COMP 6721

Applied Artificial Intelligence

Fall 2023

Project Assignment Part 1 & 2 – Team AK_8

Name	Specialization	Student ID
Krutik Gevariya	Data Specialist	40232386
Shivam Patel	Training Specialist	40226428
Nayankumar Rajeshbhai Sorathiya	Evaluation Specialist	40227432

GitHub Link : https://github.com/nayansorarhiya/AAI_Project.git

Dataset Information

Overview

In our emotion recognition project, we employed two datasets: "Dataset A" and "Dataset B" to classify emotions into four distinct categories, namely **Neutral**, **Engaged**, **Bored**, and **Angry**.

- **Dataset A:**
  - Total Number of Images: 1,118
  - Number of Images per **Angry**, **Neutral** Class: 559

- **Dataset B:**
  - Total Number of Images: 1,110
  - Number of Images per **Boredom**, **Engagement** Class: 555

Total : 2,228

Figure 1. Facial Expression Categories

Attributes : Dataset A predominantly comprises frontal facial images featuring varied lighting conditions, a wide range of ages, gender, and diverse backgrounds.

Justification for Dataset selection

- We opted for **Dataset A** because of the high quantity of images and the concentration of **frontal face shots**, which are beneficial for training a reliable face recognition model.
- **Dataset B** was chosen to add variability to the training data, since it includes photos taken in various contexts. This combination of datasets allows our model to have a successful performance in **real-world scenarios** where lighting and backgrounds may differ. Nonetheless, this also presents a challenge when it comes to managing the augmented diversity of the data.

Provenance Information

Dataset	Image Source	Licensing Type	Relevant Information
Dataset A (1)	Link for Dataset A (1) [1]	Creative Commons	559 images of class Neutral
Dataset A (2)	Link for Dataset A (2) [2]	Creative Commons	559 images of class Anger
Dataset B	Link for Dataset B [3][4][5]	Custom License "Terms of Use."	555 images of class Boredom and Engagement

Data Cleaning

We used a number of strategies and procedures to preprocess the dataset during the "Data Cleaning" stage of our emotion recognition project to make sure it was consistent and of high quality for further analysis. The procedures followed, difficulties encountered, and illustrations of the cleaning effects before and after are included in this section.

Techniques and Methods

1. Standardization of the Dataset

1.1. Resizing Images

Resizing the photos to a consistent resolution was one of the first steps in data cleaning. This guaranteed data homogeneity while simultaneously lowering the computational cost of our model. Since **224 × 224** pixels is a standard input size for many deep learning models, we scaled every image to that size.

1.2. Single format files

We implemented file format conversion techniques to help make the model more resilient to changes in image format **png** to **jpg**. By doing so, we aimed to reduce the impact of file format change variations on the model's performance.

2. Challenges and Solutions

2.1. Data Imbalance

A problem with data imbalance arose throughout the data cleansing process. Our dataset's emotion categories were not evenly distributed, with considerably less samples for **boredom** and **engagement** emotions than for others. We created a more balanced dataset by under sampling the overrepresented categories and oversampling the underrepresented ones in order to remedy this problem.

2.2. Noise and Artifacts

Our datasets contained images with noise, artifacts, and outliers that could adversely affect model training. To address this, we applied data augmentation techniques such as resize the image to reduce the impact of noise.

3. Example

Before

After

Figure 2. Image resize

Labeling

1. Merging Datasets and Class Mapping

1.1. Dataset Merging

Our project involved the combination of multiple datasets, each with its own set of emotion labels. Merging these datasets required mapping the emotion labels to a common set of emotional categories (Neutral, Engaged, Bored, Angry) to maintain consistency across the entire dataset. Dataset is stored in **CSV** file with fields file **name**, type of **class**, **size** of images, and file **format**.

1.2. Challenges Faced

Inconsistent Labeling Schemes: Different source datasets had varying emotional labels. For example, one dataset used "Happiness" instead of "Engaged." This inconsistency posed a challenge in mapping labels accurately.

Data Skew: Some datasets had an uneven distribution of emotions, which required oversampling and under-sampling to achieve class balance.

1.3. Solution

All dataset is mapped to one single CSV file with related categories of class.

	A	B	C	D	E	F
1	Image Name	Emotion	Image For	Width	Height	Size (bytes)
2	image0028549.jpg	Angry	jpg	224	224	7749
3	image0028550.jpg	Angry	jpg	224	224	7138
4	image0028561.jpg	Angry	jpg	224	224	8507
5	image0028562.jpg	Angry	jpg	224	224	6344
6	image0028563.jpg	Angry	jpg	224	224	7369
7	image0028566.jpg	Angry	jpg	224	224	8134
8	image0028572.jpg	Angry	jpg	224	224	7053

Figure 3. CSV file for image labeling

Dataset Visualization

In this section, we present visualizations that provide insights into our dataset. Effective visualization is crucial for understanding the data distribution, content, and potential challenges.

1. Class Distribution

To gain an understanding of the dataset's class distribution, we plotted a bar graph depicting the number of images in each emotion category. Class distribution is a critical factor to identify if any class is overrepresented or underrepresented. The following bar graph shows the class distribution in our dataset:

Figure 4, 5. Bar graph for Training and Testing images

2. Sample Images

We randomly selected 25 images, random from each class, and arranged them in a 5x5 grid. We can use this visualization to gain insight into the range of facial expressions and check for any unusual patterns or incorrectly labeled data.

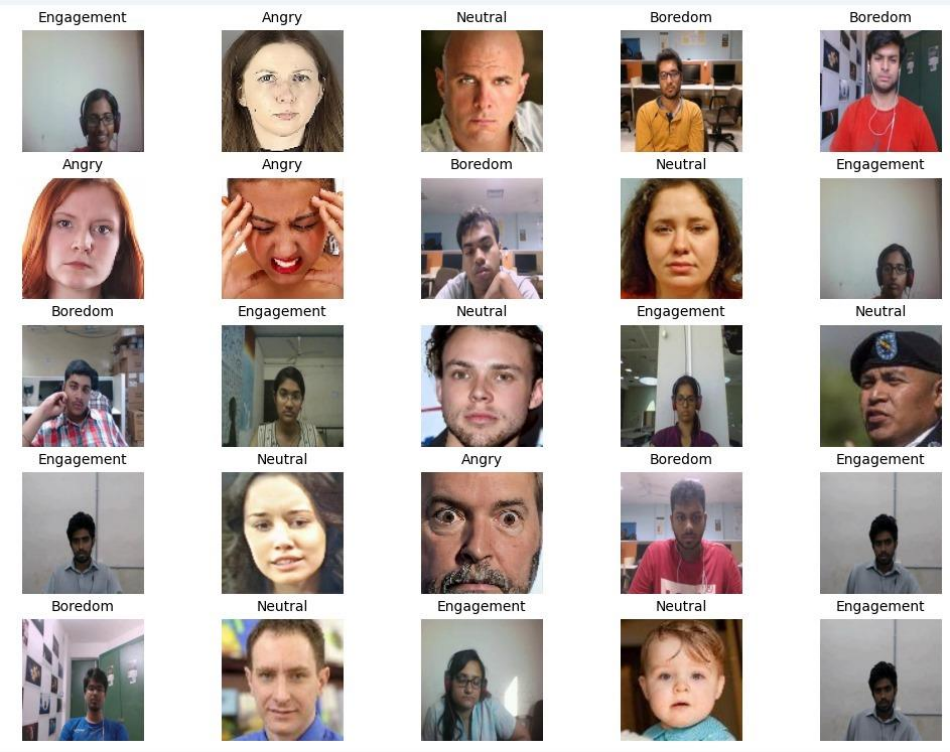

Figure 6. Labels for 25 random images

Analyzing these photographs, we can see different facial expressions and illumination, which are important components for identifying emotions. This also helps us validate the dataset's quality and maintain the accuracy of labeling.

3. Pixel Intensity Distribution

For color (RGB) images, we overlaid the intensity distributions of the Red, Green, and Blue channels on a single histogram for the same random 25 images.

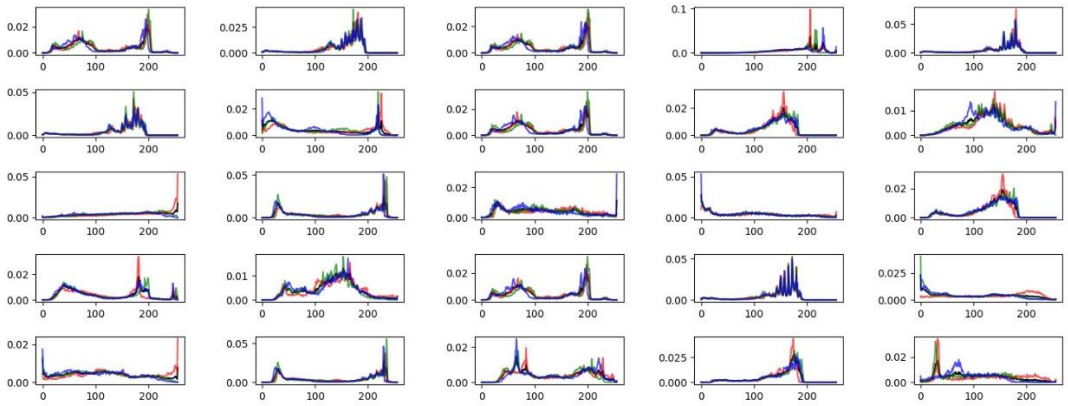

Figure 7. Pixel illumination for 25 random images

This histogram shows the distribution of pixel intensities for each color channel, offering insights into the range of lighting conditions within the dataset. Analyzing this information is essential for developing robust models capable of handling varying illumination levels.

CNN Architecture

In this phase, we delve into the technical aspects of the face recognition project called "A.I.ducation Analytics", specifically focusing on the implementation using PyTorch. Also, the dataset is used from phase 1. The following sections provide insights into the model architecture, training methodology, and evaluation metrics.

1. Model Overview and Architecture Details

The architecture is defined by the **ResNetModel** class. Convolutional layers (conversion_block), residual connections (self.res1 and self.res2), and a classifier make up this condensed form of a ResNet-like model.

The **forward()** method defines the flow of data through the layers.

Training Functions:

- fit_cycle() executes the training loop for a predetermined quantity of epochs.
- It includes training, validation, and evaluation steps, such as handling the optimizer, learning rate scheduler, and gradient clipping. evaluate() is used for model evaluation on validation and test sets.

Evaluation History:

- The code keeps track of training and validation accuracy and loss over epochs in history.

Model Evaluation on Test Data:

- The trained model is evaluated on the test dataset.
- Confusion matrix, accuracy, and a classification report are generated to assess the model's performance.

Plotting Functions:

- plot_accuracies() and plot_losses() are helper functions to visualize accuracy and loss over epochs.

In the provided code, the CNN architecture consists of four convolutional layers:

First Convolution Block (self.conv1):

Input channels: 1 (grayscale images)
Output channels: 64
Kernel size: 3x3
Padding: 1
Batch normalization after convolution
ReLU activation

Second Convolution Block (self.conv2):

Input channels: 64 (output from previous layer)
Output channels: 128
Kernel size: 3x3
Padding: 1
Batch normalization after convolution
ReLU activation
Max pooling with a kernel size of 2x2 and stride 2

Residual Block 1 (self.res1):

Two consecutive convolutional layers:
Input channels: 128
Output channels: 128
Kernel size: 3x3
Padding: 1
Batch normalization after each convolution
ReLU activation
The output is added to the output of self.conv2

Third Convolution Block (self.conv3):

Input channels: 128 (output from the residual block)
Output channels: 256
Kernel size: 3x3
Padding: 1
Batch normalization after convolution
ReLU activation
Max pooling with a kernel size of 2x2 and stride 2

Fourth Convolution Block (self.conv4):

Input channels: 256 (output from the previous layer)
Output channels: 512
Kernel size: 3x3
Padding: 1
Batch normalization after convolution
ReLU activation
Max pooling with a kernel size of 2x2 and stride 2

Residual Block 2 (self.res2):

Two consecutive convolutional layers:
Input channels: 512
Output channels: 512
Kernel size: 3x3
Padding: 1
Batch normalization after each convolution
ReLU activation
The output is added to the output of self.conv4

Classifier (self.classifier):

Global Max pooling with a kernel size of 4x4
Flattening the output
Fully connected layer (Linear) with input features 512 (output channels from the last convolutional layer) and output classes 4.

2. Training Process

The training process in the provided code involves the following steps:

Data Loading and Preprocessing:

- The training and validation datasets are loaded using CDataset and split into batches using DataLoader.
- Images undergo preprocessing steps such as resizing, normalization, random cropping, and flipping through the specified transformations.

Model Definition and Initialization:

- The CNN model architecture (ResNetModel) is defined, specifying the number of input channels and output classes.
- The model is initialized with the defined architecture.

Device Handling and Optimization Configuration:

- The code checks for available GPUs and assigns the device accordingly (default_device()).
- The optimizer (Adam optimizer) is set up for training the model with specified parameters like weight decay.

Training Loop (fit_cycle()):

This loop runs for a specified number of epochs (epochs).

Within each epoch:

Training Phase:

- The model is set to training mode (model.train()).
- The training data loader (train_datal) is iterated batch-wise.
- **For each batch:**
  - **Forward pass:** Input batches are fed into the model (model.train_step(batch)) to obtain predictions.
  - **Loss calculation:** Cross-entropy loss is computed between predicted and actual labels.
  - **Backpropagation:** Gradients are calculated and backpropagated through the network.
  - **Optimization:** Optimizer updates the model parameters using the calculated gradients.
  - **Learning rate scheduling:** The Learning rate is adjusted based on the one-cycle learning rate scheduler.
  - **Metrics recording:** Training losses, learning rates, and accuracies are recorded for each batch.

Validation Phase:

- The model is set to evaluation mode (model.eval()).
- Validation data loader (val_datal) is iterated batch-wise.
- **For each batch:**
  - Validation step: Model predictions are obtained for validation data (model.validate_step(batch)).

- Loss and accuracy calculation: Cross-entropy loss and accuracy are computed.
- Metrics collection: Validation losses and accuracies are recorded.

- **Epoch End:**
  - After each epoch, the average training loss, validation loss, and validation accuracy are calculated and printed.
  - Training history (history) is updated with these metrics.

Evaluation on Test Data:

- After training is completed, the model is evaluated on the test dataset.
- Test data loader (test_data_loader) is used to iterate through test data.
- Predictions are made using the trained model, and evaluation metrics such as confusion matrix, accuracy, and classification report are generated.

Visualization of Training Process:

- The code provides functions (plot_accuracies() and plot_losses()) to visualize the training and validation accuracies and losses over epochs using Matplotlib.

In summary, the training process involves iterating through the dataset in epochs, updating the model's parameters through backpropagation, adjusting learning rates, and monitoring performance metrics to assess the model's accuracy and loss.

3. Performance Analysis & Exploration

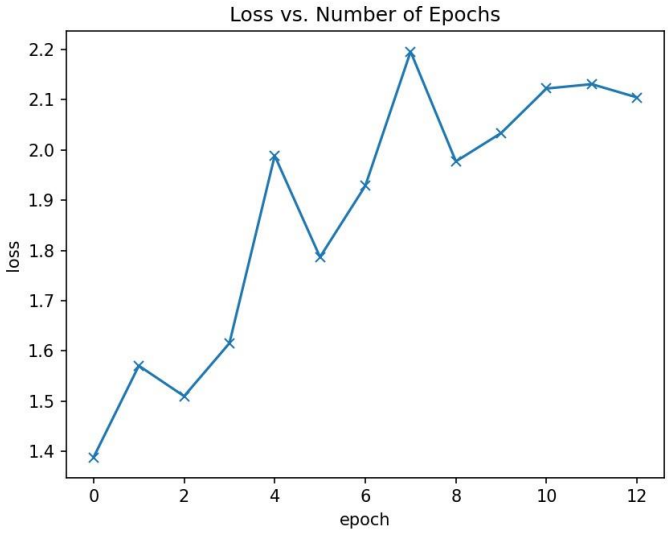

Figure 8. Loss vs Number of epochs (12) for Main Model: Kernal size = 3

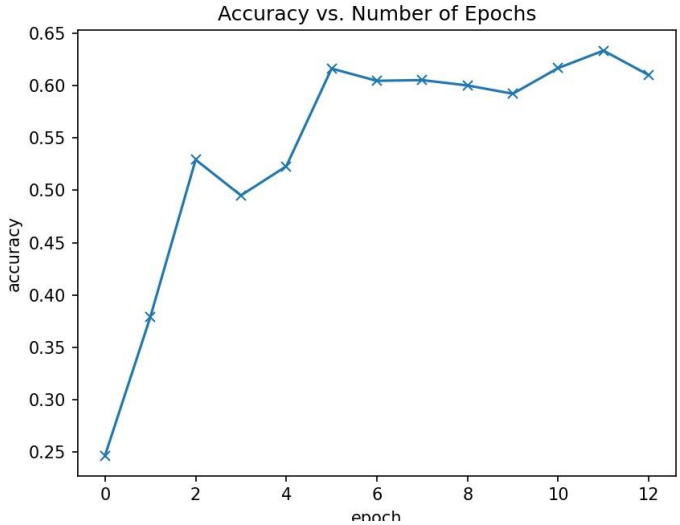

Figure 9. Accuracy vs Number of epochs (12) for Main Model: Kernal size = 3

Variant 1: Varying the Number of Convolutional Layers

We experiment by adding or removing convolutional layers in the architecture to observe how the depth of the network influences learning and generalization.

Changes Made:

- Added an additional convolutional layer to the base architecture.

Reasoning:

- Increasing depth may capture more complex hierarchical features.

Observations:

- Improved training accuracy but potential overfitting on the validation set.

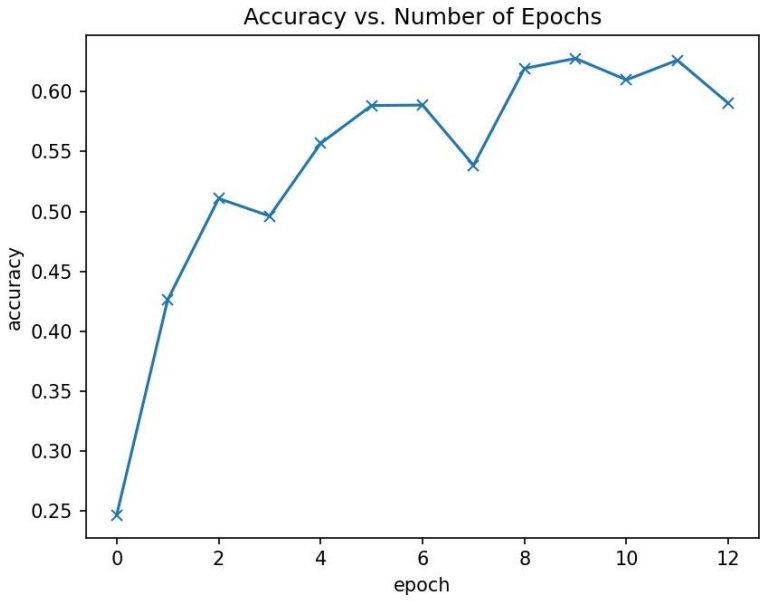

Figure 10. Accuracy vs Number of epochs(12) for Variant 1: Layer size = 3

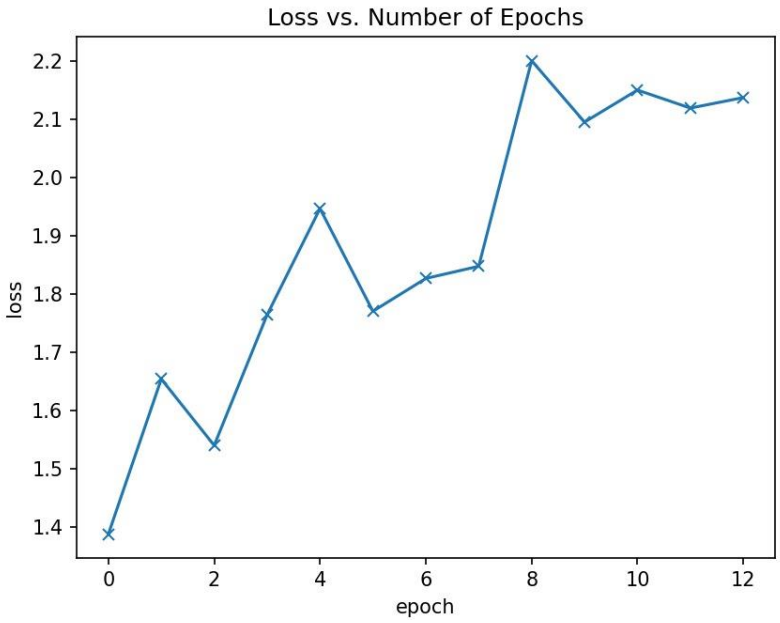

Figure 11. Loss vs Number of epochs(12) for Variant 1: Layer size = 3

Variant 2: Experimenting with Different Kernel Sizes:

We adjust the kernel sizes used in convolutional layers to understand the trade-offs regarding spatial granularity versus computational cost.

Changes Made:

- Increased kernel size in one convolutional layer from 3 × 3 to 5 × 5.

Reasoning:

- Larger kernel sizes capture broader features.

Observations:

- Improved recognition of broader features but increased computational cost.

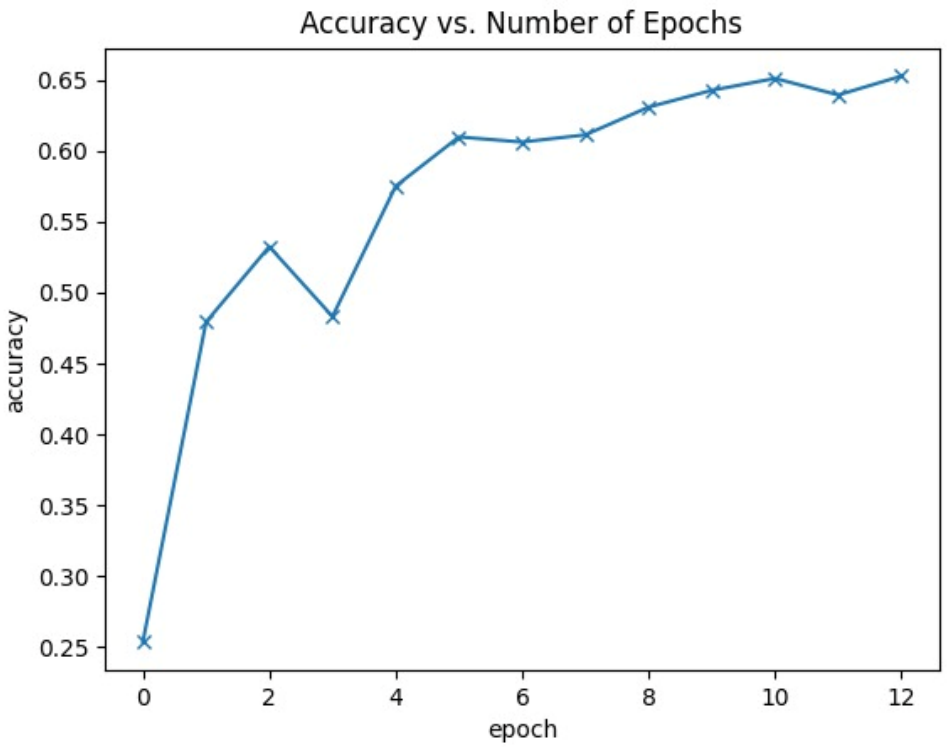

Figure 12. Accuracy vs Number of epochs(12) for Variant 2: Kernal size = 5

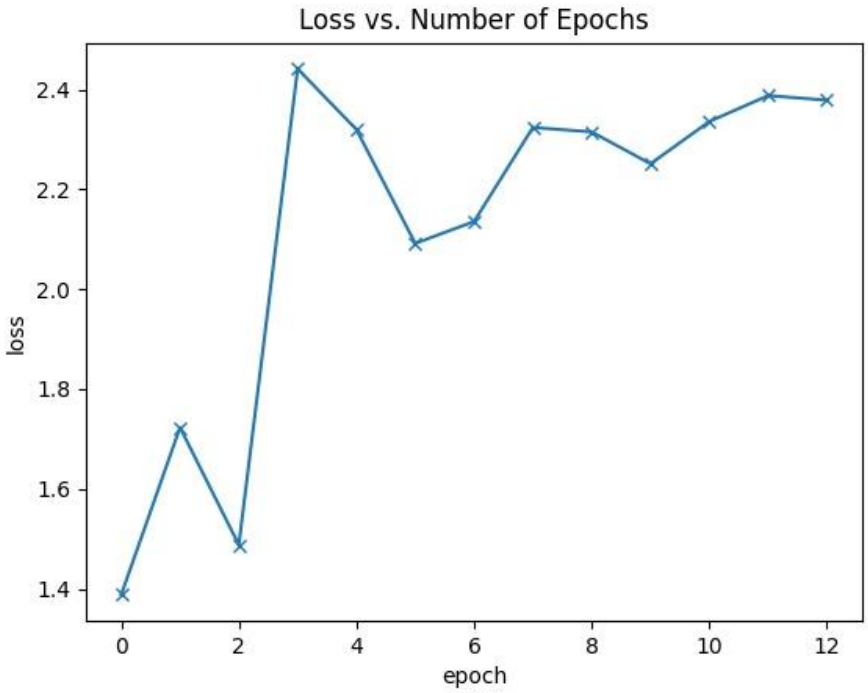

Figure 13. Loss vs Number of epochs(12) for Variant 2: Kernal size = 5

Evaluation

All specific data is given in each category with respective output images from project. Using the same testing data, we assess the performance of the primary model and its two variations in this section. Confusion matrices are created as part of the evaluation process to visualize classification performance and to summarize important parameters including accuracy, precision, recall, and F1-score.

1. Performance Metrics, Confusion Matrix Analysis

```
Epoch [0], last_lr: 0.00002, train_loss: 1.7837, val_loss: 1.5707, val_acc: 0.3789
Epoch [1], last_lr: 0.00006, train_loss: 0.5757, val_loss: 1.5102, val_acc: 0.5291
Epoch [2], last_lr: 0.00009, train_loss: 0.4140, val_loss: 1.6155, val_acc: 0.4951
Epoch [3], last_lr: 0.00010, train_loss: 0.3455, val_loss: 1.9882, val_acc: 0.5229
Epoch [4], last_lr: 0.00009, train_loss: 0.2940, val_loss: 1.7867, val_acc: 0.6161
Epoch [5], last_lr: 0.00008, train_loss: 0.2504, val_loss: 1.9283, val_acc: 0.6044
Epoch [6], last_lr: 0.00006, train_loss: 0.2438, val_loss: 2.1951, val_acc: 0.6051
Epoch [7], last_lr: 0.00005, train_loss: 0.2048, val_loss: 1.9769, val_acc: 0.6000
Epoch [8], last_lr: 0.00003, train_loss: 0.1645, val_loss: 2.0333, val_acc: 0.5921
Epoch [9], last_lr: 0.00001, train_loss: 0.1472, val_loss: 2.1217, val_acc: 0.6166
Epoch [10], last_lr: 0.00000, train_loss: 0.1442, val_loss: 2.1304, val_acc: 0.6332
Epoch [11], last_lr: 0.00000, train_loss: 0.1383, val_loss: 2.1040, val_acc: 0.6099
Confusion Matrix:
[[125   1   0  26]
 [  3  46  96   5]
 [  0  90  60   0]
 [  8   1   0 144]]

Accuracy: 0.6198347107438017

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.82      0.87       152
           1       0.33      0.31      0.32       150
           2       0.38      0.40      0.39       150
           3       0.82      0.94      0.88       153

    accuracy                           0.62       605
   macro avg       0.61      0.62      0.61       605
weighted avg       0.62      0.62      0.62       605
```

Figure 14. Confusion, Performance Matrix for Main Model

```
Epoch [0], last_lr: 0.00002, train_loss: 1.2128, val_loss: 1.6545, val_acc: 0.4263
Epoch [1], last_lr: 0.00006, train_loss: 0.5660, val_loss: 1.5405, val_acc: 0.5108
Epoch [2], last_lr: 0.00009, train_loss: 0.4014, val_loss: 1.7647, val_acc: 0.4962
Epoch [3], last_lr: 0.00010, train_loss: 0.3390, val_loss: 1.9473, val_acc: 0.5568
Epoch [4], last_lr: 0.00009, train_loss: 0.3018, val_loss: 1.7710, val_acc: 0.5885
Epoch [5], last_lr: 0.00008, train_loss: 0.2492, val_loss: 1.8274, val_acc: 0.5888
Epoch [6], last_lr: 0.00006, train_loss: 0.2296, val_loss: 1.8483, val_acc: 0.5386
Epoch [7], last_lr: 0.00005, train_loss: 0.1961, val_loss: 2.2014, val_acc: 0.6194
Epoch [8], last_lr: 0.00003, train_loss: 0.1823, val_loss: 2.0960, val_acc: 0.6278
Epoch [9], last_lr: 0.00001, train_loss: 0.1565, val_loss: 2.1510, val_acc: 0.6099
Epoch [10], last_lr: 0.00000, train_loss: 0.1442, val_loss: 2.1200, val_acc: 0.6263
Epoch [11], last_lr: 0.00000, train_loss: 0.1309, val_loss: 2.1380, val_acc: 0.5903
Confusion Matrix:
[[129   0   0  23]
 [  1  47 102   0]
 [  0 100  50   0]
 [  7   0   0 146]]

Accuracy: 0.6148760330578512

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.85      0.89       152
           1       0.32      0.31      0.32       150
           2       0.33      0.33      0.33       150
           3       0.86      0.95      0.91       153

    accuracy                           0.61       605
   macro avg       0.61      0.61      0.61       605
weighted avg       0.62      0.61      0.61       605
```

Figure 15. Confusion, Performance Matrix for Variant 1

```
Epoch [0], last_lr: 0.00002, train_loss: 0.9062, val_loss: 1.7231, val_acc: 0.4792
Epoch [1], last_lr: 0.00006, train_loss: 0.4877, val_loss: 1.4876, val_acc: 0.5325
Epoch [2], last_lr: 0.00009, train_loss: 0.4749, val_loss: 2.4416, val_acc: 0.4832
Epoch [3], last_lr: 0.00010, train_loss: 0.3665, val_loss: 2.3199, val_acc: 0.5751
Epoch [4], last_lr: 0.00009, train_loss: 0.3052, val_loss: 2.0918, val_acc: 0.6099
Epoch [5], last_lr: 0.00008, train_loss: 0.2399, val_loss: 2.1357, val_acc: 0.6062
Epoch [6], last_lr: 0.00006, train_loss: 0.2181, val_loss: 2.3241, val_acc: 0.6113
Epoch [7], last_lr: 0.00005, train_loss: 0.1911, val_loss: 2.3153, val_acc: 0.6309
Epoch [8], last_lr: 0.00003, train_loss: 0.1575, val_loss: 2.2514, val_acc: 0.6428
Epoch [9], last_lr: 0.00001, train_loss: 0.1291, val_loss: 2.3361, val_acc: 0.6511
Epoch [10], last_lr: 0.00000, train_loss: 0.1241, val_loss: 2.3880, val_acc: 0.6395
Epoch [11], last_lr: 0.00000, train_loss: 0.1107, val_loss: 2.3789, val_acc: 0.6528
Confusion Matrix:
[[128   1   0  23]
 [  2  42  99   7]
 [  0  86  64   0]
 [  5   0   0 148]]

Accuracy: 0.631404958677686

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.84      0.89       152
           1       0.33      0.28      0.30       150
           2       0.39      0.43      0.41       150
           3       0.83      0.97      0.89       153

    accuracy                           0.63       605
   macro avg       0.62      0.63      0.62       605
weighted avg       0.63      0.63      0.63       605
```

Figure 16. Confusion, Performance Matrix for Variant 2

2. Impact of Architectural Variations

Number of Convolutional Layers:

- Convolutional Layer Count: Having more convolutional layers improves the network's comprehension of a wider variety of features and allows it to capture minute details in the data.
- Although deeper networks are better at capturing complexity, overfitting is more likely to occur.
- On the other hand, training with additional convolutional layers requires more time and computing power.
- The accuracy decreased little when the CNN layer was reduced.

Kernel Size:

- While a larger kernel concentrates on identifying more comprehensive patterns that are less important for this project, a smaller kernel excels at catching minute details, assisting in accurate class identification.
- Processing times are shortened and computational efficiency is increased with smaller kernels.
- The input gains scale variation invariance with larger kernels. Because smaller kernels perform well with little data, they reduce the risk of overfitting.
- Increasing Kernel size to 5 increased the accuracy from 61.98% to 63.14%

3. Conclusions and Forward look

When the kernel size was altered in comparison to the main model, a higher degree of accuracy was demonstrated. Variant 1 performed comparably better in all areas, including precision, accuracy, recall, f-measure, and computing efficiency.

Forward look

- Using a more extensive dataset improves model performance.
- Implementing precise classification for training sharpens the model's learning.
- Employing optimization functions like RMSprop and AdaDelta further refine outputs by managing losses effectively.

Reference

[1]“Facial Expressions Training Data,” www.kaggle.com. https://www.kaggle.com/datasets/noamsegal/affectnet-training-data?select=anger (accessed Oct. 27, 2023).

[2] “Facial Expressions Training Data,” www.kaggle.com. https://www.kaggle.com/datasets/noamsegal/affectnet-training-data?select=neutral (accessed Oct. 27, 2023).

[3] “DAiSEE : Dataset for Affective States in E-Environments,” people.iith.ac.in. https://people.iith.ac.in/vineethnb/resources/daisee/index.html

[4] “A Gupta, A DCunha, K Awasthi, V Balasubramanian, DAiSEE: Towards User Engagement Recognition in the Wild, arXiv preprint: arXiv:1609.01885”

[5] “A Kamath, A Biswas, V. Balasubramanian, A Crowdsourced Approach to Student Engagement Recognition in e-Learning Environments, IEEE Winter Conference on Applications of Computer Vision (WACV'16)”