THE UNIVERSITY OF DANANG

UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Advanced Science and Technology

ARTIFICIAL INTELLIGENCE IN ENGINEERING

REPORT

FINAL PROJECT

RECOGNIZING STUDENT'S EMOTIONS AND CONCENTRATION USING DEEP LEARNING ALGORITHMS

Instructor: Prof. Nguyen Thi Anh Thu

Prof. Pham Van Tuan

Class: 21ES

Group: 5

Group members: Nguyen The Nhan

Phan Dinh Khanh

Phan Ba Nguyen Bao

Tran Phuoc Dien

Da Nang, 16th June 2025

I. Select topic

In a classroom environment, maintaining student concentration is an important factor to ensure effective learning. However, monitoring student concentration based solely on direct teacher observation can be inaccurate and lack objectivity. Factors such as fatigue, attention to external factors, or lack of concentration can affect the learning process.

Main issues:

- Difficulty in assessing student concentration levels based solely on direct observation.
- Lack of automated tools that can analyze and provide accurate information about student concentration levels in the classroom.
- Enhance teacher-student interaction by providing timely feedback on student concentration levels.

II. Conceptual design

AI Classroom Attention Detection Model:
This model will use image recognition (computer vision) and sentiment analysis (emotion recognition) technology to assess the level of students' attention in the classroom.

The main steps in the design include:

a. Image and video preprocessing:
   - Use a camera or webcam in the classroom to collect videos of students.
   - Video preprocessing to detect students' faces in each frame and extract features such as gaze, posture, and facial expression.

b. Attentive analysis through face and postures:
   - Facial recognition: Use a facial recognition model (e.g., OpenCV, dlib, or MTCNN) to detect and track students' faces.
   - Posture and gesture analysis: Determine students' posture (e.g., sitting upright, looking at the board).
   - Emotional Expression Analysis: Use an emotional analysis model (e.g. FER-2013 and EmoReact) to identify students' emotional expressions, thereby assessing their level of interest or fatigue.
   - Predicting the angle of the head relative to the camera (solvePnP).

c. Generative AI Recommendations:
   - **Study tips** tailored to the student's engagement levels.
   - **Break suggestions** based on detected fatigue or low attention spans.
   - **Motivational feedback**: If the system detects that a student is disengaged, GEN AI could offer encouragement and

suggest specific actions to re-engage them, such as asking questions or reviewing material.

- **Suggested learning strategies**: Based on the student's emotional expression (e.g., stress or frustration), the model might suggest relaxation techniques or different methods to approach learning.
- **Focus improvement tips**: For students detected to be frequently distracted, GEN AI could suggest time management techniques or suggest short activities to boost attention.

III. Flowchart

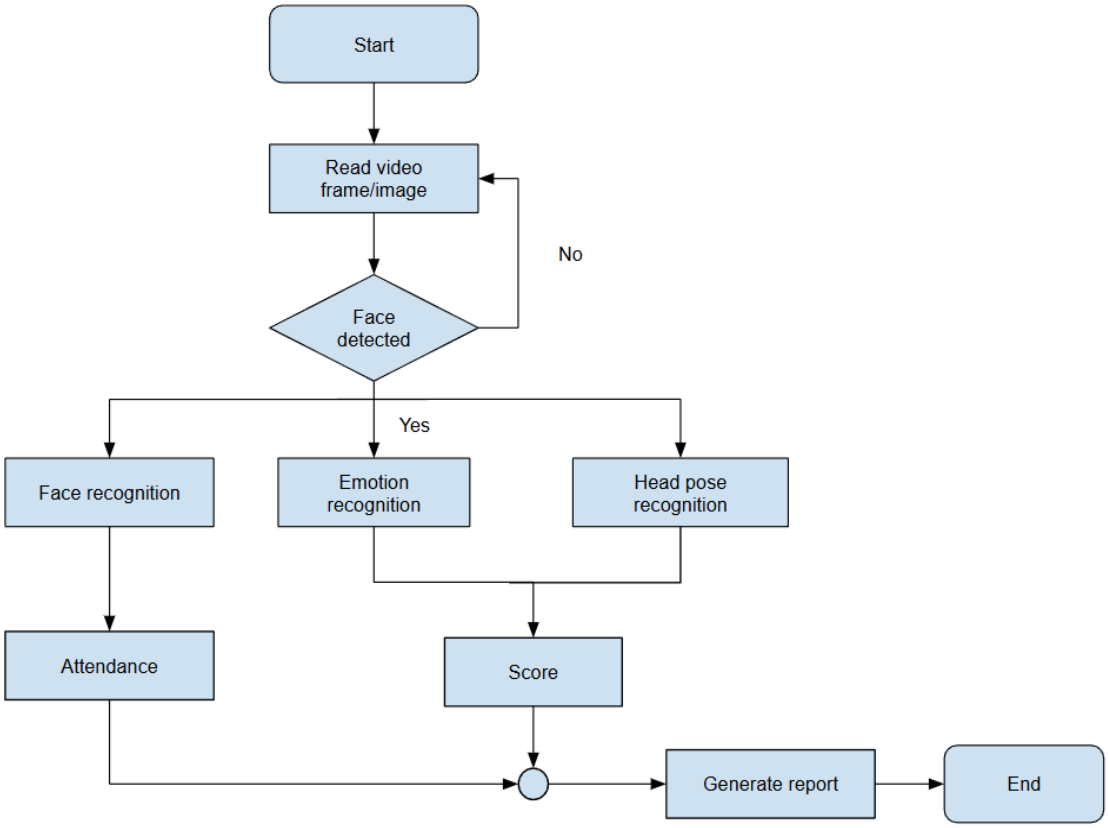


Fig 1: The flowchart of the project

IV. Database

- Database for Emotion Recognition:
  - **FER-2013** (Facial Expression Recognition): is a widely studied dataset and has been used in ICML competitions and many research papers. It is one of the more difficult datasets, with a Human-level accuracy of only about 65±5%. You can easily download it on Kaggle, and it consists of 35,887 images that have been normalized to 48x48 pixels and converted to grayscale. However, Fer-2013 is not a balanced dataset, with images of 7 facial expressions having the following distribution: Angry(4,953), Disgust(547), Fear(5,121), Happy (8,989), Sad (6,077), Surprise

(4,002), and Neutral (6,198).

- **CK+:** consists of 593 image sequences from 123 participants, aged 18 to 50. Each image sequence contains 10 to 60 frames of a participant moving from a neutral expression to a target expression. Each frame is approximately 640x480 pixels and has grayscale and/or color values.

- Also in this project we use the dataset from **merging FER-2013 data with CK+** including 57,200 pictures to have the same 48x48 pixel. However, it is not a balanced dataset, with images of 7 facial expressions having the following distribution: Angry(7,900), Disgust(1,400), Fear(8,200), Happy (14,400), Sad (9,500), Surprise (6,200), and Neutral (9,600). We split the dataset into the ratio of train - validation - test respectively 2 - 6 - 2.

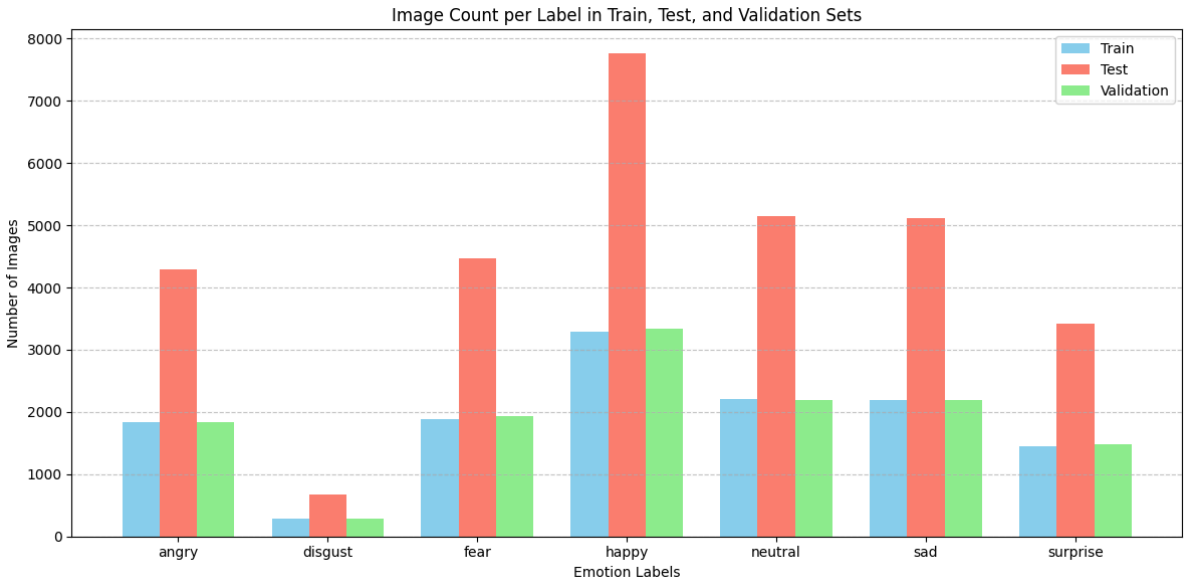


Fig 2: Dataset of combined with the CK+ and FER-2013

- Because the dataset size of disgust is very different from other types of datasets, our team will balance the datasets. We do this by transformations: rotate, reposition/scale, adjust color, flip image.

```python
transform_list = [
    transforms.RandomRotation(degrees=10),
    transforms.RandomAffine(degrees=0, translate=(0.05, 0.05), scale=(0.95, 1.05)),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3),
    transforms.RandomHorizontalFlip(p=0.5)
]
```

Fig 3: These functions transformation pictures

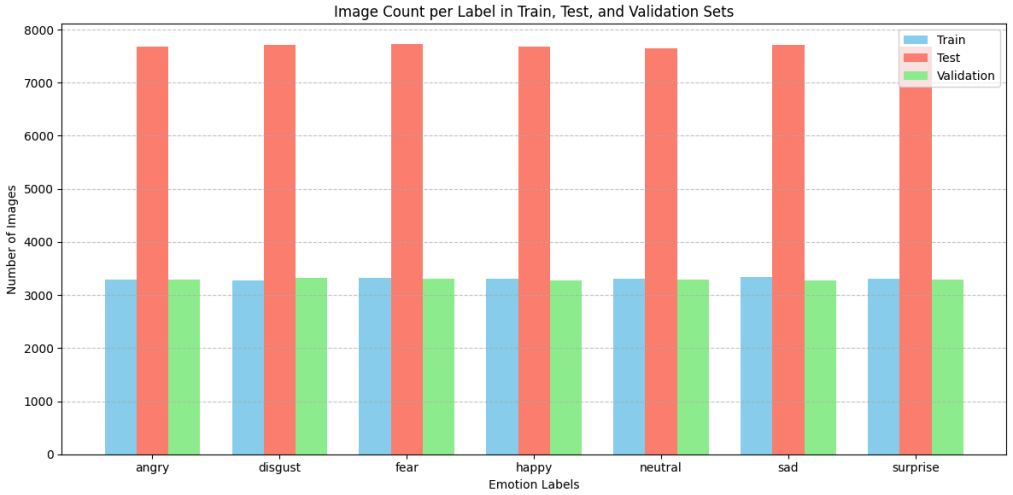


Fig 4: The number of dataset after baland

- Database for Class Emotion:
  - Providing input and output to train the Reported AI gen model is an indispensable task in supervised learning. We have designed our own emotional data in the classroom with many different cases in a class of 20 students. The data is created from more than 900 students, with each student having a confidence level of that emotion along with the time of expressing the emotion. The time of expressing the emotion is randomly distributed throughout the class. The output is manually evaluated with high accuracy. The data is divided into 3 parts, test, train, validation, with a ratio of 6:2:2.

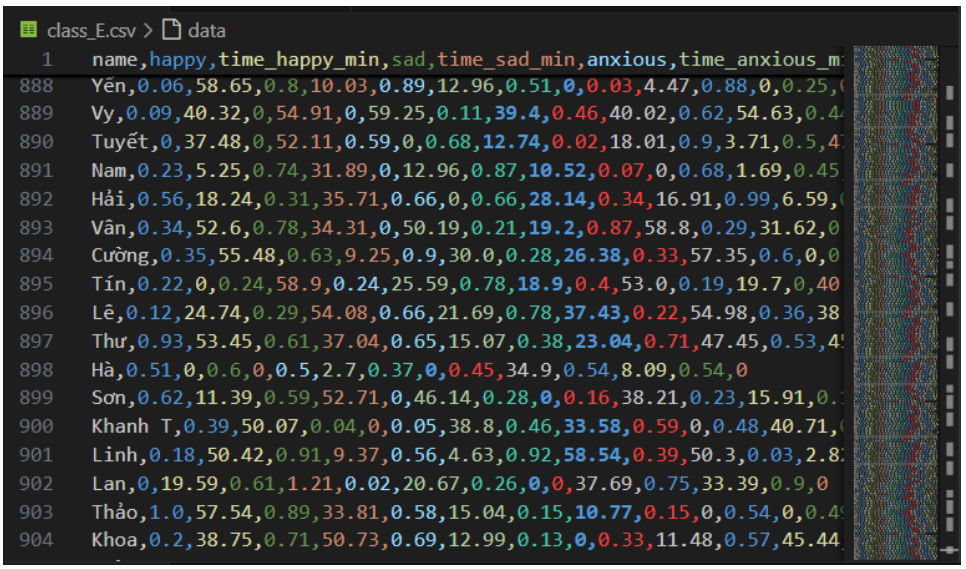


Fig 5: The amount of data for train Gen AI

V. Model:

1. Model for Face detection, recognition
   - Face detection
     - Conduct a survey of popular Face Detection models today. Including: MTCNN, CascadeClassifier, Haarcasde, Mediapipe. The survey is based on the criteria: Accuracy in difficult conditions, Computation speed, Computation resources, Detection distance, Ability to detect multiple faces at the same time.

Table 1: Decision Matrix of Face Detection Models

Decision Matrix (Face Detection)	MTCNN	Blaze Face	Haarcascade	Mediapipe
Accuracy in difficult conditions	4	3	3	4
Calculation speed	3	5	5	3
Computational resources	4	4	4	4
Detection distance	4	2	3	3
Ability to detect multiple faces at once	4	3	4	4
Average	3.8	3.4	3.6	3.6

- MTCNN has outstanding Face Detection capabilities and meets the given criteria. This is suitable for small and medium sized classroom settings.
- The module uses MTCNN (Multi-task Cascades Convolutional Networks) from the facenet_pytorch library for face detection and recognition.
  - Input:
    - Format: RGB image (convert from BGR to RGB using OpenCV)

```
rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

  - Preprocessing:

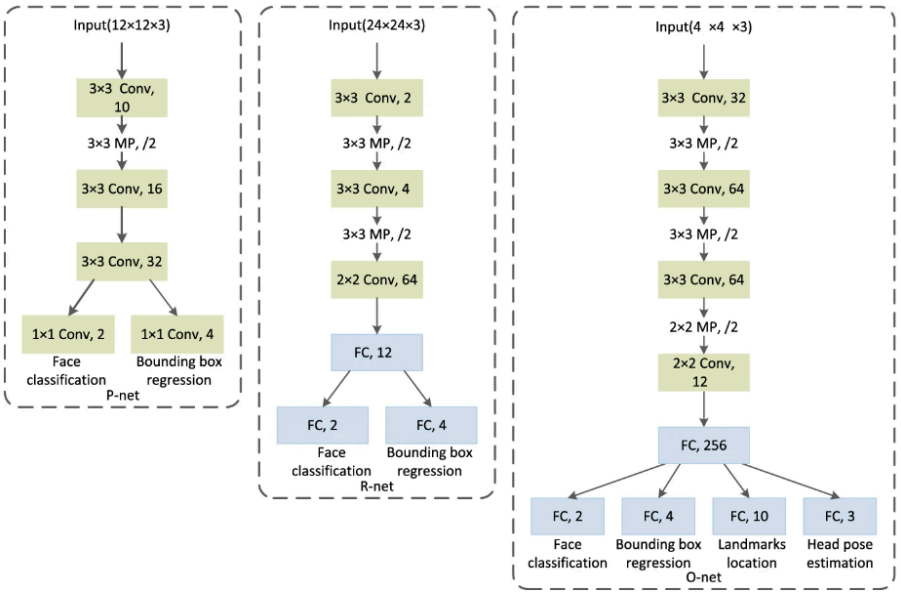


Fig 6: MTCNN model architecture[1]

- The MTCNN (Multi-task Cascaded Convolutional Neural Network) model performs face detection using three cascaded networks:
  - P-Net: Proposes candidate face regions.
  - R-Net: Filters out invalid boxes and refines them.
  - O-Net: Precisely predicts the face bounding box and five facial landmarks (eyes, nose, mouth corners).
- The input image is internally scaled into a pyramid to detect faces of various sizes.
- If any faces are found, MTCNN crops and resizes them (usually to 160×160), preparing them for the next step in face recognition.

- Output:
  - After processing, MTCNN returns the following:
    - Bounding boxes: Coordinates (x1, y1, x2, y2) for each detected face.

```
x1, y1, x2, y2 = [int(coord) for coord in boxes[i]]
x1, y1 = max(x1, 0), max(y1, 0)
x2, y2 = min(x2, frame.shape[1]), min(y2, frame.shape[0])
face_crop = frame[y1:y2, x1:x2]
if face_crop.size == 0:
    continue
```

    - Facial landmarks: Five key facial points (both eyes, nose, and mouth corners).
    - Cropped face tensors: These resized face crops can be passed directly to a recognition model such as Inception Resnet V1.

- Face Recognition
  - Conduct a survey of popular Face Recognition models today. Including: Resnet, MobiFace, ArcFace, FaceNet. The survey is

based on the criteria: Accuracy, Computation speed, Computation resources, Model complexity, Stability and reliability.

Table 2: Decision Matrix of Recognition Models

Decision Matrix (Face Recognition)	Inception Resnet V1	MobiFace	ArcFace	FaceNet
Accuracy	5	4	5	5
Calculation speed	4	4	4	5
Computational resources	5	2	4	5
Model complexity	5	4	5	4
Stability and reliability	5	3	4	4
Average	4.8	3.4	4.4	4.6

- Resnet V1 provides superior Face Recognition capabilities and meets the criteria.

2. Model for Emotion recognition
   a. Model Algorithms: (ResNet50)
   ResNet (Residual Networks) is a deep neural network architecture designed to address the problem of training very deep networks. It uses **skip connections** or **residual connections**, allowing the model to transmit information directly through the layers without degradation, thus facilitating training by mitigating issues like vanishing and exploding gradients.

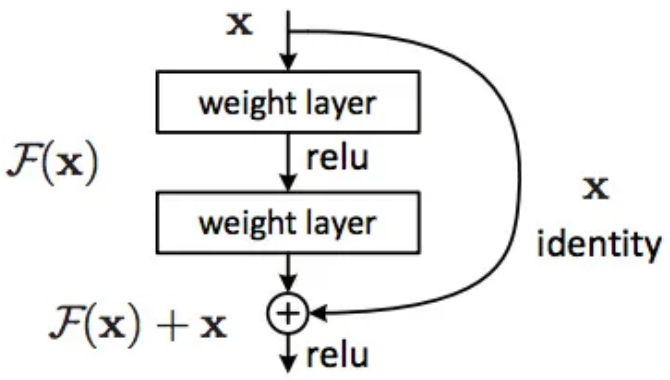


Fig 7. Skip Connection [ ]

In a neural network, a skip connection is simply a shortcut that allows the input of a layer to bypass one or more intermediate layers and be added to the output of another layer. In other words, rather than flowing sequentially through every layer, information can skip some layers and be directly added to the output of the current layer.

$$Y = F(x) + x$$

Where:

- x is the input to the layer.
- F(x) is the transformation (or operation) applied to x by the layer.
- Y is the sum of the original input x and the transformed output F(x)

Benefit of skip connection:

Vanishing Gradient Problem: In very deep networks, during backpropagation, gradients can become very small (i.e., vanish) as they are propagated back through each layer. This makes it very hard for the network to update weights effectively in deeper layers. Skip connections provide a direct path for gradients to flow through, helping maintain their magnitude during training, thus allowing deeper networks to learn more effectively.

Exploding Gradient Problem: On the flip side, gradients can also explode in very deep networks, meaning they become excessively large, leading to unstable weight updates. By allowing gradients to skip over certain layers, skip connections provide a form of **regularization**, which can help prevent this issue.

Easier Optimization: Skip connections make the optimization problem easier by allowing the network to learn an **identity mapping**. If learning the residual (i.e., the difference between the input and the output) is too difficult, the network can simply learn to pass the input forward unaltered. This helps avoid situations where the network struggles to learn the correct features, especially in deep networks.

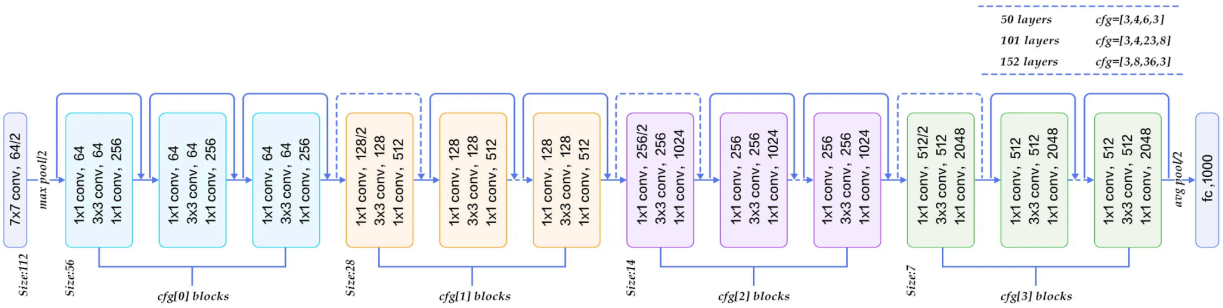


Fig 8. ResNet Architecture [3]

ResNet-50 is a variant of ResNet with 50 layers. The architecture of ResNet-50 consists of the following main components:

- **Convolutional Layers (Conv)**: These layers perform convolution operations on images.
- **Residual Blocks**: These blocks use skip connections to pass information through the layers.
- **Batch Normalization (BN)**: Helps stabilize the training process.
- **Fully Connected Layer**: Used for the final classification output

In ResNet-50, the layers of a residual block consist of:

- **Convolutional Layer**
- **Batch Normalization**
- **Activation (ReLU)**

The original input (often referred to as the skip or shortcut) is added directly to the output of the block.

If the dimensions of the input and output differ (for example, due to downsampling in the case of convolution), a 1x1 convolution is used to match the dimensions before the addition. This ensures that the addition operation is valid.

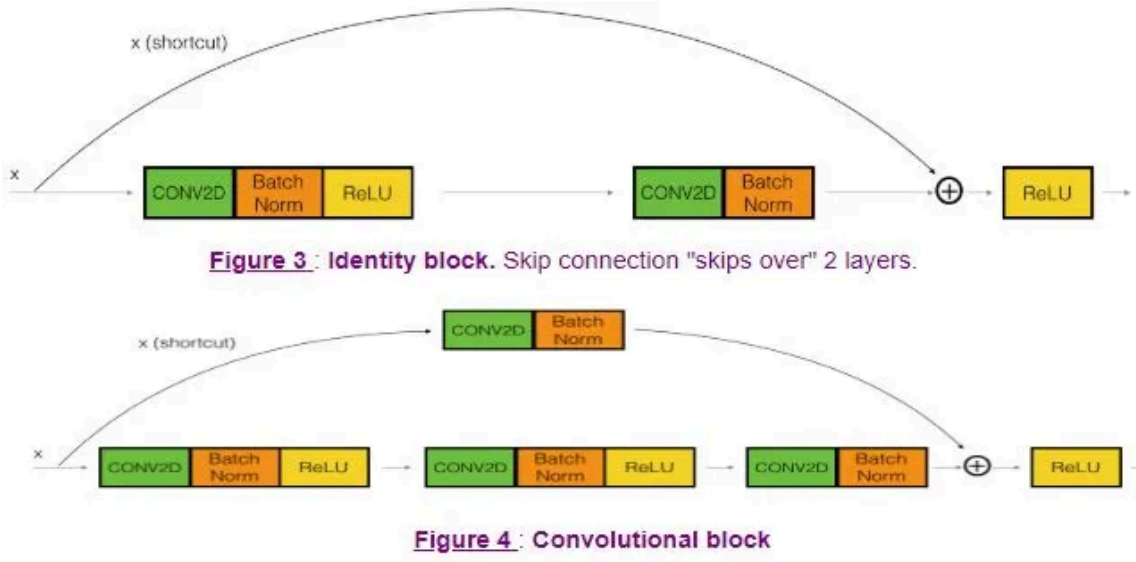


Fig9. Skip connection process [4]

The general structure of **ResNet-50** includes:

- 1 initial **conv1** layer with a 7x7 kernel.
- Residual blocks with convolution layers and **shortcut** connections to mitigate gradient issues.
- Finally, the model has a fully connected layer that outputs 7 classes for the emotion classification task (for FER-2013).

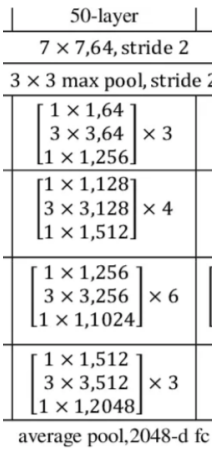


Fig 10. ResNet 50 layers

b. Training the Model

- **Dataset Preparation**
  - **Dataset**: The FER-2013 dataset used was combined with the Ck+ dataset, which consists of labeled facial images corresponding to emotions such as Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral.
  - **Preprocessing:** The images were converted to grayscale with a size of 48x48 pixels and normalized to the range [0,1].
  - **DataLoader Creation:** The `DataLoader` was used to load the dataset with a batch size of 64, employing Data Augmentation with transformations like RandomHorizontalFlip and RandomRotation.

```
#Check the number of data
print(f"Number of training samples: {len(train_dataset)}")
print(f"Number of valid samples: {len(val_dataset)}")
print(f"Number of test samples: {len(test_dataset)}")
print(f"Number of batches in train_loader: {len(train_loader)}")
```

```
Number of training samples: 53841
Number of valid samples: 23050
Number of test samples: 23138
Number of batches in train_loader: 842
```

```
#Check batch size
inputs, labels = next(iter(train_loader))
print(f"Input shape: {inputs.shape}")  # [32, 1, 48, 48]
print(f"Labels shape: {labels.shape}")  # [32]
```

```
Input shape: torch.Size([64, 1, 48, 48])
Labels shape: torch.Size([64])
```

- **Model Configuration**
  - **Pretrained ResNet-50**: The ResNet-50 model was pre-trained on ImageNet and used as the starting point for transfer learning.
  - **Adjusting Conv1 Layer**: For the FER-2013 dataset, the input images are grayscale (1 channel), so the `conv1` layer was replaced with a `Conv2d` layer accepting 1 input channel instead

of 3 (RGB).

```
model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
```

- **Fully Connected Layer**: The final fully connected layer was modified to output 7 classes for emotion classification.

```
# Change fully connected
num_ftrs = model.fc.in_features
# model.fc = nn.Linear(num_ftrs, 7)  # 7 emotions
model.fc = nn.Sequential(
    nn.Linear(num_ftrs, 256),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(256, 7)
)  # 7 label
```

- **Training Setup**
  - **Loss Function**: **CrossEntropyLoss** was used as the loss function, suitable for multi-class classification problems. Class weighting was used to balance the data.

```
#Load dataset
train_dataset = ImageFolder(root=train_dir, transform= train_transforms)
val_dataset = ImageFolder(root=val_dir, transform= test_transforms)
test_dataset = ImageFolder(root=test_dir, transform= test_transforms)

# Apply classweighting into train_dataset
class_weights = calculate_class_weights(train_dataset).to(device)
print(class_weights)
```

```
Class counts: {0: 7682, 1: 7713, 2: 7722, 3: 7676, 4: 7650, 5: 7713, 6: 7685}
Calculated class weights: [1.001246   0.99722177 0.9960595  1.0020286  1.0054342  0.99722177
 1.0008551 ]
```

  - **Optimizer**: The **Adam optimizer** was used with a learning rate of 0.001 and weight decay of 1e-4 to avoid overfitting.
  - **Learning Rate Scheduler**: A **StepLR scheduler** was employed to reduce the learning rate every 10 epochs.
  - **Early Stopping**: To prevent overfitting, the early stopping technique was used, which stops training if there's no

```
# Hàm mất mát
criterion = nn.CrossEntropyLoss(weight = class_weights)

# Bộ tối ưu hóa
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay = 1e-4)

# Scheduler
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.5)
```

improvement in validation accuracy for a set number of epochs.

- **Training Process**
  - The model was trained for 80 epochs.
  - The model's loss and accuracy were calculated for both the train and validation sets during each epoch.
  - Model performance was assessed using metrics such as accuracy, precision, recall, and F1-score and Confusion matrix

c. Result:

- **Evaluation Metrics**

- **Training Loss**: The loss gradually decreased across epochs, indicating that the model was learning and improving. In the lass epoch, loss in training set reduced to 0.7009, and in the validation set is 0.8272
- **Training Accuracy**: Achieved approximately 75%accuracy on the training set by the end of training.
- **Validation Accuracy**: The model achieved 74% accuracy on the validation set.
- **Precision, Recall, F1-Score**:

  The precision, recall, and F1 score are fairly balanced, with values around 73%, indicating that the model is doing a decent job at both identifying the correct labels and minimizing errors.

  Precision and recall are relatively close to each other, meaning that the model is not biased toward either false positives or false negatives.

  F1 score is in a good range, suggesting that the model is achieving a solid trade-off between precision and recall.

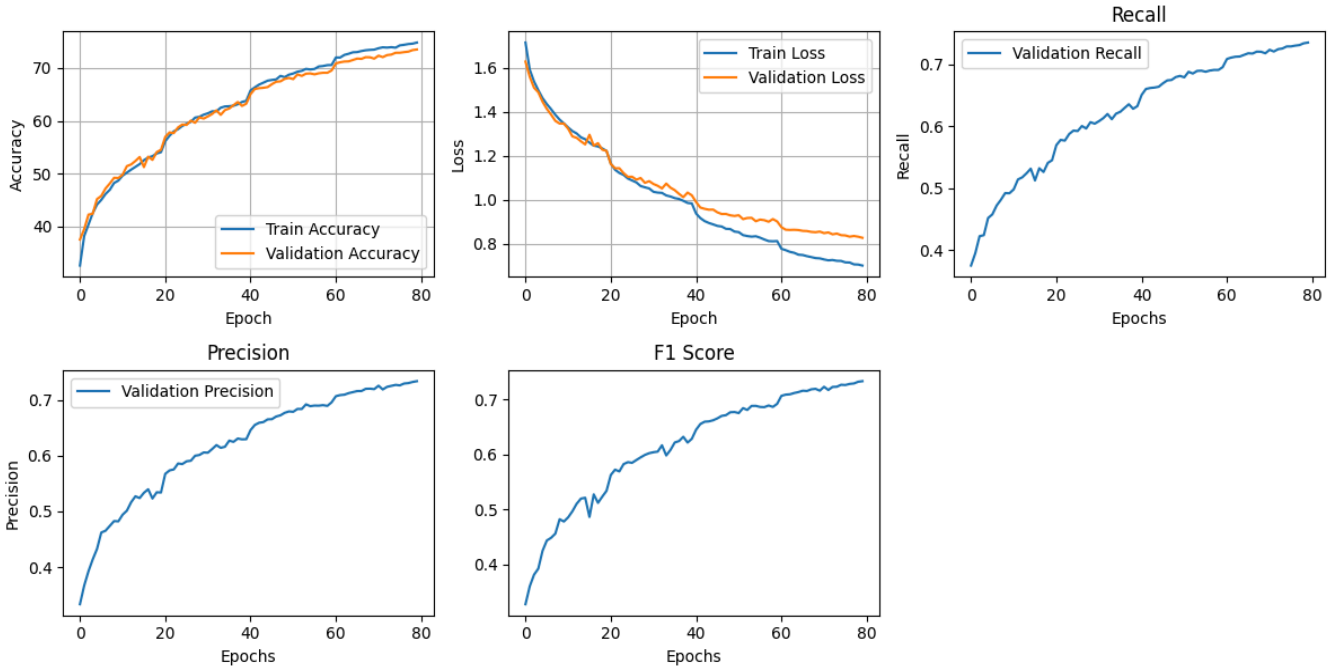


Fig 11. Result after training

- **Confusion Matrix**

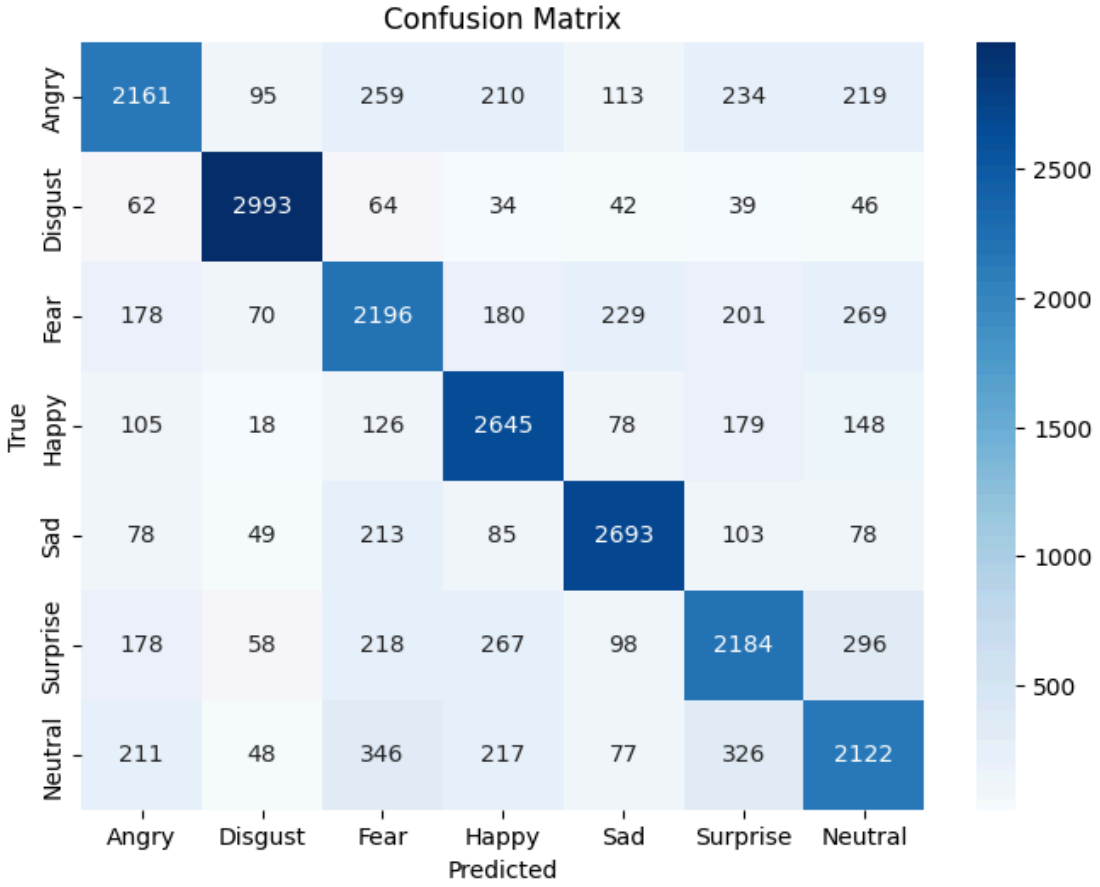


Fig 12. Confusion matrix

- The confusion matrix showed that the model correctly classified the majority of the emotions.
- **High Precision and Recall for "Disgust"**: The Disgust class has a high number of correct predictions (2993), and misclassifications to other emotions are relatively few. This suggests that the model is quite good at identifying disgust.

- **Lower Precision and Recall for "Surprise"**: The Surprise class has fewer correct predictions compared to others and a higher number of misclassifications to Fear and Neutral. This may indicate that the model struggles to distinguish surprise from other emotions.

- **Neutral and Sad**: The Neutral class seems to be misclassified into Sad, and Sad seems to be confused with Fear. This could be due to similar facial expressions in these emotions, which might lead to confusion.
    - **Model Applied on Video**
    - The model was applied to real-time emotion detection from facial expressions using a webcam.
    - **Results**: The model successfully identified emotions from video frames with high accuracy but encountered some misclassifications in low light conditions or when faces were obscured.

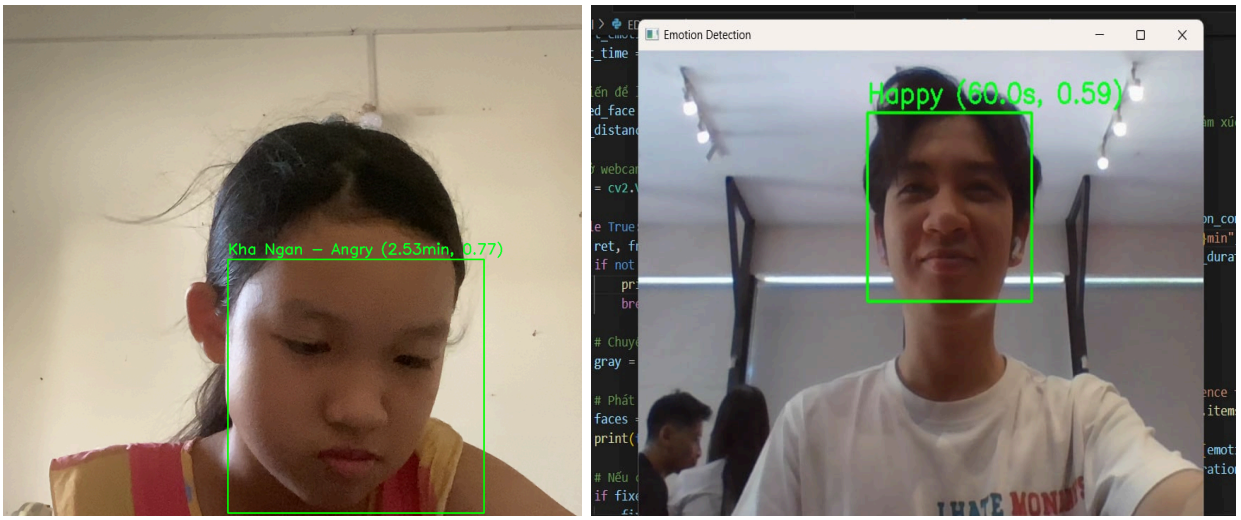


Fig 13. Apply model on video

d. Conclusion

The ResNet-50 model, after fine-tuning on the FER-2013 dataset, showed promising results for emotion classification. Although the model achieved high accuracy on both the training and validation sets, there is still room for improvement, particularly for rare emotions or in conditions like poor lighting.

Suggested Improvements:

- **Data Augmentation**: More diverse data augmentation techniques such as zooming, lighting adjustments, or rotating facial orientations could help make the model more robust.
- **Hyperparameter Tuning**: Adjusting hyperparameters such as learning rate, batch size, or weight decay could further optimize the model.
- **Ensemble Methods**: Implementing ensemble methods like bagging or boosting could improve the model's performance by combining multiple models to make more accurate predictions.

This model can be deployed in real-world applications such as psychological analysis, intelligent user interfaces, or emotion recognition in videos.

3. Model GEN AI

a. Model T5-base architecture

- What is the T-5 Model?

  The T5 architecture comprises a stack of transformer encoder-decoder layers, each layer iteratively processing input text to capture contextual information and provide meaningful representations. These interconnected layers allow for efficient information flow and hierarchical representation learning. T5 delivers cutting-edge performance across several NLP benchmarks while preserving a simple and scalable architecture.
  T5 was chosen for fine-tuning due to its versatile text-to-text framework, which allows it to handle a wide range of NLP tasks (e.g., summarization, translation, question answering) with minimal adjustments. Unlike GPT, which is focused on autoregressive text generation, or BERT, which excels at text understanding but lacks generative capabilities, T5's encoder-decoder architecture makes it suitable for both understanding and generating text. Its pre-trained model can be fine-tuned across various tasks with relatively less data, making it highly flexible and efficient for diverse NLP applications.

- T-5 base ArchitectureThe:

  T5 model architecture uses an encoder-decoder structure for text-to-text tasks. The input text is first converted into embeddings, with positional encoding added to capture word order. The encoder layers (12 in total) process the input through self-attention and feed-forward networks, generating hidden representations. These

are then passed through output embeddings before reaching the decoder layers (12), which generate the final output (e.g., a summary) using both self-attention and cross-attention with the encoder. The model transforms input text into output text through this process.

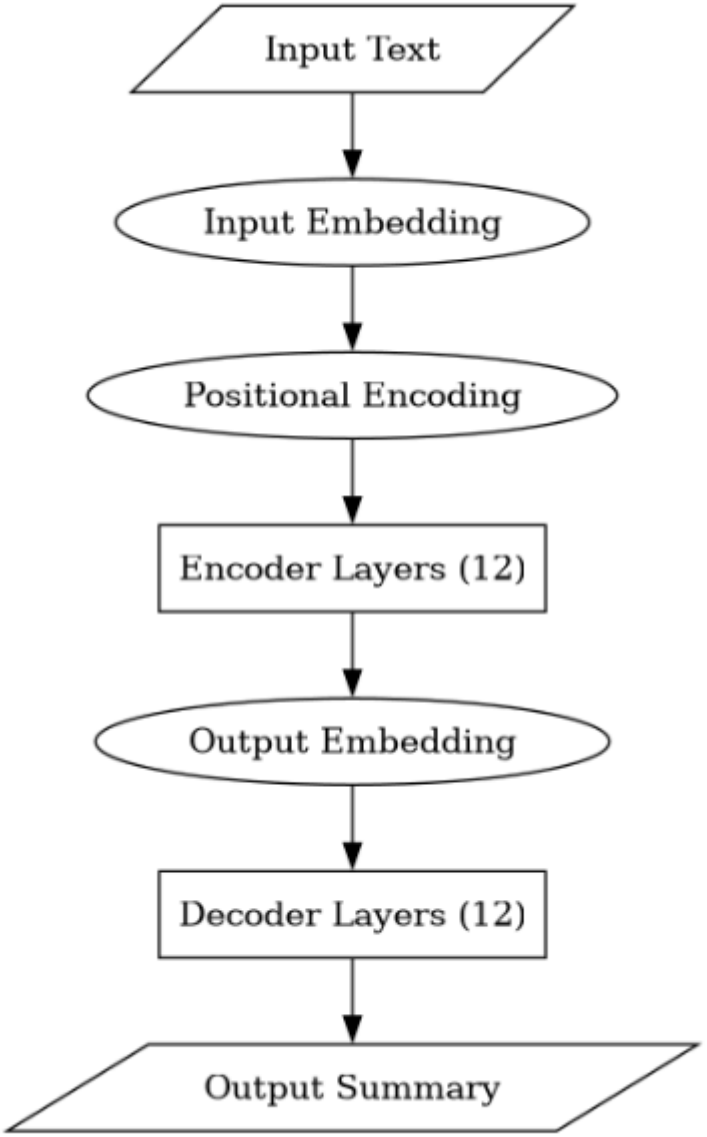


Fig 14: Architecture of T5 model[2]

- Comparison with Other T-5 Models[2]

Model	Encoder Layers	Decoder Layers	Parameters
T5-Small	6	6	~60 million
T5-Base	12	12	~220 million
T5-Large	24	24	~770 million
T5-3B	24	24	~3 billion
T5-11B	24	24	~11 billion

b. Training Model (fine-tune)

- Dataset Preparation:

  Data Collection: The dataset is crucial for fine-tuning the T5 model. For tasks like text summarization, the dataset could be sourced from publicly available collections such as CNN/Daily Mail or XSum. For other tasks like question answering, datasets such as SQuAD or Natural Questions are suitable. The data should consist of pairs of input text and target output (e.g., long articles and corresponding summaries for summarization).

  Data Splitting: The dataset needs to be split into three subsets: Training Set, Validation Set, and Test Set. A common split ratio is 60% for training, 20% for validation, and 20% for testing. This ensures that the model is properly trained, evaluated, and tested on separate data to avoid overfitting.

  Text-to-Text Format: T5 operates in a text-to-text format, where both the input and output are treated as textual sequences. For instance, in summarization, the input would be a lengthy article, and the output would be its summary. This consistent text-to-text format allows T5 to be versatile across a wide range of NLP tasks.

- Preprocessing:

  Manual Data Alignment: I carefully aligned the input and output pairs to ensure that each example in the dataset represented the correct relationship. This step was crucial for tasks where context and specific output are required. For example, I verified that the summaries were concise and accurately reflected the main points of the original texts, ensuring that the model would learn to generate summaries with similar properties.

  Tokenization & Padding: To prepare the data for input into the model, I tokenized the input and output texts using T5Tokenizer.

Tokenization ensures that the text is split into smaller pieces (tokens), making it easier for the model to understand and process. I also ensured that padding was applied where necessary to handle sequences of different lengths and maintain uniformity.

Data Cleaning & Normalization: To ensure the model only learns relevant patterns, I manually cleaned the dataset, removing noise such as irrelevant information, duplicate examples, or malformed text. I also performed text normalization, including lowercasing, removing unwanted characters, and handling punctuation correctly. This ensured consistency across the dataset, allowing the model to focus on the key patterns for learning.


```
    },
    {
      "input_text": "Phuong: happy=0.45 (5.28min); sad=0.76 (10.0min); anxious=0.45 (7.4min); excited=0.95 (0.68min); bored=0.19 (5.38min); confident=0.55 (8.55min); tired=0.2 (7.71min) | Tien: happy=0.42 (9.06min); sad=0.
      "report_text": "Overall, the class is doing well, especially Thanh. Attention is needed for My and Phuong to improve their mood."
    },
    {
      "input_text": "Son: happy=0.53 (6.04min); sad=0.42 (6.39min); anxious=0.54 (6.42min); excited=0.79 (8.83min); bored=0.0 (2.53min); confident=0.27 (7.76min); tired=0.87 (7.02min) | Quoc: happy=0.13 (11.48min); sad=0.4
      "report_text": "Quoc shines with enthusiasm, while Son and Kien show signs of fatigue and need some motivation."
    },
    {
      "input_text": "Trung: happy=0.11 (5.02min); sad=0.87 (3.27min); anxious=0.24 (2.48min); excited=0.39 (8.4min); bored=0.78 (4.5min); confident=0.62 (9.56min); tired=0.58 (11.77min) | Tram: happy=0.61 (0.35min); sad=0.
      "report_text": "The class shows a positive atmosphere with Le standing out. However, Ai and Minh might need some additional support."
    }
```


Fig 15 : Training Data Input-Output Pairs

- Model configuration:
  - Training Arguments:

    output_dir: Set to `"/kaggle/working/t5-class-report"`. This directory is where the model checkpoints and logs will be saved. This is important for tracking the training progress and retrieving the model after training.

    num_train_epochs: Set to 15. A higher number of epochs (15 in this case) ensures the model has enough opportunities to learn from the data. Since the model is being fine-tuned, more epochs help refine the pre-trained model to adapt better to the specific task.

    per_device_train_batch_size and per_device_eval_batch_size: Both are set to 4. A batch size of 4 is used to balance between training time and memory constraints, especially when working with large models like T5. A smaller batch size is often necessary when GPU memory is limited.

    learning_rate: Set to 2e-5 (0.00002). A small learning rate is used when fine-tuning a pre-trained model to prevent it from making too large weight updates, which could disrupt the knowledge it has learned previously. This helps in maintaining stability during training.

    warmup_steps: Set to 10. This gradual warmup of the learning rate helps avoid instability early in training, which is especially important when fine-tuning large models.

    weight_decay: Set to 0.01. This regularization parameter helps prevent overfitting by penalizing large weight values, encouraging the model to generalize better.

logging_steps: Set to 10. Logging the progress every 10 steps allows for monitoring the training process and ensuring that the model is learning as expected. It's useful for debugging or tracking training performance.

save_total_limit: Set to 1. This ensures only the latest model checkpoint is saved, conserving storage space and preventing excessive disk usage.

- Trainer Configuration:

  model: This is the pre-trained T5 model that will be fine-tuned using the above settings.

  train_dataset and eval_dataset: These datasets hold the training and validation data respectively. They must be preprocessed and tokenized before training begins.

  data_collator: The `collate_fn` is used to combine individual samples into batches. It ensures that all sequences in the batch are of equal length, typically through padding.

  tokenizer: The same tokenizer used for preprocessing the input text is passed here. It's essential that the tokenizer is consistent between training and inference.

- Training:

  trainer.train(): This starts the fine-tuning process. The model will train for the specified 15 epochs using the parameters defined in `TrainingArguments`.

  trainer.save_model(): After training completes, the model is saved to the specified directory (`"/kaggle/working/t5-class-report-final"`).

c. Result:

The failure in generating the report can be attributed to the limitations in knowledge, training data, computational resources, and the training duration. These constraints likely impacted the model's ability to learn effectively and produce accurate results. Moving forward, addressing these limitations by improving the quality and quantity of the data, increasing computational resources, and extending the training duration could help achieve better performance.

Fig 16: Result of Gen_reported_model

d. Suggestions for Next Steps:

Given that the model has successfully generated a report (as indicated by the output text), the focus should shift away from the WandB logging issue, especially since the issue appears to be external to the model's core functionality.

Switch to Hugging Face API for Report Generation:

Instead of relying on WandB for logging or server-based tracking, the **Hugging Face API** can be used to retrieve or interact with the generated report. Hugging Face offers an easy-to-use, free API with a generous number of requests, making it an ideal choice for accessing model outputs. You can use this API to send the generated report data to an endpoint where it can be processed or stored in a desired location, such as a file system or a database.

This transition involves setting up a simple **GET API** endpoint that will accept the generated report, most likely in **JSON** or **text format**, and allow for retrieval from the server. Hugging Face's API will help with easy integration of model inference and report retrieval without needing to manage external services.

Set Up an API Endpoint for Report Retrieval:

Hugging Face provides a **transformers** library to interact with their hosted models, making it easy to send requests to fetch the generated report. Once the model generates the report, it can be stored locally or in a database, then made accessible via an API endpoint.

For example, after running the model **DeepSeek-R1-Distill-Qwen-1.5B**, the report can be saved as a text file or in a database. The API can be set up to access this saved report and return it to the user upon request.

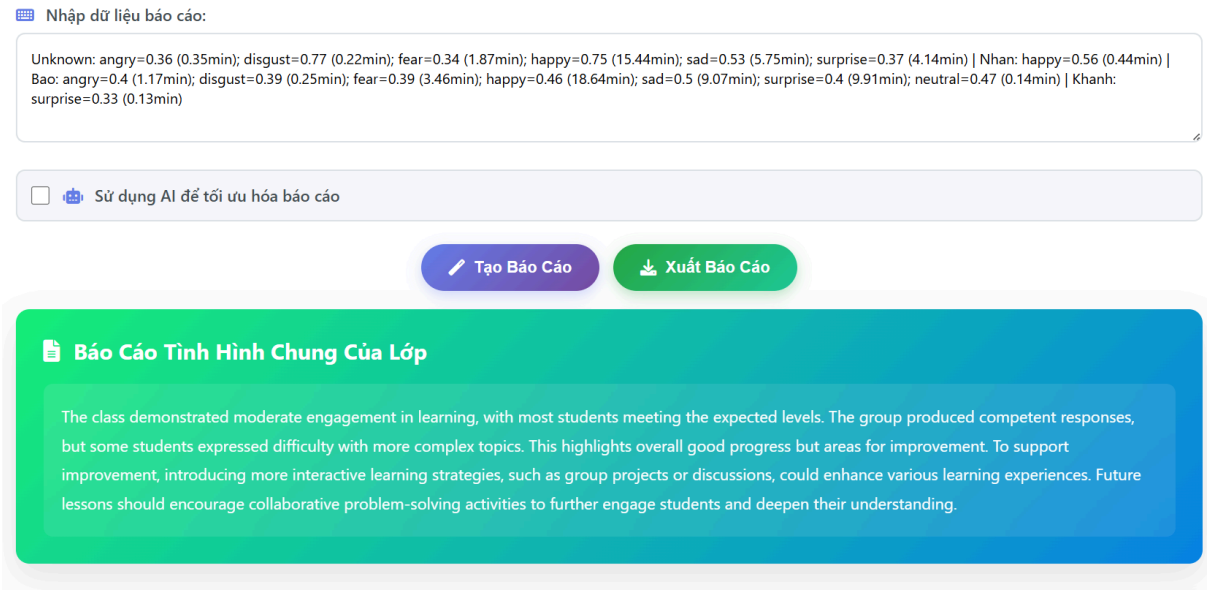
Handle Report Generation and API Call:

After generating the report with the **DeepSeek-R1-Distill-Qwen-1.5B** model, instead of logging the results to WandB, save the report to a local file or a database and trigger the **Hugging Face API** to retrieve and display the report. By using Hugging Face's API, you can ensure that the model’s output is easily accessible without depending on external services that may cause failures like the one previously observed.

This approach simplifies the deployment, reduces potential issues, and provides a flexible, scalable solution for report generation and retrieval

e. Gen_reported result:

The successful integration with the Hugging Face API marks a significant achievement in streamlining the model inference and report retrieval process. By utilizing Hugging Face's free and user-friendly API, we have bypassed technical and resource limitations, making the workflow more reliable and efficient. This solution enables the creation of generative reports for classroom use, overcoming challenges while ensuring a smooth and accessible process.



VI. Result project

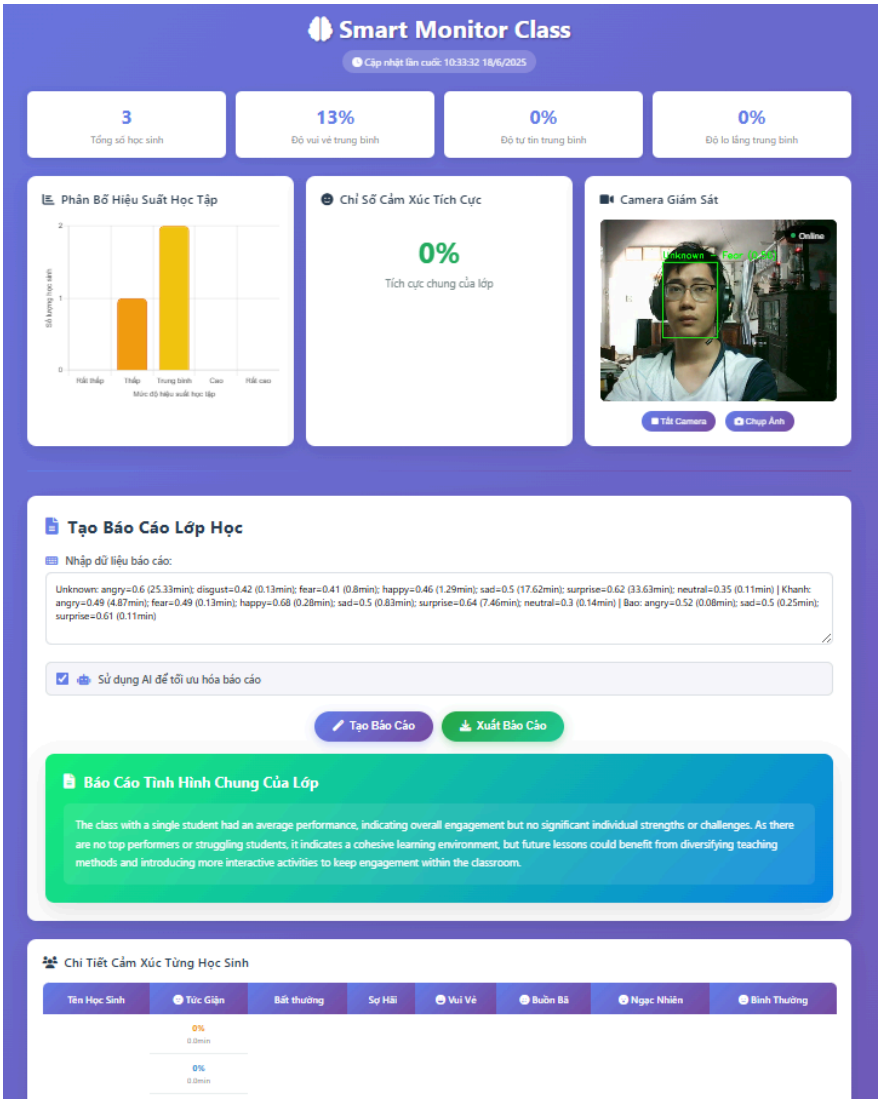


Fig 17 The result of project

VII. Web Application Design:

1. Frontend: User Interface

The frontend of the "Student Emotion Analysis Dashboard" application is designed with the main goal of being intuitive, modern and easy to use to monitor the emotional state of students in the classroom through cameras and data charts. The interface is oriented towards the best user experience, while supporting efficient retrieval and report generation.

a. Technology:

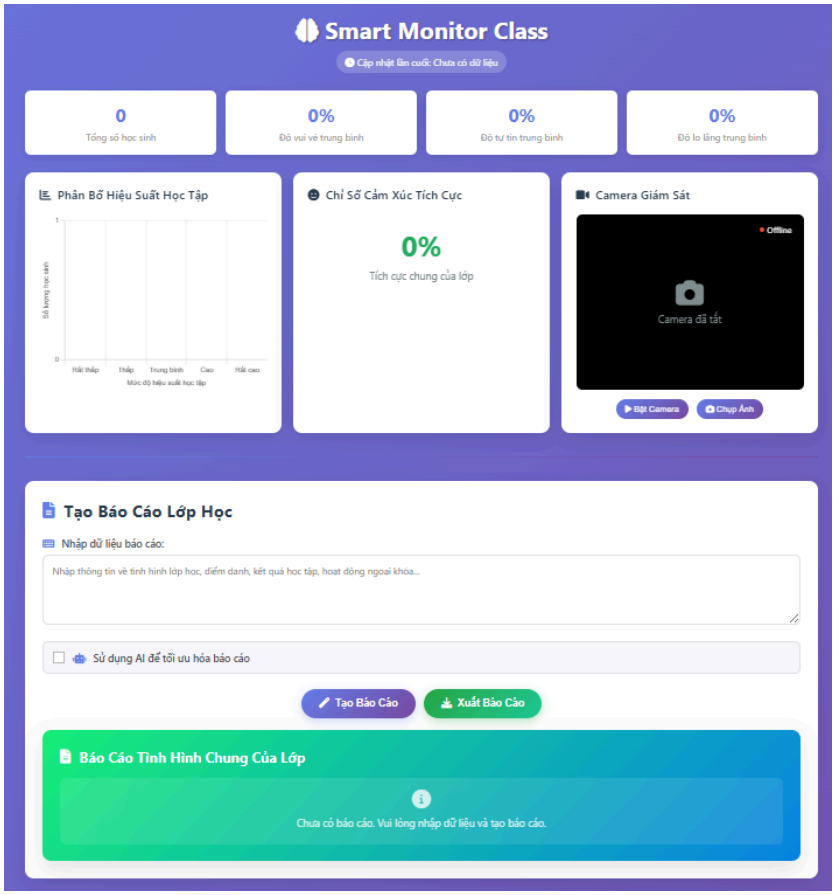
- HTML5: create website content structure.
- CSS3: design user interface in modern, responsive style.
- JavaScript (Vanilla): control interaction and dynamic data update.
- Chart.js: draw student performance charts.
- FontAwesome: display intuitive icons, increase liveliness for the dashboard.

b. Interface structure:

- Title & update time: displays dashboard name and last update time.
- Quick stats: displays number of students, average happiness, confidence and anxiety as cards.
- 3-column dashboard: Learning performance chart, class positive emotion index, surveillance camera interface and on/off/photo capture control.
- Report generation section: Users enter data or choose to use AI to generate class reports, there is a button to export the report in text format.
- Student detail table: displays each student's emotions horizontally (angry, happy, sad, surprised,...)
- Data refresh button: updates student emotion data in real time.

c. Interactive features:

- Surveillance camera: Display camera status (online/offline), allows to turn on/off and take photos right on the interface.
- Automatically download emotion data via fetch from API /api/emotions.
- Automatically generate charts, update emotion index and student table.
- Highly responsive: displays well on both large screens and mobile devices.



2. Backend:

The system's backend is built through APIs and data streams, which provide the foundation for the frontend to function efficiently, making it easy for teachers to monitor student emotions in real time.

- In the web application analyzing student emotions, the backend is the part responsible for processing: Recognizing student faces via camera. - Predicting emotions using deep learning models (ResNet50). - Recording emotion data into JSON files. - Extracting and synthesizing data into class reports. - Serving API for frontend to display charts, data tables. - Providing camera streams for the interface.

a. Main backend structure:

- Image processing library/model: OpenCV, PyTorch, MTCNN, ResNet50.
- Main processing flow:
  - **/video_feed**: Returns video stream from camera for frontend to display.
  - **/api/emotions**: API returns student emotion data (in JSON format).
    - Each student has a name and a list of emotions with a score + time.
    - The function generates a string like:
    - Bao : happy=0.8 (5 min); sad=0.1 (1 min) | Nhan: ...
    - This data is displayed on the frontend and passed to the AI to generate the report.
  - **/get_input_text**: Small API to get input_text to automatically update into report form.
  - check_data_freshness(): Check if sentiment data is new → re-parse and re-generate report using AI.
    - Compare get_model_output() with current_model_output.
    - If different → update:
    - cached_parsed_data: parsed JSON data.
    - cached_report: text report from AI.
    - last_update_time: updated timestamp.→ Helps ensure the frontend always displays the latest data.
  - **generate_frames()**: processes video from camera, recognizes faces, predicts emotions, saves results. Use cv2.VideoCapture(0) to get frames from the webcam. Use MTCNN to detect faces and InceptionResNetV1 to identify identities (if same as the saved person). Use a retrained ResNet50 model to predict emotions from faces (48x48 grayscale images). Calculate the time of emotional appearance, accumulate confidence scores. Display label + frames, then encode into MJPEG stream to return to the browser.

b. Communication between Backend and Frontend:

Communicate	frontend	backend
Student emotional data	JavaScript calls fetch('/api/emotions') periodically	@app.route("/api/emotions") returns JSON
Class report data	User enters form + chooses to use AI	@app.route("/") handles POST, generates report
Surveillance camera	Call interface <img src="/video_feed">	@app.route("/video_feed") send MJPEG image
Refresh data	The “Refresh” button calls loadData()	Backend reads JSON file, updates data
Automatic report generation	“Create Report” button POST form	Call StudentEmotionAnalyzer.analyze_and_generate_report(...)

VIII. Team mission

	Member	Mission	%
1	Phan Đinh Khanh	Preparing dataset and detect face	25%
2	Nguyen The Nhan	Implementing model and training ResNet50	30%
3	Tran Phuoc Dien	GEN AI	25%
4	Phan Ba Nguyen Bao	Web Application Design	20%

Reference

[1] https://huytranvan2010.github.io/Architecture-MTCNN/

[2]: https://www.analyticsvidhya.com/

[3]: ResNet50. ResNet-50 is a convolutional neural… | by Aditi Rastogi | Dev Genius

[4] https://blog.devgenius.io/resnet50-6b42934db43