**FACE MASK DETECTION USING CNN**

**Submitted for**

**Statistical Machine Learning CSET211**

Submitted by:

**(E23CSEU0660) HARSH RATHORE**

**(E23CSEU0648) JAYESH ARORA**

**(E23CSEU0659) VINEET BHALLA**

Submitted to

**SANCHALI DAS**

**July-Dec 2024**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

A close-up of a logo

Description automatically generated

**INDEX**

|  |  |  |
| --- | --- | --- |
| Sr.No | Content | Page No |
| 1 | Abstract |  |
| 2 | Introduction |  |
| 3 | Related Work |  |
| 4 | Methodology |  |
| 5 | Hardware/Software Required |  |
| 6 | Experimental Results |  |
| 7 | Conclusions |  |
| 8 | Future Scope |  |
| 9 | GitHub Link |  |

**Abstract**

The COVID-19 pandemic underscored the importance of wearing face masks to reduce the spread of infectious diseases. However, manual monitoring of mask compliance in public areas is challenging, especially in crowded spaces. This project aims to develop an automated system for detecting whether individuals are wearing masks, leveraging the power of Convolutional Neural Networks (CNNs). CNNs are a class of deep learning models specifically designed for image processing tasks. The project uses a carefully curated dataset of images containing masked and unmasked faces. After preprocessing and augmenting the data to ensure diversity and robustness, a custom CNN model was designed, trained, and validated. The model achieved high accuracy in detecting face masks under various conditions, including different lighting and orientations. This project not only demonstrates the application of machine learning in public health but also highlights its potential for real-world deployment in surveillance and compliance monitoring systems.

**Introduction**

**Motivation**

Face mask detection has emerged as a critical requirement in enforcing public health policies during pandemics. Wearing masks has been scientifically proven to reduce viral transmission, making it an essential preventative measure. However, the enforcement of mask mandates in crowded or large-scale settings is impractical through manual means. Automated solutions are needed to efficiently monitor compliance and minimize human intervention.

**Problem Statement**

The primary problem this project addresses is the automated detection of face masks using image data. The task involves classifying individuals in images or video feeds into two categories: wearing a mask or not wearing a mask. The system must handle a variety of challenges, such as differences in face shapes, lighting conditions, and types of masks.

**Objectives**

1. To develop a machine learning model capable of detecting face masks with high accuracy.
2. To train the model using a large and diverse dataset to ensure generalization across different scenarios.
3. To deploy the system in a way that supports real-time processing, making it suitable for use in crowded public spaces.

**Scope**

The application of this project extends to various domains, including public surveillance, transportation hubs, workplaces, and healthcare facilities. Beyond pandemic scenarios, the system can also be adapted for other forms of object detection.

**Related Work**

Face mask detection has become a widely researched area in the field of computer vision. Initial approaches relied on traditional image processing techniques, where handcrafted features such as edges, textures, and colors were extracted from images. While these methods worked for simple cases, they often failed under complex conditions, such as varying lighting or occlusions.

With the advent of machine learning, researchers began using models like Support Vector Machines (SVMs) and Decision Trees for binary classification tasks. However, these models required extensive feature engineering and were limited by their inability to scale with complex datasets.

In recent years, deep learning techniques have revolutionized image-based tasks. Convolutional Neural Networks (CNNs) have become the go-to approach for object detection and classification. Models like ResNet, MobileNet, and YOLO have demonstrated exceptional performance in tasks ranging from face recognition to real-time object tracking. Notable studies include Loey et al. (2020), where YOLO was combined with ResNet for mask detection, achieving high precision and recall. Another study by Wang et al. (2021) explored MobileNet for edge-device deployment.

Our project builds upon these existing works by focusing on a custom CNN model optimized for mask detection. Unlike pre-trained models, a custom architecture allows flexibility in balancing accuracy and computational requirements, making it suitable for real-time applications.

**Methodology**

**4.1 Dataset and Preprocessing**

The dataset forms the backbone of any machine learning project. For this project, we used publicly available datasets from platforms like Kaggle, supplemented with additional data from open image repositories. The dataset consists of 10,000 images, evenly divided between masked and unmasked faces.

**Data Preprocessing Steps**:

1. **Resizing**: All images were resized to 128x128 pixels to ensure uniformity.
2. **Normalization**: Pixel values were scaled to a range of [0,1] to enhance convergence during training.
3. **Data Augmentation**: Techniques such as rotation, flipping, and brightness adjustment were applied to increase variability and prevent overfitting.

**4.2 Model Architecture**

The custom CNN model consists of the following layers:

1. **Convolutional Layers**: Three convolutional layers with 32, 64, and 128 filters, respectively, for feature extraction.
2. **Pooling Layers**: MaxPooling layers reduce spatial dimensions, making computations efficient.
3. **Dropout Layers**: Dropout was applied with a rate of 0.5 to prevent overfitting.
4. **Fully Connected Layers**: Two dense layers, including a final softmax layer for binary classification.

**4.3 Training Process**

The model was trained using the Adam optimizer with a learning rate of 0.001. Categorical crossentropy was used as the loss function to handle the two-class problem. Training was conducted over 25 epochs, with early stopping based on validation loss.

**4.4 Evaluation Metrics**

Evaluation metrics include accuracy, precision, recall, and F1-score. A confusion matrix was also used to analyze misclassifications.

**Convolutional Neural Networks (CNNs) in Machine Learning**

Convolutional Neural Networks (CNNs) are a specialized class of artificial neural networks designed primarily for processing structured data such as images, videos, and time series. Inspired by the organization of the visual cortex in the human brain, CNNs have become foundational in modern machine learning, especially in tasks requiring high performance in image and video recognition, object detection, and more.

**Architecture of CNN**

A CNN consists of three main types of layers, each playing a unique role in feature extraction and prediction: convolutional layers, pooling layers, and fully connected layers. Together, these layers form a pipeline that automatically learns features from raw data and maps them to desired outputs.

1. **Convolutional Layer**  
   The convolutional layer is the core building block of CNNs. It applies convolution operations using filters (kernels) to extract spatial features from input data, such as edges, textures, and patterns. A kernel is a small matrix that slides over the input image and performs element-wise multiplication and summation. This process results in feature maps that represent the presence and location of learned features in the data.

Key concepts:

* + **Stride** determines how much the filter shifts during convolution.
  + **Padding** ensures that the spatial dimensions of the output remain consistent with the input by adding extra layers of zeros around the borders of the input.
  + **ReLU (Rectified Linear Unit)** is often applied after convolution to introduce non-linearity, enabling the network to model complex patterns.

1. **Pooling Layer**  
   Pooling layers reduce the spatial dimensions of feature maps, decreasing computational requirements and enhancing feature robustness. Common pooling operations include max pooling (selecting the maximum value in a region) and average pooling (calculating the mean value). For example, max pooling captures the most prominent feature in a given area, making the network invariant to small shifts or distortions.
2. **Fully Connected Layer**  
   In the final stages, feature maps are flattened into a one-dimensional vector and passed through fully connected layers. These layers operate like traditional neural networks, learning high-level representations and mapping them to output classes or continuous values, depending on the task.

**How CNNs Work**

1. **Feature Extraction**  
   The initial convolutional and pooling layers extract low-level features like edges and gradients. As data progresses through deeper layers, the network learns increasingly abstract and complex features, such as shapes and object parts.
2. **Feature Aggregation and Decision Making**  
   Fully connected layers aggregate the learned features into predictions. For example, in an image classification task, the network assigns probabilities to predefined categories.
3. **Backpropagation and Optimization**  
   CNNs are trained using backpropagation, where the network adjusts its filters and weights based on errors between predictions and actual outputs. Gradient descent and its variants, like Adam, are commonly used optimization algorithms.

**Applications of CNNs**

CNNs excel in various domains due to their ability to capture spatial hierarchies in data. Some prominent applications include:

1. **Image Classification**  
   CNNs are widely used in tasks like identifying objects in images (e.g., cat vs. dog) and facial recognition.
2. **Object Detection and Localization**  
   Advanced CNN-based models like YOLO (You Only Look Once) and Faster R-CNN can detect and locate multiple objects in an image or video.
3. **Medical Imaging**  
   CNNs assist in detecting anomalies in X-rays, MRIs, and CT scans, such as tumors or fractures.
4. **Natural Language Processing (NLP)**  
   Though more commonly associated with images, CNNs can process text data (e.g., sentiment analysis) by treating sequences of words as structured input.
5. **Autonomous Vehicles**  
   CNNs enable vehicles to identify road signs, lanes, and obstacles in real-time.

**Advantages of CNNs**

1. **Parameter Sharing**  
   Filters are shared across the entire input, significantly reducing the number of parameters compared to fully connected networks.
2. **Translation Invariance**  
   Convolutional operations ensure that the network recognizes features regardless of their position in the input.
3. **Hierarchical Feature Learning**  
   CNNs automatically learn features at various levels of abstraction, eliminating the need for manual feature engineering.

**Challenges and Limitations**

1. **Computational Costs**  
   Training CNNs requires significant computational resources due to the large number of operations.
2. **Data Dependency**  
   CNNs often require large datasets to achieve good performance, which can be challenging in domains with limited labeled data.
3. **Overfitting**  
   With complex architectures, CNNs may memorize the training data rather than generalize to unseen examples. Techniques like dropout and data augmentation are used to address this.

**Conclusion**

CNNs are a cornerstone of modern machine learning, driving breakthroughs in fields as diverse as computer vision, healthcare, and robotics. Their ability to automatically extract and learn features from structured data has made them indispensable. As research continues, innovations like attention mechanisms and hybrid architectures promise to extend their versatility even further.

**Hardware/Software Required**

To implement and deploy the face mask detection system, both hardware and software resources are essential. This section outlines the specific requirements used in this project.

**Hardware Requirements**

1. **GPU-Enabled System**: A system equipped with a Graphics Processing Unit (GPU) is necessary for efficient training and testing of the CNN model. We used an NVIDIA GTX 1650, which provides sufficient computational power for small to medium-scale deep learning projects.
2. **Webcam or Camera**: Real-time detection requires a live video feed. A standard 720p webcam was used for testing real-time applications.
3. **Storage**: Approximately 20GB of storage is needed for datasets, model files, and logs.
4. **RAM**: At least 8GB of RAM is recommended to handle the dataset preprocessing and real-time detection.

**Software Requirements**

1. **Programming Language**: Python was chosen for its rich ecosystem of machine learning libraries and frameworks.
2. **Deep Learning Libraries**: TensorFlow and Keras were used for model development. These libraries simplify the process of building and training CNNs.
3. **Image Processing**: OpenCV was utilized for handling real-time video feeds and preprocessed image data.
4. **Visualization Tools**: Matplotlib and Seaborn were employed for plotting graphs and analyzing model performance.
5. **Integrated Development Environment (IDE)**: Jupyter Notebook was used for coding, testing, and visualization. Its interactive environment is ideal for machine learning projects.

**Deployment Environment**

For deployment, a lightweight Flask framework was used to create a web-based application. Flask allows easy integration of the trained model for inference while maintaining low overhead. The final system was tested on a Windows 10 machine, but it can also run on Linux or macOS with minimal changes.

By carefully selecting and optimizing the hardware and software resources, we ensured a balance between cost and performance, making the system scalable for real-world applications.

**Experimental Results**

The experimental results validate the effectiveness of the proposed face mask detection system. The model was trained on 10,000 images and evaluated on a test set containing 2,000 images. The results indicate high accuracy and robustness across various conditions.

**Quantitative Metrics**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Accuracy | 94.3% |
| Precision | 93.8% |
| Recall | 92.5% |
| F1-Score | 93.1% |

1. **Accuracy**: The model correctly classified 94.3% of the test images.
2. **Precision**: With a precision of 93.8%, the system minimizes false positives, ensuring reliable detection of masked individuals.
3. **Recall**: A recall of 92.5% indicates the model's ability to detect unmasked faces effectively.
4. **F1-Score**: The harmonic mean of precision and recall further supports the model's balance between the two metrics.

**Confusion Matrix**

The confusion matrix revealed that out of 2,000 test images, only 120 were misclassified. False negatives were slightly higher, indicating scope for further improvement in detecting unmasked faces under challenging conditions.

**Real-Time Performance**

In real-time testing, the system achieved frame rates of 20-25 frames per second (FPS) on a standard laptop. This makes it suitable for deployment in live surveillance systems.

Overall, the experimental results demonstrate the model's practical utility in public health monitoring and enforcement scenarios.

**Conclusions**

This project successfully developed a face mask detection system using Convolutional Neural Networks. The model demonstrated high accuracy and robustness, making it suitable for real-time applications. The results indicate that CNNs are well-suited for such classification tasks, leveraging their ability to extract complex features from image data.

The key achievements of the project include:

1. Development of a custom CNN model tailored for the face mask detection task.
2. High performance on a diverse dataset, ensuring generalization across real-world scenarios.
3. Deployment in a real-time system using Flask, demonstrating practical feasibility.

This project contributes to the field of statistical machine learning by providing an end-to-end solution for a socially relevant problem. The insights gained from this work can be extended to other object detection tasks, such as helmet detection or social distancing monitoring.

Despite its success, the project has limitations. The model's performance slightly decreases under extreme conditions, such as poor lighting or severe occlusions. Future work will address these challenges by exploring advanced architectures and transfer learning techniques.

In conclusion, this project showcases the power of machine learning in addressing real-world problems, particularly those related to public health and safety.

**Future Scope**

The face mask detection system has significant potential for further improvement and application.

**Technical Improvements**

1. **Model Optimization:**
   * Use lighter architectures like MobileNet or EfficientNet to enable deployment on edge devices, such as Raspberry Pi.
   * Quantization techniques can further reduce the model's size and improve inference speed.
2. **Advanced Training:**
   * Incorporate transfer learning using pre-trained models on large-scale datasets like ImageNet**.**
   * Train on adversarial samples to make the model robust against challenging inputs.

**Broader Applications**

1. **Integration with IoT:**
   * Combine the system with IoT devices for automated enforcement, such as triggering alerts or sending reports to authorities.
   * Deploy the model in drones for large-scale public surveillance.
2. **Multi-Class Detection:**
   * Extend the model to detect not just masks but also other public safety measures like gloves or face shields**.**
3. **Cross-Domain Use:**
   * Adapt the system for industrial applications, such as detecting helmets or other protective gear in construction sites.

**Societal Impact**

The system's deployment can have a lasting impact on public health monitoring, even beyond pandemic scenarios. It can act as a cornerstone for automated compliance systems, reducing the burden on human resources.

**GitHub Link**

The complete project, including source code, datasets, and a detailed README file, is hosted on GitHub. Access it here:

https://github.com/HRath0re/Face-Mask-Detection-\_-CNN