifies a collection of callback functions to be utilized during training.
history_q = model_cnn1.fit(x=train_generl,
                           steps_per_epoch=train_generl.n//train_generl.batch_size,
                           epochs=epochs,
                           validation_data =test_generl,
                           validation_steps =test_generl.n//test_generl.batch_size,
                           callbacks=callback
                           )
model_cnn1.save("model_cb.h5") # Saving the model to a Hierarchical Data Format

model_cnn1.save("model_cb.keras") # Saving the model to a Keras Sequential model file
```

```
687/687 [==============================] - 1012s 1s/step - loss: 0.0111 - accuracy: 0.9962 - val_loss: 0.0013 - val_accuracy: 0.9994
Epoch 21/30
687/687 [==============================] - ETA: 0s - loss: 0.0104 - accuracy: 0.9964
Epoch 21: saving model to model_cb.h5
687/687 [==============================] - 1011s 1s/step - loss: 0.0104 - accuracy: 0.9964 - val_loss: 0.0017 - val_accuracy: 0.9996
Epoch 22/30
687/687 [==============================] - ETA: 0s - loss: 0.0144 - accuracy: 0.9954
```

Fig : 10 Training the model B

STAGE -4- Evaluation metrics model A

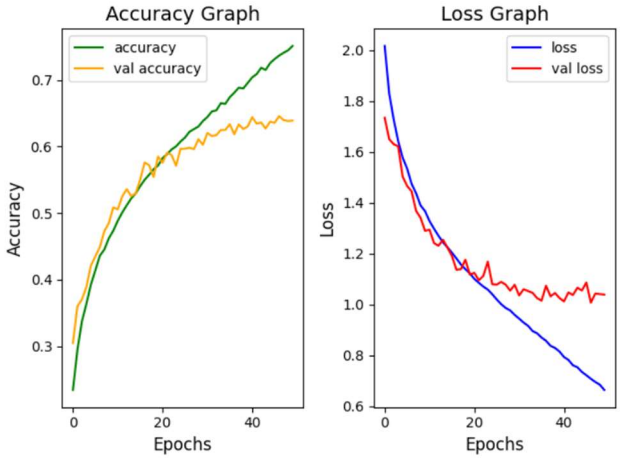
Facial Expression Recognition

Code is to visualize the performance of the deep learning model during training by plotting the training and validation accuracy and loss over the epochs.

```
import matplotlib.pyplot as plt

# Accuracy graph
plt.subplot(1, 2, 1)
plt.plot(accuracy, label='accuracy', color='green')
plt.plot(val_accuracy, label='val accuracy', color='orange')
plt.title('Accuracy Graph', fontsize=14)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.legend(fontsize=10)

# Loss graph
plt.subplot(1, 2, 2)
plt.plot(loss, label='loss', color='blue')
plt.plot(val_loss, label='val loss', color='red')
plt.title('Loss Graph', fontsize=14)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.legend(fontsize=10)

plt.tight_layout()
plt.show()
```

Fig 11: Code for plots

First subplot in a 1x2 grid, for plotting the accuracy graph and validation accuracy over epochs, with the label 'val accuracy'. plots the training loss with a blue color and labels it as 'loss'.It plots the validation loss with a red color and labels it as 'val loss'.




```
#Assess the model's performance on the validation dataset and save the outcomes in the variable named 'evaluation'
evaluation=model8.evaluate(validation_v)
#Display the validation loss retrieved from the evaluation outcomes
print("Validation Loss:", evaluation[0])
# Output the validation accuracy acquired from the evaluation outcomes.
print("Validation Accuracy:", evaluation[1])
```

```
56/56 [==============================] - 6s 112ms/step - loss: 1.0365 - accuracy: 0.6398
Validation Loss: 1.0364824533462524
Validation Accuracy: 0.6398245096206665
```

Fig :12 Code for validation accuracy

Validation Accuracy is 0.63982 for 50 epochs.

Confusion matrix:

Emotion recognition dataset with 7 classes are happy, sad, anger, fear, surprise, disgust, and neutral, the confusion matrix for a CNN model would be a 7x7 matrix. Each row and column in the matrix would represent one of the seven emotion classes. For example, the element at the intersection of the "Happy" row and "Happy" column represents the number of happy instances that were correctly classified as happy .

```python
import itertools
# function to plot confusion matrix
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm.Reds):
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]   # Normalize the confusion matrix
    plt.figure(figsize=(10,10)) # figure size
    plt.imshow(cm, interpolation='nearest', cmap=cmap)  # plotting the confusion matrix
    plt.title(title) # title of the plot
    plt.colorbar() # Add a color bar
    marks = np.arange(len(classes)) # Create markers for class labels
    plt.xticks(marks, classes, rotation=45) # Define the x-axis tick labels using the class names and rotate them for better readability
    plt.yticks(marks, classes) # y-axis tick marks using the class names.

    f = '.2f' # Specify the format for the text displayed in the confusion matrix
    t = cm.max() / 2. # Threshold for text color
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])): #  Iterate over confusion matrix elements
        plt.text(j, i, format(cm[i, j], f), # Add text to plot with format
                 horizontalalignment="center", # Center alignment
                 color="white" if cm[i, j] > t else "black") # Adjust the color of the text in the confusion matrix based on a specified threshold
        plt.ylabel('True') # y-axis label
    plt.xlabel('Predicted') #  x-axis label

    plt.tight_layout() # Adjust layout
# confusion matrix
confusion = confusion_matrix(test, predict)
np.set_printoptions(precision=2)

# plot normalized confusion matrix
plt.figure()
# Designate the colormap (cmap) for the color scheme used in the confusion matrix plot
plot_confusion_matrix(confusion, classes=class_name, title='Confusion Matrix', cmap=plt.cm.Reds)
plt.show() # Show the plot
```

Fig: 13 Code for confusion matrix

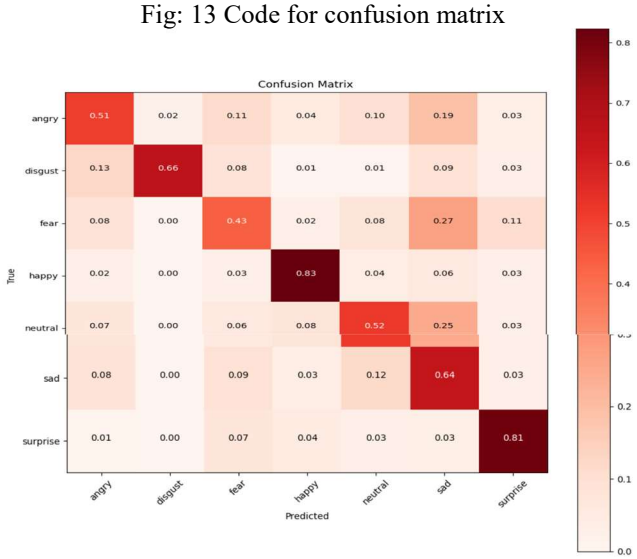


Fig 14: Confusion matrix

The diagonal elements dark red represent the true positive rates. For example, the model correctly classified 0.83 (83%) of the happy instances as happy. but struggles with other classes like angry 0.51 and fear 0.43.

STAGE -4- Evaluation metrics model B

Plotting the training and validation accuracy and loss curves for a machine learning model using the Matplotlib library. The variables accuracy, val_accuracy, loss, and val_loss contain the respective training and validation accuracy and loss values for each epoch during the model training process.

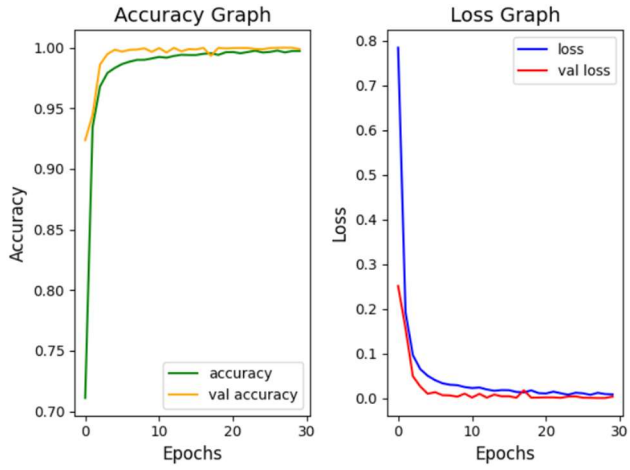


Fig :15 Accuracy and loss graphs

```python
#Assess the model's performance on the validation dataset and save the outcomes in the variable named 'evaluation'
evaluation=model_cnn1.evaluate(test_generl)
#Display the validation loss retrieved from the evaluation outcomes
print("Validation Loss:", evaluation[0])
# Output the validation accuracy acquired from the evaluation outcomes.
print("Validation Accuracy:", evaluation[1])
```

```
172/172 [==============================] - 48s 280ms/step - loss: 0.0034 - accuracy: 0.9988
Validation Loss: 0.0034051877446472645
Validation Accuracy: 0.9988183975219727
```

Fig: 16 Performance on the validation data

Achieving a very low training loss of 0.0034 and a high training accuracy of 99.88%. Additionally, the model's performance on the validation dataset is equally impressive, with a validation loss of 0.00340 and a validation accuracy of 99.88%.

Confusion Matrix:

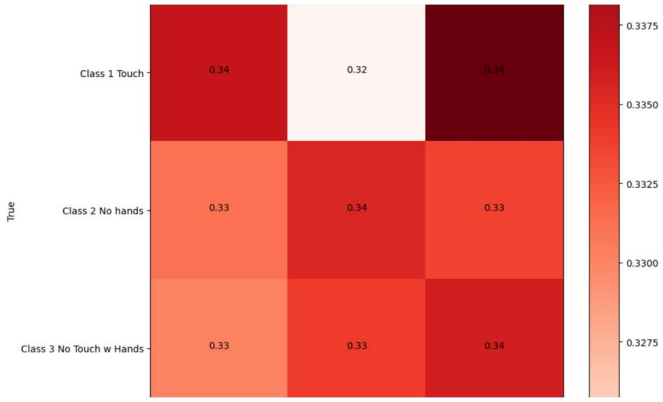


Fig 17: confusion matrix

The model seems to perform better at classifying instances from Class 1 Touch 34% correct and Class 2 No hands 34% correct, but struggles more with instances from Class 3 No Touch w Hands 34% incorrect, misclassified as Class 1 Touch.

Test file:

The test file evaluates its performance on a new testing dataset. The test loss and test accuracy for the pre-trained model are evaluated on the new testing data. Accuracy of the model's predictions by comparing the predicted labels with the actual labels. Written code for confusion matrix.

```python
# Load your trained model
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

base_path = "C:\\Users\\jalad\\Downloads\\final dataset" # base path
test_data_dir = ""                                      # test data path
model = load_model('model_cb.h5')  # this is the weights of th trained model

# data generator for the new testing data
test_datagen = ImageDataGenerator(rescale=1./255)
# a data generator for the new testing data
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(112, 112), # target size
    batch_size=64,         # batch size
    class_mode='categorical', # type of labels
    shuffle=False  # do not shuffle the data
)

# Evaluate the model on the new testing data
test_loss, test_acc = model.evaluate_generator(test_generator)
print(f'Test loss: {test_loss:.4f}, Test accuracy: {test_acc:.4f}') #the test loss and test accuracy

# Make predictions on the new testing data
predictions = model.predict(test_generator)
```

```python
import numpy as np

# Retrieve the predicted class labels by choosing the class with the highest probability for each prediction.
predict = [np.argmax(probas) for probas in predictions]
# the actual class labels from the validation dataset and save them into the variable named 'test'.
test = test_generator.classes
# Get the names of the classes from the validation dataset and store them in the variable named 'class_name'.
class_name = test_generator.class_indices.keys()
accuracy=np.mean(predict==test)
print("overall accuracy:" , accuracy)
```

V. *LINKS TO DATASETS*

a*) Facial Expression Recognition*

Dataset:

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

b)Don't touch your face

Dataset: Given by professor.

To test the model, the professor has test data. I created the test file.

VI. DISCUSSION

To improve the model, it is needed to perform data augmentation, facial landmark detection as a preprocessing step to localize and align faces, which can help the model focus on relevant facial regions and improve recognition accuracy.

Ensemble with Other Modalities: body gestures, which can provide complementary information and potentially improve overall recognition accuracy.

The combination of feature enhancement and dimensionality reduction through PCA can lead to better accuracy for both models.

REFERENCES

[1] Ketan Sarvakar, R. Senkamalavalli, S. Raghavendra, J. Santosh Kumar, R. Manjunath, Sushma Jaiswal, Facial emotion recognition using convolutional neural networks, Materials Today: Proceedings, Volume 80, Part 3, 2023, Pages 3560-3564, ISSN 2214-7853.

[1] Zhu D, Fu Y, Zhao X, Wang X, Yi H. Facial Emotion Recognition Using a Novel Fusion of Convolutional Neural Network and Local Binary Pattern in Crime Investigation. Comput Intell Neurosci. 2022 Sep 22;2022:2249417. doi: 10.1155/2022/2249417. PMID: 36188698; PMCID: PMC9522492.

[2] Michelin, Allan & Korres, George & Ba'ara, Sara & Assadi, Hadi & Alsuradi, Haneen & Rony, Ridowan & Argyros, Antonis & Eid, Mohamad. (2021). FaceGuard: A Wearable System To Avoid Face Touching. Frontiers in Robotics and AI. 8. 10.3389/frobt.2021.612392.

[3] Took some reference from Kaggle.