**Project Title: Emotion Detection from Uploaded Images**

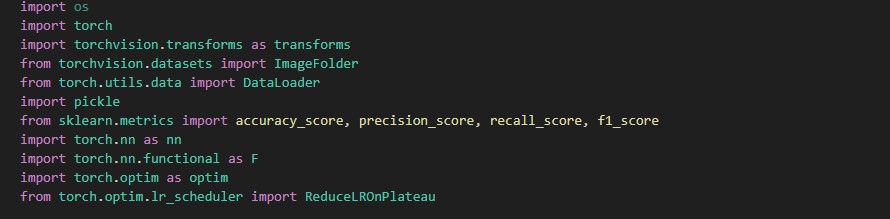
**Project Overview**

The "Emotion Detection from Uploaded Images" project aims to create a comprehensive system enabling users to upload an image through a Streamlit application and accurately detect and classify the emotion present in the image using Convolutional Neural Networks (CNNs). This project integrates machine learning, computer vision, and user interface design to develop a user-friendly application.

**Objectives**

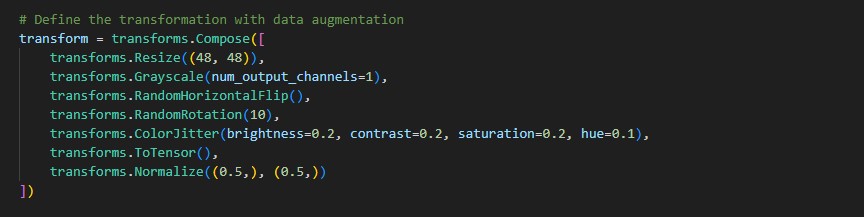
* Develop and design an end-to-end solution for emotion detection from images.
* Implement a polished, user-friendly Streamlit application for image upload and emotion classification.
* Address potential applications in healthcare, education, and customer service where emotion detection is valuable.

1. **Import**



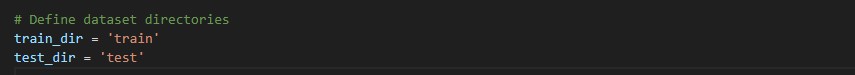
* **os**: Module to interact with the operating system.
* **torch**: Main PyTorch package.
* **torchvision.transforms**: Common image transformations.
* **torchvision.datasets**: Provides a way to load datasets.
* **torch.utils.data.DataLoader**: Provides an iterable over a dataset.
* **pickle**: Used for serializing and deserializing Python objects.
* **sklearn.metrics**: Metrics to evaluate model performance.
* **torch.nn**: Provides neural network layers.
* **torch.nn.functional**: Provides functional operations.
* **torch.optim**: Provides optimization algorithms.
* **torch.optim.lr\_scheduler**: Adjusts learning rate based on validation metrics.

1. **Data Augmentation and Transformation**

****

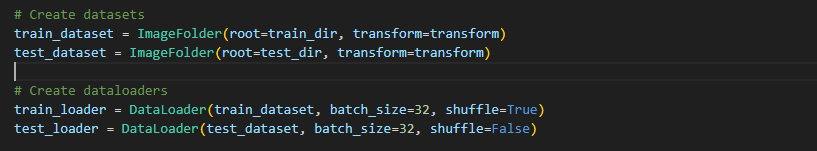
* transforms.Resize: Resizes images to 48x48 pixels.
* transforms.Grayscale: Converts images to grayscale.
* transforms.RandomHorizontalFlip: Randomly flips images horizontally.
* transforms.RandomRotation: Randomly rotates images.
* transforms.ColorJitter: Randomly changes brightness, contrast, saturation, and hue.
* transforms.ToTensor: Converts images to PyTorch tensors.
* transforms.Normalize: Normalizes images to have mean 0.5 and standard deviation 0.5.

1. **Dataset Directories**

****

* Defines the directories where training and test images are stored.

1. **Create Datasets and DataLoaders**

****

* ImageFolder: Loads images from the specified directories and applies transformations.
* DataLoader: Creates iterable data loaders with specified batch size and shuffling.

1. **Define the CNN Model**



Emotion CNN: Defines the architecture of the CNN.

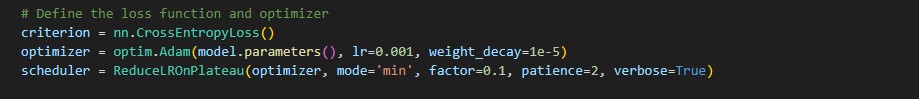
* conv1, conv2, conv3: Convolutional layers.
* pool: Max-pooling layer.
* fc1, fc2: Fully connected layers.
* dropout: Dropout layer for regularization.

1. **Instantiate the Model**



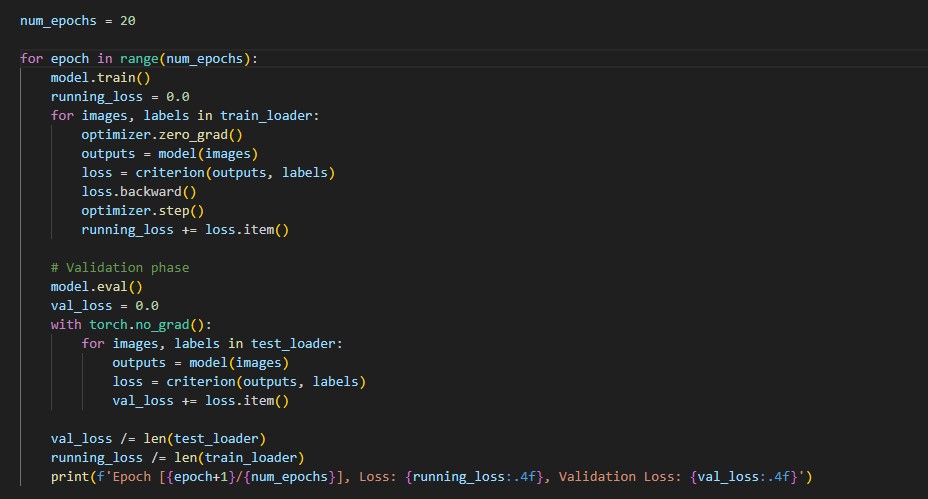
* Creates an instance of the CNN model.

1. **Define Loss Function and Optimizer**



* criterion: Cross-entropy loss for classification.
* optimizer: Adam optimizer with learning rate and weight decay.
* scheduler: Reduces learning rate when a plateau in validation loss is detected.

1. **Training Loop**



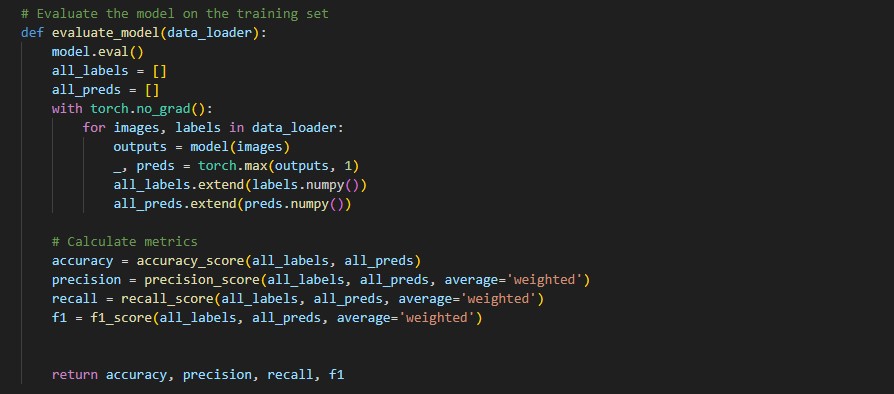
* model.train(): Sets the model to training mode.
* optimizer.zero\_grad(): Clears previous gradients.
* outputs = model(images): Forward pass.
* loss = criterion(outputs, labels): Calculates loss.
* loss.backward(): Backward pass to calculate gradients.
* optimizer.step(): Updates model parameters.
* model.eval(): Sets the model to evaluation mode.
* with torch.no\_grad(): Disables gradient calculation for validation.
* val\_loss: Accumulates and averages validation loss.

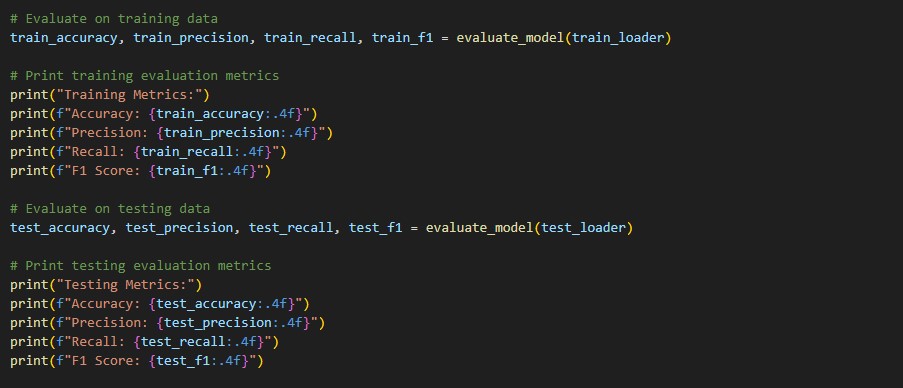
1. **Save the Trained Model**

****

* model\_path: Path to save the model.
* pickle.dump(): Saves the model state dictionary.

1. **Evaluate the Model**

****

****

* evaluate\_model(): Evaluates the model on a given data loader.
* torch.max(outputs, 1): Gets predicted labels.
* accuracy\_score(), precision\_score(), recall\_score(), f1\_score(): Calculate evaluation metrics.

The evaluation results for both training and testing datasets are printed. This provides insight into the model's performance on both seen and unseen data.