**Problem Statement**

Implement **Image Classification using Convolutional Neural Networks (CNNs)** for **binary classification** (Face Mask Detection: with\_mask vs without\_mask).

**Objective**

* To understand the architecture and working of Convolutional Neural Networks.
* To learn how to preprocess image data for training CNNs.
* To implement a CNN model using **Keras and TensorFlow** for binary classification.
* To evaluate model performance using validation data.
* To visualize training accuracy and loss over epochs.

**S/W Packages and H/W apparatus used**

* **Operating System:** Windows/Linux/MacOS
* **Kernel:** Python 3.x
* **Tools:** Jupyter Notebook, Anaconda, or Google Colab
* **Hardware:** CPU with minimum 4GB RAM; GPU optional for faster training

**Libraries and Packages:**

* TensorFlow
* Keras
* NumPy
* OpenCV
* Matplotlib
* scikit-learn

**Theory**

A **Convolutional Neural Network (CNN)** is a type of deep learning algorithm primarily designed for processing structured grid data such as images. CNNs automatically detect features and patterns in images, making them highly effective for image classification.

**Structure of CNNs:**

* **Input Layer:** Accepts image data.
* **Convolutional Layers:** Apply filters to extract low-level to high-level features (edges, textures, shapes).
* **Pooling Layers:** Downsample feature maps, reducing computational complexity while retaining important features.
* **Fully Connected Layers:** Flatten the feature maps and connect to dense layers to perform classification.
* **Output Layer:** Produces class probabilities (in this case, 2 classes: *with\_mask*, *without\_mask*).

**Activation Functions:**

* **ReLU**: introduces non-linearity.
* **Sigmoid**: used in the output layer for binary classification.

**Training:**

* CNNs are trained using **backpropagation** and **gradient descent**, updating weights to minimize the loss function.

**Methodology**

1. **Data Acquisition:**
   * Use the Kaggle dataset: [Face Mask Detection – 12K Images](https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset?utm_source=chatgpt.com).
   * The dataset contains two folders: *with\_mask* and *without\_mask*.
2. **Data Preparation:**
   * Normalize pixel values (0–255) to (0–1).
   * Use ImageDataGenerator for augmentation (rotation, zoom, shift, flip) to improve generalization.
3. **Model Architecture:**
   * Pretrained **MobileNetV2** used as the feature extractor (transfer learning).
   * Added layers:
     + Global Average Pooling
     + Dense layer (128 units, ReLU activation)
     + Dropout (to reduce overfitting)
     + Dense output layer (1 unit, sigmoid activation).
4. **Model Compilation:**
   * Optimizer: Adam
   * Loss: Binary Crossentropy
   * Metric: Accuracy
5. **Model Training:**
   * Train on augmented dataset with validation split (80/20).
   * Early stopping and model checkpoint callbacks used.
6. **Model Evaluation:**
   * Evaluate accuracy and loss on validation set.
7. **Visualization:**
   * Plot accuracy and loss vs. epochs.
8. **Inference:**
   * Predict mask/no-mask on unseen test images.
   * Optional: integrate with OpenCV to detect faces from webcam feed.

**Advantages**

* **Automatic Feature Extraction:** No manual feature engineering required.
* **Transfer Learning:** MobileNetV2 reduces training time.
* **Good Accuracy:** Works well with binary datasets.

**Limitations**

* Requires sufficient training images.
* Computationally intensive without GPU.
* Sensitive to hyperparameters.
* May overfit if augmentation is not applied.

**Applications**

* **Face Mask Detection** (COVID-19 compliance monitoring).
* **Facial Recognition** (security, attendance systems).
* **Medical Imaging** (tumor vs healthy classification).

**Working / Algorithm**

**Step 1:** Load dataset (with\_mask, without\_mask).  
**Step 2:** Preprocess images (resize, normalize, augment).  
**Step 3:** Build CNN (MobileNetV2 + custom dense layers).  
**Step 4:** Compile model (Adam + Binary Crossentropy).  
**Step 5:** Train with validation data.  
**Step 6:** Evaluate accuracy and loss.  
**Step 7:** Plot training/validation accuracy and loss.  
**Step 8:** Test model with new images / webcam input.

**Conclusion**

In this experiment, a CNN-based binary image classifier was implemented using TensorFlow and Keras. By leveraging **MobileNetV2** and transfer learning, the model achieved high accuracy in distinguishing between **with\_mask** and **without\_mask** images. The results demonstrate the effectiveness of CNNs for binary classification tasks, especially when combined with data augmentation and pre-trained networks.