**Student-Surveillance**

**ScholarWatch: AI-Powered Student Surveillance System**

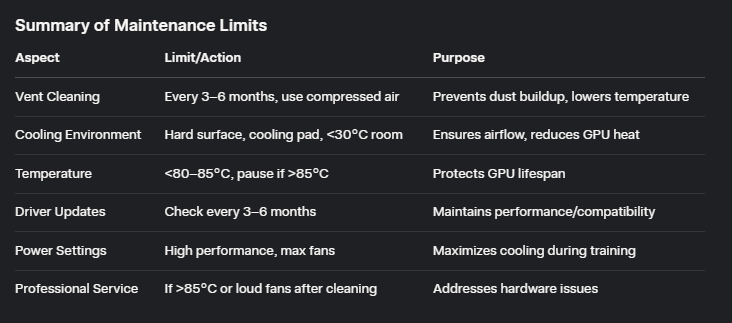
* Plan:- Create 3 models Smartphone detection, Face recognition, Head pose estimation and merge all three into one to monitor students from a cctv feed. It will monitor students and see if a particular student is using mobile phone or if he is peeking left right etc.
* Jupyer notebook will be my work environment with python 3,10 and tensorflow pytorch for gpu
* TensorFlow version: 2.10.0
* Torch version: 2.7.0+cu118

Setup

1. Created a venv student-surveillance-venv and activated it

student-surveillance-venv\Scripts\activate

1. Installed juptyter notebook and ipykernel
2. Register the venv as new jupyter kernel
3. Create jupyter notebook and test it
4. Limitations



1. Initialize git

**Face Detection & Recognition**

1. Verify Dependencies
2. Loading and Preparing the Custom Dataset

* Load the labels.csv file to map IDs to names.
* Read images from the train, test, and validate folders.
* Use MTCNN to detect and crop faces from the images.
* Preprocess the images (resize to 160x160, normalize, etc.) for compatibility with a face recognition model (e.g., a model based on FaceNet or a custom CNN).
* Organize the data into a format suitable for training (e.g., PyTorch Dataset for easy batching).

3. Define a Pre-trained ResNet Model for Fine-Tuning(Use a pre-trained ResNet-18 model from torchvision.

We’ll:

1. Use a pre-trained ResNet-18 model from torchvision.
2. Replace the final fully connected layer to output probabilities for your 4 classes (Manika, Rochan, Sayali, Shuchi).
3. Freeze the earlier layers initially (to leverage pre-trained weights) and fine-tune the last few layers.
4. Define the loss function (cross-entropy) and optimizer (Adam with a smaller learning rate for fine-tuning).

* **Model Architecture**:
  + **ResNet-18**: A pre-trained ResNet-18 model with 18 layers (convolutional and fully connected). It’s pre-trained on ImageNet, so it has learned general features like edges, textures, and shapes.
  + **Freezing Layers**: We freeze all layers except the final fc layer to preserve the pre-trained weights. This is especially helpful with a small dataset (121 images) to prevent overfitting.
  + **Modified fc Layer**: The original fully connected layer (512 → 1000 for ImageNet) is replaced with 512 → 512 → 4, adding a ReLU and dropout (0.5) to reduce overfitting.
* **Loss Function**: CrossEntropyLoss for multi-class classification (4 classes).
* **Optimizer**: Adam with a smaller learning rate (0.0001) since we’re fine-tuning a pre-trained model. We only optimize the parameters that require gradients (i.e., the fc layer).
* **Scalability**: As your dataset grows (e.g., to tens of thousands of images), you can:
  + Unfreeze more layers (e.g., model.resnet.layer4) to fine-tune deeper parts of the network.
  + Switch to a deeper model like ResNet-50.
  + Add data augmentation to improve generalization (we’ll do this in the next step).

**4.**

**Step 4: Training the Model**

Now that the model is defined, we’ll:

1. Add data augmentation to improve generalization, especially since your dataset is small (121 training images) but will grow in the future.
2. Set up a training loop to fine-tune the model on your training set.
3. Evaluate performance on the validation set during training to monitor progress.
4. Save the trained model for future use.

 **Data Augmentation**:

* RandomHorizontalFlip(): Randomly flips images horizontally (50% chance).
* RandomRotation(10): Rotates images by up to ±10 degrees.
* ColorJitter(): Randomly adjusts brightness, contrast, and saturation to simulate lighting variations.
* These augmentations are applied only to the training set to make the model more robust.

 **Training Loop**:

* Trains for 20 epochs (you can adjust num\_epochs if needed).
* Computes training loss and accuracy per epoch.
* Evaluates on the validation set after each epoch.
* Saves the model with the best validation accuracy to best\_model.pth.

 **Scalability**:

* As your dataset grows, you can increase num\_epochs, unfreeze more layers (e.g., model.resnet.layer4), or switch to a deeper model like ResNet-50.
* Data augmentation helps the model generalize better as the dataset scales.
* Validation Loss: 0.8706, Validation Accuracy: 79.41%
* Validation Time: 95.62 seconds
* Training complete.
* Best Validation Accuracy: 82.35%

Step 5: Evaluating the Model on the Test Set and Setting Up Inference

Loaded 21 images for test split

Failed detections in test set: 0

Test Loss: 0.9494, Test Accuracy: 76.19%

6. Deploying the Model for Real-Time Use

**IMP Links**

1. [yakhyo/head-pose-estimation: 👤 | Real Time Head Pose Estimation: Accurate head pose estimation using ResNet 18/34/50 and MobileNet V2/V3 models. Evaluate yaw, pitch, and roll with pre-trained weights for quick integration.](https://github.com/yakhyo/head-pose-estimation)

2. [Real-time 6DoF full-range markerless head pose estimation](https://repository.hanyang.ac.kr/bitstream/20.500.11754/187508/1/110298_%EC%9D%B4%EC%84%B1%EC%98%A8.pdf)

3. [Mobile phone detection Dataset > Overview](https://universe.roboflow.com/exam-detection-a9bsf/mobile-phone-detection-mtsje)