SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE NAME: Machine Learning Laboratory **COURSE CODE:** DJS22ITL602

CLASS: Third Year B.Tech **SEM:** VI

Name:Falak Shah

EXPERIMENT NO. 9 CO

Measured:

CO3 – Apply various machine learning techniques

TITLE: Mini Project: Stage II

AIM / OBJECTIVE: Mini Project

Step 4: Tuning and optimizing our model
Step 5:Making predictions

DESCRIPTION OF EXPERIMENT:

In this mini project you are expected to choose any algorithm in machine learning with respect to some use case of your choice. It can be a small-scale project where you apply machine learning algorithms to a specific dataset to solve a problem, often focusing on a single concept or technique, typically used for learning purposes and usually involving data collection, cleaning, feature engineering, model training, and evaluation within a manageable scope.

Key characteristics of a mini machine learning project to consider in this experiment:

Step 4 -Training using Machine Learning Model:

Once the data has been pre-processed and relevant features have been extracted, the next step is training the machine learning model. This involves:

Splitting the Data:

The dataset is divided into training, validation, and test sets (e.g., 70%-20%-10% split). Cross-validation techniques like k-fold cross-validation may be used to improve model reliability.

Selecting a Model:

- Based on the problem type (classification, regression, etc.), models like Decision Trees, SVM, Random Forest, Gradient Boosting, or Deep Learning architectures (CNNs, LSTMs, Transformers) are chosen.
- For ensemble learning, multiple models are combined to improve predictive accuracy.

Training the Model:

- The selected model is trained using the training dataset.
- Loss functions (e.g., Cross-Entropy Loss, MSE) and optimization algorithms (SGD, Adam, RMSprop) are used to minimize errors.
- Regularization techniques (L1, L2, Dropout) are applied to prevent overfitting. **Hyperparameter**

  Tuning:

  Grid Search, Random Search, or Bayesian Optimization is used to find the best hyperparameters (e.g., learning rate, batch size, number of layers, activation functions).

Handling Imbalanced Data (if applicable):

Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or class weighting are used to balance the dataset.

Step 4 -Evaluating Model Performance:

After training, the model's performance is assessed to ensure it generalizes well to unseen data. The evaluation process involves:

Performance Metrics:

- For classification tasks: Accuracy, Precision, Recall, F1-score, ROC-AUC, PR-AUC.
- For regression tasks: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R^2-score, Mean Absolute Error (MAE).
- For deep learning models: Loss curves, Confusion Matrices, and Custom Metrics may be analyzed.

Validation Techniques:

- The trained model is tested on the validation dataset, and hyperparameters are fine-tuned accordingly.
- k-fold cross-validation is performed to assess model robustness.

Bias-Variance Trade-off Analysis:

- High training accuracy but poor test performance indicates overfitting.
- Poor training and test accuracy suggest underfitting.
- Regularization, pruning, and ensemble learning techniques help in balancing bias and variance.

Error Analysis:

- Misclassified samples are analysed to understand model weaknesses.
- Feature importance is examined to determine which features contribute most to predictions.

Comparison with Baseline Models:

- The trained model is compared with simpler models or existing benchmarks to check if improvements are significant.

Deployability Check:

- The final model is tested for real-world scenarios, including latency, computational efficiency, and scalability before deployment.

PROCEDURE:

1. Train the selected model using appropriate techniques and optimize hyper-parameters.

```
import os
import zipfile
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.preprocessing.image import load_img, img_to_array,
ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import (
    Dense, Dropout, GlobalAveragePooling2D, Flatten, Conv2D,
```

```
    BatchNormalization, Activation, MaxPooling2D, Input, concatenate, SeparableConv2D
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, TensorBoard
from tensorflow.keras.applications import EfficientNetB0, ResNet50V2, MobileNetV2
from tensorflow.keras.utils import to_categorical

# For explainable AI
import lime
from lime import lime_image
import shap
from skimage.segmentation import mark_boundaries
from tensorflow.keras.models import load_model
import datetime
import cv2
import shutil
from sklearn.model_selection import train_test_split

# 1. Check GPU Availability and Configure Memory Growth FIRST
# This must be done before any other TensorFlow operations
physical_devices = tf.config.list_physical_devices('GPU')
print("Num GPUs Available:", len(physical_devices))
if len(physical_devices) > 0:
    try:
        # Configure memory growth for all GPUs
        for gpu in physical_devices:
            tf.config.experimental.set_memory_growth(gpu, True)
        print("Memory growth enabled for all GPUs")
    except RuntimeError as e:
        # If already initialized, print error but continue
        print(f"Error setting memory growth: {e}")

# 2. Enable mixed precision for better performance on compatible GPUs
try:
    policy = tf.keras.mixed_precision.Policy('mixed_float16')
    tf.keras.mixed_precision.set_global_policy(policy)
    print('Mixed precision enabled')
except Exception as e:
    print(f'Mixed precision not enabled: {e}')
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```

# 3. Extract Dataset
zip_path = "/content/archive (8).zip"
extract_path = "/content/dataset"

if not os.path.exists(extract_path):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
    print("Dataset extracted.")
else:
    print("Dataset already extracted.")

# 4. Define Paths
folder_path = "/content/dataset/images/"
train_dir = os.path.join(folder_path, "train")
val_dir = os.path.join(folder_path, "validation")

# Create a separate test set from validation
test_dir = os.path.join(folder_path, "test")
if not os.path.exists(test_dir):
    os.makedirs(test_dir)

    # Get all classes
    classes = os.listdir(val_dir)

    for cls in classes:
        val_class_dir = os.path.join(val_dir, cls)
        test_class_dir = os.path.join(test_dir, cls)

        if not os.path.exists(test_class_dir):
            os.makedirs(test_class_dir)

        # Get all files for this class
        files = os.listdir(val_class_dir)

        # Split files - move 30% to test
        test_files = np.random.choice(files, size=int(len(files)*0.3),
replace=False)

        # Move files to test directory
        for file in test_files:
```

```python
            shutil.move(os.path.join(val_class_dir, file), os.path.join(test_class_dir, file))

    print("Test set created from validation set.")

# 5. Data Augmentation & Data Generators - More Advanced
picture_size = 96  # Further increased for more details
batch_size = 32    # Reduced for better gradient updates

    # Function to apply advanced preprocessing
    def preprocess_image(img):
        # Convert to float
        img = img.astype(np.float32)

        # Histogram equalization for better contrast
        if len(img.shape) == 3 and img.shape[2] == 1:
            img = img[:,:,0]  # Get the single channel
            img = cv2.equalizeHist(img.astype(np.uint8))
            img = np.expand_dims(img, axis=-1)  # Add channel dimension back

        # Normalize to [0,1]
        img = img / 255.0

        return img

    # More advanced augmentation
    datagen_train = ImageDataGenerator(
        preprocessing_function=preprocess_image,
        rotation_range=15,
        width_shift_range=0.15,
        height_shift_range=0.15,
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
        brightness_range=[0.85, 1.15],
        validation_split=0.1,
        # Add slight noise for regularization
        zca_whitening=False,  # Too computationally expensive
        channel_shift_range=0.1
    )
```

```
datagen_val = ImageDataGenerator(
    preprocessing_function=preprocess_image
)

datagen_test = ImageDataGenerator(
    preprocessing_function=preprocess_image
)

train_set = datagen_train.flow_from_directory(
    train_dir,
    target_size=(picture_size, picture_size),
    color_mode="grayscale",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=True
)

val_set = datagen_val.flow_from_directory(
    val_dir,
    target_size=(picture_size, picture_size),
    color_mode="grayscale",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=False
)

test_set = datagen_test.flow_from_directory(
    test_dir,
    target_size=(picture_size, picture_size),
    color_mode="grayscale",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=False
)

class_names = list(train_set.class_indices.keys())
no_of_classes = len(class_names)
print(f"Classes: {class_names}")

# 6. Display Sample Images with preprocessing
```

```
    plt.figure(figsize=(15, 10))
    expression = class_names[0]  # Pick first class dynamically
    for i in range(9):
        plt.subplot(3, 3, i + 1)
        img_path = os.path.join(train_dir, expression, os.listdir(os.path.join(train_dir, expression))[i])

        # Show both original and preprocessed
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img_resized = cv2.resize(img, (picture_size, picture_size))

        # Apply preprocessing
        img_processed = cv2.equalizeHist(img_resized)

        # Display side by side
        combined = np.hstack((img_resized, img_processed))
        plt.imshow(combined, cmap='gray')
        plt.title(f"Original | Preprocessed", fontsize=8)
        plt.axis("off")
    plt.suptitle(f"Sample Images from Class: {expression}")
    plt.show()

    # 7. Define Advanced Model - Ensemble Approach
    def build_custom_cnn():
        input_img = Input(shape=(picture_size, picture_size, 1))

        # First pathway - standard convolutions
        x1 = Conv2D(64, (3, 3), padding='same')(input_img)
        x1 = BatchNormalization()(x1)
        x1 = Activation('relu')(x1)
        x1 = Conv2D(64, (3, 3), padding='same')(x1)
        x1 = BatchNormalization()(x1)
        x1 = Activation('relu')(x1)
        x1 = MaxPooling2D(pool_size=(2, 2))(x1)
        x1 = Dropout(0.2)(x1)

        # Second pathway - larger kernels for capturing broader facial features
        x2 = Conv2D(64, (5, 5), padding='same')(input_img)
        x2 = BatchNormalization()(x2)
        x2 = Activation('relu')(x2)
        x2 = Conv2D(64, (5, 5), padding='same')(x2)
```

```
        x2 = BatchNormalization()(x2)
        x2 = Activation('relu')(x2)
        x2 = MaxPooling2D(pool_size=(2, 2))(x2)
        x2 = Dropout(0.2)(x2)

        # Combine pathways
        x = concatenate([x1, x2])

        # Continue with deeper layers
        x = Conv2D(128, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = SeparableConv2D(128, (3, 3), padding='same')(x)  # More efficient
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)
        x = Dropout(0.3)(x)

        x = Conv2D(256, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = SeparableConv2D(256, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)
        x = Dropout(0.4)(x)

        x = Conv2D(512, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = SeparableConv2D(512, (3, 3), padding='same')(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)
        x = Dropout(0.4)(x)

        # Global pooling
        x = GlobalAveragePooling2D()(x)

        # Fully connected layers
        x = Dense(512)(x)
```

```
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Dropout(0.5)(x)

        x = Dense(256)(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Dropout(0.5)(x)

        # Output layer
        output = Dense(no_of_classes, activation='softmax')(x)

        model = Model(inputs=input_img, outputs=output)
        return model

    # Create model
    model = build_custom_cnn()

    # 8. Compile Model with advanced optimizer
    opt = Adam(learning_rate=0.0003)  # Further reduced learning rate
    model.compile(
        optimizer=opt,
        loss='categorical_crossentropy',
        metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall(), tf.keras.metrics.AUC()]
    )

    model.summary()

    # 9. Define Enhanced Callbacks
    # Create log directory for TensorBoard
    log_dir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
    os.makedirs(log_dir, exist_ok=True)

    checkpoint = ModelCheckpoint(
        "model_best.h5",
        monitor='val_accuracy',
        verbose=1,
        save_best_only=True,
        mode='max'
```

```
    )

    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=15,  # Increased patience
        verbose=1,
        restore_best_weights=True
    )

    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.1,
        patience=7,
        verbose=1,
        min_delta=0.0001,
        min_lr=0.00001
    )

    tensorboard_callback = TensorBoard(
        log_dir=log_dir,
        histogram_freq=1,
        update_freq='epoch'
    )

    callbacks_list = [early_stopping, checkpoint, reduce_lr, tensorboard_callback]

    # 10. Train the Model
    epochs = 75  # Further increased epochs with early stopping

    # history = model.fit(
    #     train_set,
    #     steps_per_epoch=train_set.n // train_set.batch_size,
    #     epochs=epochs,
    #     validation_data=val_set,
    #     validation_steps=val_set.n // val_set.batch_size,
    #     callbacks=callbacks_list
    # )

    # 11. Plot Enhanced Training History
    plt.style.use('dark_background')
```

```python

    plt.figure(figsize=(20, 15))
    plt.subplot(2, 2, 1)
    plt.title('Training and Validation Loss', fontsize=14)
    plt.ylabel('Loss', fontsize=12)
    plt.xlabel('Epochs', fontsize=12)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.legend(loc='upper right')
    plt.grid(True, linestyle='--', alpha=0.5)

    plt.subplot(2, 2, 2)
    plt.title('Training and Validation Accuracy', fontsize=14)
    plt.ylabel('Accuracy', fontsize=12)
    plt.xlabel('Epochs', fontsize=12)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.grid(True, linestyle='--', alpha=0.5)

    plt.subplot(2, 2, 3)
    plt.title('Learning Rate Over Time', fontsize=14)
    plt.ylabel('Learning Rate', fontsize=12)
    plt.xlabel('Epochs', fontsize=12)
    # Get learning rate from the optimizer's learning rate schedule
    try:
        # For newer TF versions
        lr_values = [opt._decayed_lr(tf.float32).numpy() for _ in range(len(history.history['loss']))]
    except (AttributeError, TypeError):
        # Fallback for other versions or if the first approach doesn't work
        if hasattr(opt, 'learning_rate'):
            initial_lr = opt.learning_rate.numpy() if hasattr(opt.learning_rate, 'numpy') else float(opt.learning_rate)
        else:
            initial_lr = 0.0003  # The initial value you set

        # Create a simplified version that just shows the effect of reduce_lr
        lr_values = []
        for i in range(len(history.history['loss'])):
            # Roughly estimate LR based on reduce_lr's effect
```

```
            lr_decline = sum(1 for j in range(i) if j > 0 and
                            history.history['val_loss'][j-1] - history.history['val_loss'][j] < reduce_lr.min_delta)
            lr_drops = lr_decline // reduce_lr.patience
            current_lr = initial_lr * (0.1 ** lr_drops)
            lr_values.append(current_lr)
    plt.plot(lr_values)
    plt.yscale('log')
    plt.grid(True, linestyle='--', alpha=0.5)

    plt.subplot(2, 2, 4)
    plt.title('Precision-Recall Metrics', fontsize=14)
    plt.ylabel('Value', fontsize=12)
    plt.xlabel('Epochs', fontsize=12)
    plt.plot(history.history['precision_1'], label='Training Precision')
    plt.plot(history.history['val_precision_1'], label='Validation Precision')
    plt.plot(history.history['recall_1'], label='Training Recall')
    plt.plot(history.history['val_recall_1'], label='Validation Recall')
    plt.plot(history.history['auc_1'], label='Training AUC')
    plt.plot(history.history['val_auc_1'], label='Validation AUC')
    plt.legend(loc='lower right')
    plt.grid(True, linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.savefig('training_history.png')
    plt.show()

    # 12. Evaluate Model on Test Set
    model.load_weights("model_best.h5")  # Load best weights

    # Evaluate on test set
    test_loss, test_acc, test_precision, test_recall, test_auc = model.evaluate(test_set)
    print(f"\nTest Accuracy: {test_acc:.4f}")
    print(f"Test Precision: {test_precision:.4f}")
    print(f"Test Recall: {test_recall:.4f}")
    print(f"Test AUC: {test_auc:.4f}")

    # Get predictions
    test_steps = np.ceil(test_set.n / test_set.batch_size)
    predictions = model.predict(test_set)
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```python
    y_pred = np.argmax(predictions, axis=1)
    y_true = test_set.classes[:len(y_pred)]  # Ensure same length

    # Plot confusion matrix with normalized values
    cm = confusion_matrix(y_true, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=(14, 12))
    sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('Normalized Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('confusion_matrix.png')
    plt.show()

    # Print classification report
    print("\nClassification Report:")
    report = classification_report(y_true, y_pred, target_names=class_names, output_dict=True)
    report_df = pd.DataFrame(report).transpose()
    print(report_df.round(3))

    # Save report to CSV
    report_df.to_csv('classification_report.csv')

    # 13. Explainable AI Implementation

    # LIME Implementation for Image Explanation
    def explain_with_lime(img_path, model, class_names):
        # Load and preprocess image
        img = load_img(img_path, target_size=(picture_size, picture_size), color_mode="grayscale")
        img_array = img_to_array(img)
        img_processed = preprocess_image(img_array)

        # First, get the model's prediction for this image
        single_img = np.expand_dims(img_processed, axis=0)
        pred = model.predict(single_img)[0]
```

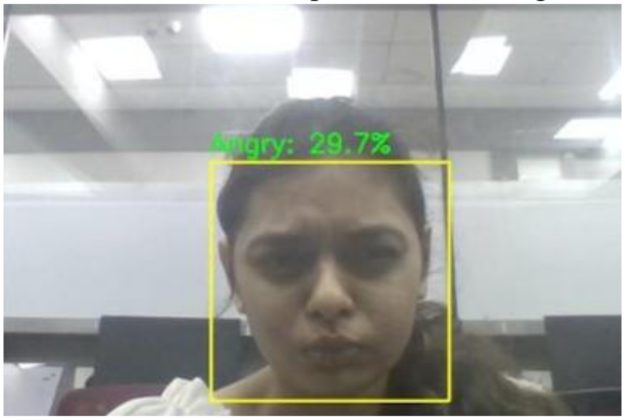
```
        top_pred_idx = np.argmax(pred)
        top_pred_label = class_names[top_pred_idx]
        top_pred_prob = pred[top_pred_idx] * 100

        # Function for LIME to make predictions
        def predict_fn(images):
            # LIME works with RGB, but our model needs grayscale
            gray_images = []
            for img in images:
                if img.shape[2] == 3:  # RGB
                    gray = cv2.cvtColor(img.astype(np.uint8), cv2.COLOR_RGB2GRAY)
                    gray = np.expand_dims(gray, axis=-1)
                else:  # Already grayscale
                    gray = img
                gray_images.append(preprocess_image(gray))

            batch = np.array(gray_images)
            preds = model.predict(batch)
            return preds

        # Create LIME explainer
        explainer = lime_image.LimeImageExplainer()

        # Create RGB version of image for LIME
        img_rgb = np.repeat(img_processed, 3, axis=-1)

        # Get explanation - Make sure we include the predicted class in top_labels
        explanation = explainer.explain_instance(
            img_rgb,
            predict_fn,
            labels=[top_pred_idx],  # Force LIME to explain the predicted class
            top_labels=5,  # Still look at top 5 classes
            hide_color=0,
            num_samples=1000
        )

        # Show the original image and explanation
        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)
```

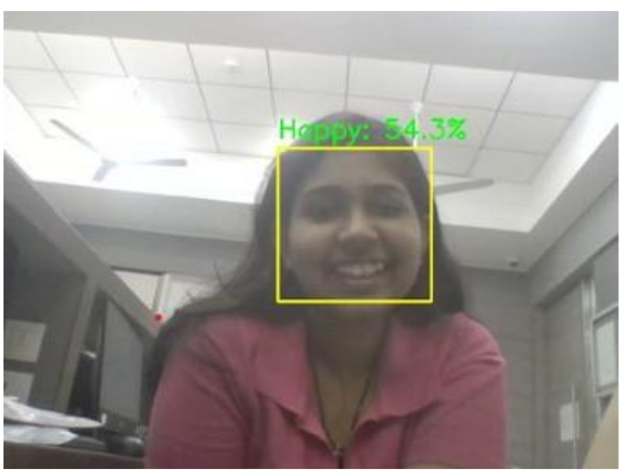
```
        plt.imshow(img, cmap='gray')
        plt.title(f"Original: Predicted {top_pred_label} ({top_pred_prob:.1f}%)")
        plt.axis('off')

        # Get mask for the predicted class
        try:
            temp, mask = explanation.get_image_and_mask(
                top_pred_idx,
                positive_only=True,
                num_features=5,
                hide_rest=False
            )

            plt.subplot(1, 2, 2)
            # Create RGB version for boundary marking
            img_for_boundaries = np.repeat(img_processed, 3, axis=-1)
            # Mark boundaries
            marked_img = mark_boundaries(img_for_boundaries, mask, color=(1, 0, 
0), mode='thick')
            plt.imshow(marked_img)
            plt.title(f"LIME Explanation: Important regions for 
{top_pred_label}")
            plt.axis('off')

        except KeyError:
            # If explanation still fails, show a message
            plt.subplot(1, 2, 2)
            plt.text(0.5, 0.5, "LIME explanation failed for this image",
                     horizontalalignment='center', verticalalignment='center')
            plt.axis('off')

        plt.tight_layout()
        plt.savefig(f'lime_explanation_{os.path.basename(img_path)}.png')
        plt.show()

        # Show predictions for all classes
        plt.figure(figsize=(10, 5))
        plt.bar(class_names, pred * 100)
        plt.title('Prediction Confidence for All Classes')
        plt.ylabel('Confidence (%)')
        plt.xlabel('Emotion')
```

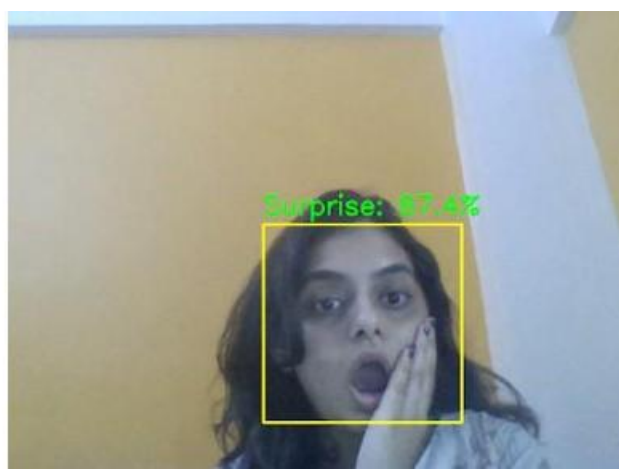
```
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig(f'prediction_confidence_{os.path.basename(img_path)}.png')
    plt.show()

    return explanation
# 14. Test the explainable AI on sample images
print("\nGenerating LIME explanations for sample images...")
# Get a few test images
test_images = []
for emotion in class_names:
    emotion_dir = os.path.join(test_dir, emotion)
    if os.path.exists(emotion_dir):
        images = os.listdir(emotion_dir)
        if images:
            test_images.append(os.path.join(emotion_dir, np.random.choice(images)))

# Apply LIME to 3 test images
for img_path in test_images[:3]:
    explanation = explain_with_lime(img_path, model, class_names)

# 16. Save the model and preprocessing information
model.save('emotion_recognition_final.h5')

# Save preprocessing information
preprocessing_info = {
    'picture_size': picture_size,
    'class_names': class_names
}

import json
with open('preprocessing_info.json', 'w') as f:
    json.dump(preprocessing_info, f)

print("\nModel training and evaluation complete. All visualization, analysis, and model saved.")
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

1. Evaluate the model's performance using relevant metrics and compare results with other models.







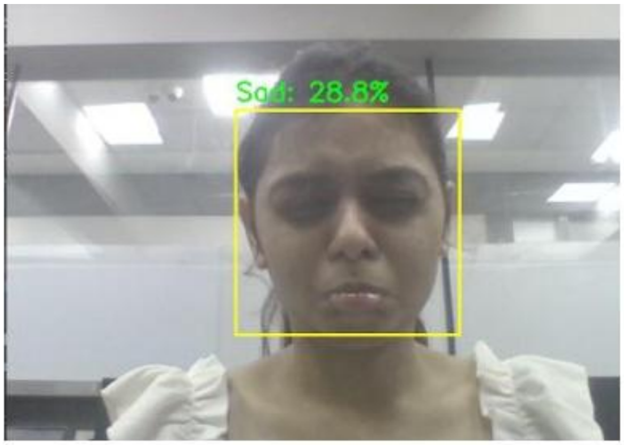


Fig. 13. Sad

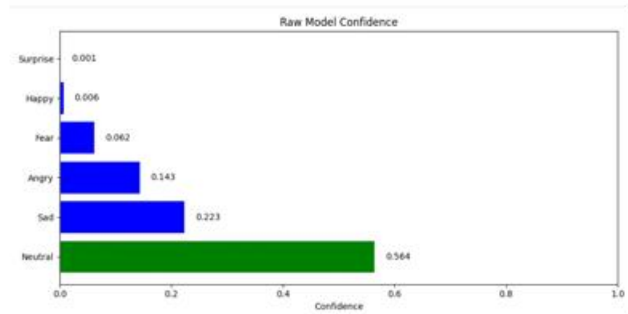


Fig. 15. Raw emotion detection output

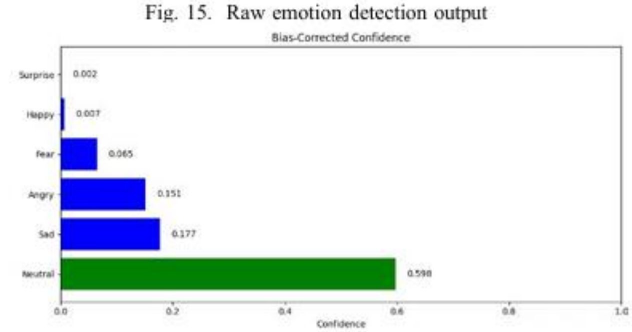


Fig. 16. Adjusted emotion visualization

CONCLUSION:

All results are shown after hypertunning of the model and the using lime for better results .

REFERENCES:

(List the references as per format given below and citations to be included the document)

1. Ethem Alpaydın, “Introduction to Machine Learning”, 4th Edition, The MIT Press, 2020.

2. Peter Harrington, “Machine Learning in Action”, 1st Edition, Dreamtech Press, 2012.

3. Tom Mitchell, “Machine Learning”, 1st Edition, McGraw Hill, 2017.

4. Andreas C, Müller and Sarah Guido, “Introduction to Machine Learning with Python: A Guide for Data Scientists”, 1st Edition, O'reilly, 2016.

5. Kevin P. Murphy, "Machine Learning: A Probabilistic Perspective", 1st Edition, MIT Press, 2012.

Website References:

[1] https://developers.google.com/machine-learning/guides/text-classification/step-4

[2]https://developers.google.com/machine-learning/guides/text-classification/step-5

[3] https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

[4]https://ai.plainenglish.io/ml-5-evaluating-machine-learning-models-how-to-measure-successbcfa24008e9c