Real Time Mask Detection With CNN

The project of the face mask detection system is designed to leverage deep learning and computer vision techniques to identify whether individuals are wearing face masks in real-time video streams. The system consists of three primary components: a face detection model, a face mask detection model, and a user interface for real-time feedback.

Initially, the face detection model utilizes a pre-trained Convolutional Neural Network (CNN) architecture, specifically a Single Shot Multibox Detector (SSD) based on the MobileNet framework. This model is trained to identify and localize faces within a given frame from the video stream. When a frame is captured, it is preprocessed by resizing and normalizing it before being passed through the face detection model. The output of this model provides bounding box coordinates around detected faces and associated confidence scores, filtering out low-confidence detections to ensure reliability.

Once faces are detected, the system proceeds to the face mask detection model, which is a separate CNN specifically trained to classify faces as "mask" or "no mask." Each detected face region is extracted from the frame, resized to the input dimensions of the mask detection model, and normalized for optimal prediction accuracy. The model outputs probabilities indicating whether a mask is present. The results are visually overlaid on the video feed, providing real-time feedback to users. By integrating these components into a Streamlit application, the system becomes user-friendly, allowing easy interaction through start and stop controls while delivering timely mask compliance insights. This system effectively combines advanced image processing techniques and deep learning to address public health challenges in an accessible format.

Additionally, The Streamlit app serves as the front end for this system, providing an intuitive interface that enables users to start and stop the video stream effortlessly. This user-centric design enhances the practicality of the mask detection system, ensuring that individuals can easily access the technology in various environments, such as schools, workplaces, or public events. The seamless integration of the deep learning models with Streamlit not only facilitates real-time monitoring but also makes the deployment of this important public health tool straightforward and efficient.

Preprocessing Images and Training Model

Importing Libraries:

The project uses several key libraries like TensorFlow and Keras for deep learning, sklearn for data preprocessing and evaluation, and matplotlib for plotting the results.

MobileNetV2 is used as the base model for transfer learning, which is a pre-trained model on a large dataset (ImageNet).

Setting Hyperparameters:

The learning rate INIT_LR = 1e-4 controls how much to adjust the model's weights after each iteration.

EPOCHS = 20 is the number of times the model will go through the entire training data.

BS = 32 is the batch size, meaning the model processes 32 images at a time.

Loading and Preprocessing Images:

Data set consists of 7553 RGB images in 2 folders as with_mask and without_mask. Images are named as label with_mask and without_mask. Images of faces with mask are 3725 and images of faces without mask are 3828.

The images are organized in two categories: "with_mask" and "without_mask."

All images are resized to 224x224 pixels to fit the input size expected by MobileNetV2.

Each image is converted into an array and normalized by using the preprocess_input function, ensuring the pixel values are in the range suitable for the model.

Label Encoding and Splitting Data:

The labels are one-hot encoded, meaning they are transformed into a binary format (0 for "without_mask" and 1 for "with_mask").

The dataset is split into 80% training data and 20% test data to evaluate the model later on.

Data Augmentation:

Data augmentation is applied to the training data to artificially increase the diversity of the dataset. It randomly rotates, zooms, and shifts images, which helps the model generalize better.

Building the Model:

Transfer Learning: MobileNetV2, which is pre-trained on ImageNet, is used as the base model. This allows us to leverage the knowledge of a powerful model without training from scratch.

Custom Head: On top of MobileNetV2, we add custom layers (including average pooling, fully connected layers, and a final softmax output layer) to adapt it for our specific task (mask/no mask classification).

Compiling the Model:

The model is compiled using the Adam optimizer and binary_crossentropy as the loss function since this is a binary classification problem.

Metrics include accuracy to evaluate the model's performance.

Training the Model:

The model is trained for 20 epochs using the augmented training data. Each epoch is a complete pass through the entire dataset.

Validation is performed using the test data to monitor how well the model is learning.

Evaluating the Model:

After training, the model is tested on the test data to make predictions. The accuracy and performance are measured using a classification report that includes precision, recall, and F1-score.

Saving the Model:

Once trained, the model is saved as a .h5 file, which can later be used in real-time detection.

Plotting the Training Results:

Finally, a plot is created to visualize the training and validation loss and accuracy over the epochs, helping assess the model's learning process.

Key Concepts:

Transfer Learning: Instead of training a CNN from scratch, you are using MobileNetV2, a model pre-trained on a massive image dataset. Transfer learning allows you to build a model with much less data and time by adapting a pre-trained model for your specific task.

Data Augmentation: Techniques like rotation, zoom, and shifts artificially expand the dataset and improve the model's ability to generalize to new images.

One-Hot Encoding: This is a way to represent categorical labels (like "with_mask" and "without_mask") as binary arrays (e.g., [1, 0] for "with_mask" and [0, 1] for "without_mask").

Final Output:

The trained model will take in an image and predict whether the person in the image is wearing a mask or not.

The performance of the model is visualized through accuracy and loss curves, which show how well the model learned during the training process.

This code effectively sets up a reliable mask detection system that can be deployed for real-time applications using a camera feed to check mask compliance.

The model has demonstrated impressive performance throughout the training process, with training accuracy rising from approximately 96.88% to more than 99% y the final epochs. Validation accuracy also improved significantly, starting at 96.96% and reaching 98.94%. Correspondingly, the training loss decreased from 0.1223 to as low as 0.0046, indicating effective learning. The consistent reduction in validation loss further confirms the model's ability to generalize well to unseen data, achieving a low validation loss of around 0.0349. While the results are promising, implementing early stopping could be beneficial to prevent overfitting, especially if performance metrics stabilize before all epochs are completed. Overall, the training process has been successful, and the model is well-equipped for evaluation or further fine-tuning as needed.

Mask Detection from Live Video Feed

Key Libraries and Components

1. **TensorFlow/Keras**: Utilized for loading the mask detection model, which is trained to differentiate between masked and unmasked faces.
2. **OpenCV**: A powerful library for computer vision tasks. It captures video from the webcam, processes frames, and draws bounding boxes around detected faces.
3. **imutils**: This library simplifies basic image processing tasks, such as resizing frames and displaying them.

Main Functionalities

1. **Face Detection**: The code employs a deep learning model based on the SSD (Single Shot Detector) architecture for detecting faces. The model uses a pre-trained Caffe model (deploy.prototxt and res10_300x300_ssd_iter_140000.caffemodel) to identify faces within frames captured from the webcam.
2. **Mask Detection**: The mask detection model, saved as mask_detector.h5, predicts whether the detected faces are wearing masks or not. The detect_and_predict_mask function is responsible for handling the detection and prediction process:
    - It first creates a blob from the input frame to prepare it for processing.
    - The blob is then passed through the face detector to get face detections.
    - For each detected face, it extracts the Region of Interest (ROI), preprocesses it, and appends it to a list for further prediction.
    - If faces are detected, it makes batch predictions to classify them based on the model's output.
3. **Displaying Results**: The results are visually represented on the video feed:
    - Bounding boxes are drawn around detected faces.
    - Labels indicating whether a mask is present or not are overlaid on the bounding boxes, along with the corresponding confidence percentage.
    - The code allows customization of the font, size, and thickness of the displayed labels to enhance visibility.
4. **Real-time Video Processing**: The video stream is processed in a loop, where frames are continually captured, analyzed for face detections, and updated with predictions. The window displaying the video can be resized, accommodating different screen sizes for a better user experience.

User Interaction

The application continuously runs until the user presses the 'q' key, at which point the program gracefully shuts down, closing the video window and stopping the video stream.

Understanding deploy.prototxt and res10_300x300_ssd_iter_140000.caffemodel

When working with deep learning models for object detection, particularly for tasks like face detection, the model architecture and weights need to be specified and loaded properly. In this context, deploy.prototxt and res10_300x300_ssd_iter_140000.caffemodel are essential files that play distinct roles in the functioning of the face detection system:

1. deploy.prototxt

- **File Type**: Protocol Buffers (protobuf) file.
- **Purpose**: Defines the structure and configuration of the face detection model.
- **Contents**: This file outlines the neural network architecture, specifying the layers, layer types, and connections between them. It serves as a blueprint that describes how data flows through the network.
- **Model Configuration**: The deploy.prototxt file contains information such as:
    - Input size (e.g., the image dimensions).
    - The type of layers used (e.g., convolutional layers, pooling layers).
    - Activation functions.
    - Other hyperparameters that govern the model's behavior.

2. res10_300x300_ssd_iter_140000.caffemodel

- **File Type**: Caffe model file.
- **Purpose**: Stores the trained weights of the face detection model.
- **Contents**: This file contains the learned parameters (weights and biases) of the neural network as a result of the training process. It encapsulates the model’s ability to detect faces based on the data it was trained on.
- **Training Information**: The name of the file (res10_300x300_ssd_iter_140000) indicates that it corresponds to a specific training iteration (140,000 iterations) of a Single Shot MultiBox Detector (SSD) model. The numbers (300x300) refer to the input image size the model expects.

Together in Use

When deploying a face detection system:

- The **deploy.prototxt** file is loaded to define the architecture of the SSD model, specifying how input images should be processed.
- The **res10_300x300_ssd_iter_140000.caffemodel** file is loaded to set the parameters of the model based on the training it has undergone. This allows the system to utilize the learned features effectively.

Streamlit App

The application uses a webcam to detect whether individuals are wearing face masks in real time. It leverages a pre-trained mask detection model, allowing users to monitor mask compliance effortlessly.

Key Features

1. **Webcam Integration**: Utilizes the front camera to capture live video streams.
2. **Face Mask Detection**: Processes each video frame through a pre-trained model to determine if a mask is present.
3. **User Interface**:
    - **Start/Stop Controls**: Users can easily start or stop the video stream with a button click.
    - **Real-Time Feedback**: Displays predictions (i.e., "Mask" or "No Mask") on the video feed.
4. **Frame Processing**: Captured frames are resized, normalized, and formatted to meet the model's input requirements.
5. **Dynamic Updates**: Continuously updates the display with the current video feed and detection results until the stream is stopped.