ISSN 2320-9984

Volume 13 Issue 2 | June 2025 ISSN: 2320-9984 (Online)

International Journal of Modern Engineering & Management Research

Website: www.ijmemr.org

# Deep Learning-Based Face Mask Detection Using Optimized CNN

**Pragati Tiwari**
*M.Tech. Research Scholar*
*Computer Science and Engineering*
*Takshshila Institute of Engineering and Technology*
*Jabalpur (M.P.), India*
*Email: pragat2699@gmail.com*

**Swati Soni**
*Assistant Professor*
*Department of Computer Science and Engineering*
*Takshshila Institute of Engineering and Technology*
*Jabalpur (M.P.), India*
*Email: swatisoni@takshshila.org*

Abstract:— *Face mask detection has become crucial for public health and safety, requiring robust and efficient deep learning models for real-time identification. This study evaluates an existing CNN-based approach for face mask detection and introduces three newly implemented CNNs (CNN1, CNN2, CNN3) designed for binary classification (mask, no mask). These models improve upon dataset sizes, preprocessing techniques, and model configurations. The proposed models utilize Kaggle and GitHub datasets, RGB images, and structured train-validation-test splits. The dataset is downloaded, extracted, and preprocessed in Google Colab, leveraging GPU acceleration for efficient model training. Among the three models, CNN3, which incorporates three convolutional layers, three max-pooling layers, and data augmentation, achieves the best generalization performance, with a validation accuracy of 95.65%. A comparative analysis highlights CNN3's superiority in dataset scalability, accuracy, and loss minimization, making it the most optimal model for face mask detection.*

***Keywords:***— *Face Mask Detection, Convolutional Neural Network (CNN), Binary Classification, Deep Learning, Data Augmentation, Validation Accuracy*

## 1. INTRODUCTION

Face mask detection has become a critical application in public health and safety, especially during pandemics. The existing work used a small dataset of 2,229 images (grayscale, resized to 100×100 pixels) with limited preprocessing and a basic CNN architecture. In contrast, CNN1 and CNN2 utilized the Kaggle Face Mask Dataset (7,553 RGB images, resized to 128×128×3), while CNN3 employed the GitHub Build-A-Face-Mask-Detector Dataset (10,013 RGB images, resized to 70×70×3). CNN3 incorporated data augmentation to improve generalization. The architectures of the proposed CNNs were progressively enhanced, with CNN3 integrating three convolutional layers, three max-pooling layers, and a fine-tuned learning rate of 0.0001, compared to the simpler structures in the existing model. Performance evaluation demonstrated that CNN3 achieved the highest validation accuracy as 96.90%, with the lowest validation loss 0.0865, surpassing both CNN1, CNN2, and the existing model. Additionally, CNN3 exhibited minimal overfitting due to its effective use of data augmentation and a structured train-validation-test split. This work highlights the significance of dataset size, advanced preprocessing techniques, and optimized neural network architectures in achieving robust and accurate face mask detection systems.

### *A. Digital Image Processing*

- ***Binary Image*** - A binary image is one that consists of pixels that can

have one of exactly two colors, usually black and white. Binary images are also called bi-level or two -level, Pixelart made of two colours is often referred to as 1-Bit or 1bit. This means that each pixel is stored as a single bit—i.e., a 0 Black or 1 White.

- ***Gray Scale Image*** **-** A grayscale (or graylevel) image is simply one in which the only colours are shades of gray.
- ***Color image*** **-** A color image is a digital representation of a scene or object that includes color information. In digital form, color images are typically composed of pixels, with each pixel representing a tiny portion of the image and containing information about its color.

In most color image formats, each pixel is represented by three color channels: red, green, and blue (RGB). The combination of different intensities of these three primary colors creates a wide range of colors. For example, mixing full intensity of red and green creates yellow; while mixing full intensity of red and blue creates magenta, and so on.

Color images are commonly used in various applications, including photography, computer graphics, digital art, medical imaging, and scientific visualization. They allow for the accurate representation of the visual world, providing rich detail and information about the colors present in a scene.

Color images are represented using the RGB color model, where each pixel is composed of three-color channels: red, green, and blue. Each color channel typically requires 8 bits of information to represent 256 levels of intensity (0-255). Therefore, RGB color images commonly have 24 bits per pixel (8 bits per channel × 3 channels), but higher color depths such as 32 bits per pixel (where an additional alpha channel for transparency is included) are also used.

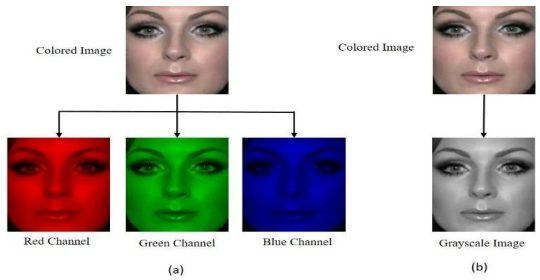


*Figure 1: RGB Image*

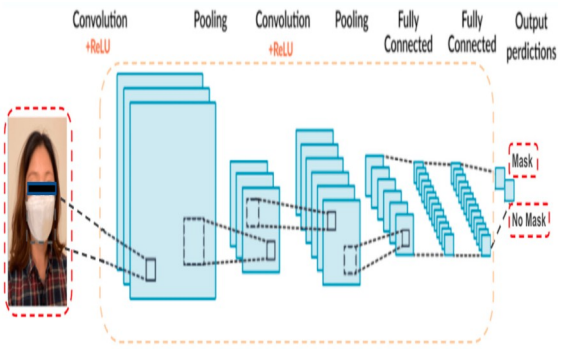


*Figure 2: Color Image Input to Convolutional Neural Network*

ANN is not appropriate for picture datasets since they require the conversion of 2 -dimensional images into 1-dimensional vectors, which makes image classification issues more challenging when using ANN. As a result, there are exponentially more trainable parameters. It requires processing and storage power to increase trainable parameters. In other words, it would be expensive and time consuming.

### *B. CNN (Convolutional Neural Networks)*

A Convolution Neural Network or CNN is a type of artificial neural network, which is widely used for image/object recognition and classification. For image processing and recognition applications, a Convolution Neural Network (CNN) is a sort of deep learning technique that works well. It is made up of multiple layers, including convolution layers, pooling layers, and fully connected layers. In order to teach CNNs to identify patterns and features linked to certain objects or classes, a sizable dataset of labelled images is used.

Sequence of CNN Layers Input => Convolution => Relu => Fully Connected => Softmax.

***Key components of a Convolutional Neural Network include:***

***Convolutional Layers:*** These layers apply convolutional operations to input images, using filters (also known as kernels) to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.

The essential part of a CNN is its convolutional layers, which are where elements like edges, textures, and forms are extracted from the input image by applying filters.

After the convolutional layers' output is processed through pooling layers, the feature maps are down-sampled to lower the spatial dimensions while keeping the most crucial data. One or more fully connected layers are then applied to the output of the pooling layers in order to classify or forecast the image.

Kernal in a CNN is a small matrix of weights that slides over the input data (such as an image), performs element-wise multiplication with the part of the input it is currently on, and then sums up all the results into a single output pixel. An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as “convolution”.

The matrix on the left contains numbers, between 0 and 255, which each correspond to the brightness of one pixel in a picture of a face.

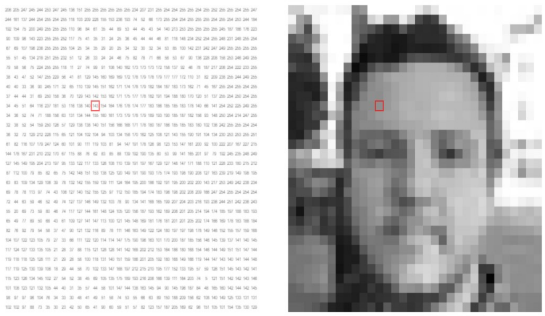

*Figure 3: Gray Scale Image Matrix*

for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right.

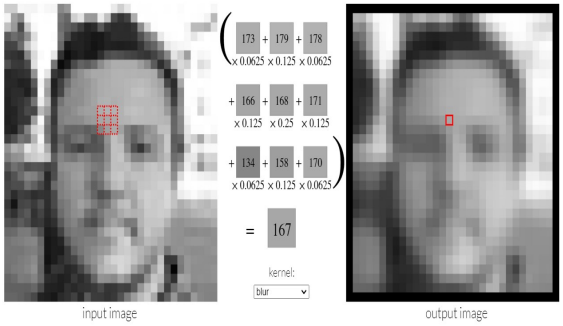


*Figure 4: Image – Kernel Product Output*

***Stride*:** It determines how many squares or pixels our filters skip when they move across the image, from left to right and from top to bottom.

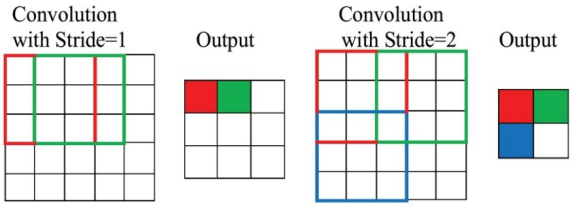


*Figure 5: Convolution with Stride*

***Padding Layer:*** In convolutional neural networks, the term “padding” refers to the number of pixels that the CNN kernel adds to an image during processing. Every additional pixel added to a CNN with padding set to 0 will have a value of none. The creation of a convolutional neural network requires padding. A small, filtered image will result from shrinking the original and using a neural network with hundreds of layers.

***Feature Map:*** is the output of a convolution layer representing specific features in the input image or feature map. During the forward pass of a CNN, the input image is convolved with one or more filters to produce multiple feature maps. Each feature map corresponds to a specific filter and represents the response of that filter to the input image.

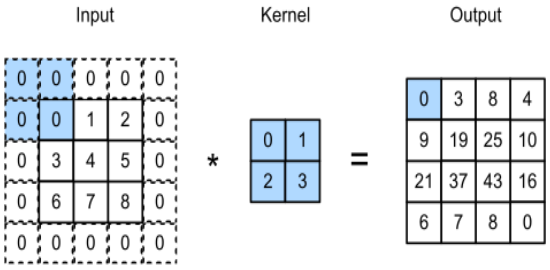


*Figure 6: Convoluted Feature Map in CNN*

***Pooling:*** By pooling layers down sampling the input's spatial dimensions, the number of network parameters and computational complexity are decreased. A typical pooling operation called "max pooling" chooses the largest value among a set of adjacent pixels. Sliding a two-dimensional filter over each feature map channel and summarising the features that fall inside the filter's coverage area constitute the pooling operation.

**Types of Pooling Layers:** Max Pooling, Min Pooling, Average Pooling

***Max Polling:*** The process of pooling that chooses the maximum element from the area of the feature map that the filter covers is called max pooling. As a result, the feature map that results from the max-pooling layer would include the most noticeable features from the prior feature map.

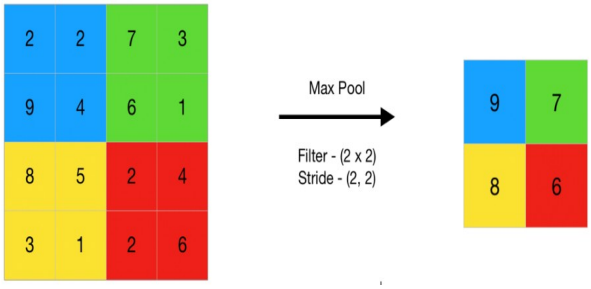


Figure 7: Max Polling

***Average Polling:*** By using average pooling, the average of the items in the feature map area that the filter covers are calculated. Therefore, average pooling provides the average of the features present in a patch, whereas max pooling provides the most noticeable feature in a specific patch of the feature map.

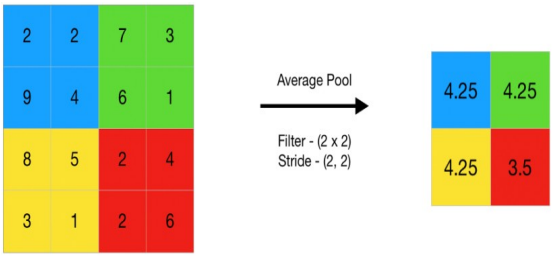


Figure 8: Average Polling

*Dimensionality reduction (Pooling Layer Advantages):* The primary benefit of pooling layers is their assistance in lowering the feature maps' spatial dimensions. Information loss (Pooling Layer Drawbacks): One of the primary drawbacks of pooling layers is their tendency to remove some information from the input feature maps, which may be crucial for the task of classification or regression at the end.

Fully Connected or Dense Layer, These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer. The dense layer is a simple Layer of neurons in which each neuron receives input from all the neurons of the previous layer, thus called as dense. The dense layer is used to classify images based on output from convolution layers.

***Activation Functions:*** Rectified Linear Unit (ReLU) and other non-linear activation functions add non-linearity to the model, enabling it to discover more intricate links in the data. A mathematical formula that is applied to each neuron's output in a neural network is called an activation function. Its goal is to provide non-linearity to the model so that the network can learn and depict more intricate correlations and patterns in the data.

A neural network would be reduced to a linear model without activation functions, unable to perform tasks like image recognition, natural language processing, or any other issue requiring the capture of complex structures in the input data.

***ReLU Activation Function*** **-** Non-linear activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model, allowing it to learn more complex relationships in the data.

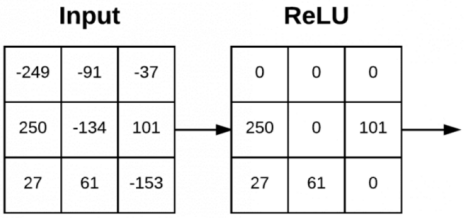


*Figure 9: ReLU Activation Function*

***Softmax Activation Function*** **-** SoftMax is an activation function which converts the inputs and output of the last layer of your neural network into a discrete probability distribution over the target classes. Commonly used in the output layer for multi-class classification problems.

Batch Normalization is a technique used in neural networks, including Convolutional Neural Networks (CNNs), to normalize the activations of a previous layer at each batch. It helps in reducing internal covariate shift, which refers to the change in the distribution of network activations due to parameter updates during training.

In batch normalization, two parameters are learned and applied to each feature map:

1. ***Scale (γ):*** This parameter scales the normalized value.
2. ***Shift (β):*** This parameter shifts the normalized value.

***Dropout:*** The dropout layer is a mask that preserves all other neurons unaltered while nullifying particular neurons' contributions to the following layer.

Dropout layers do not affect the number of parameters in a neural network. Dropout is a technique used only during the training phase to improve generalization and prevent overfitting, but it does not change the architecture of the network itself, nor does it alter the number of weights and biases.

Dropout layers in our network architecture affects how the network learns by temporarily ignoring certain neurons during training. However, dropout layers do not change the number of parameters (weights and biases) in the network. The parameters are determined by the dense (fully connected) layers and other learnable layers in the network.

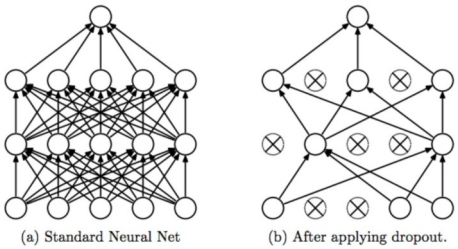


*Figure 10: Dropouts Leyer*

***Parameters:*** Trainable parameters- These are parameters that will be updated during training via backpropagation.

***Non-trainable Parameters*** **-** These parameters are not updated during training, such as the parameters in the batch normalization layers. MaxPooling2D layers do not have trainable parameters.

The Flatten layer in this model bridges the gap between the convolutional/pooling layers and the dense layers by converting the multi-dimensional tensor output of the pooling layer into a 1D vector, preparing the data for the subsequent fully connected layers.

## 2. LITERATURE REVIEW

The face mask detection approach by Das *et al.* [1] utilized two distinct datasets: Dataset 1 with 1,376 images (690 with masks, 686 without), featuring primarily front-facing, single-face images with white masks, and Dataset 2 with 853 images from Kaggle,

offering greater variability in head orientation, multiple faces, and mask colors. Preprocessing involved the use of TensorFlow and Keras for model development and image reshaping, while OpenCV handled resizing and grayscale conversion to reduce computational complexity. Images were normalized to 100×100 pixels, converted to grayscale, and labeled as 0 (with mask) or 1 (without mask). A Sequential CNN architecture was implemented, consisting of convolutional layers with ReLU activation, max-pooling layers, dropout for regularization, a dense layer, and a softmax output layer for binary classification. The model was trained using the Adam optimizer and categorical crossentropy loss, with accuracy as the evaluation metric. Data was split into 90% for training and 10% for testing, with 20% of the training data set aside for validation. Trained over 20 epochs, the model achieved an accuracy of 95.77% on Dataset 1 and 94.58% on Dataset 2, with the latter showing slightly lower performance due to increased variability in image characteristics. Training and validation loss curves were analyzed to evaluate model performance and generalization [1].

**Algorithm 1:** Face Mask Detection

**Input:** Dataset including faces with and without masks
**Output:** Categorized image depicting the presence of face mask

```
for each image in the dataset do
    Visualize the image in two categories and label them
    Convert the RGB image to Gray-scale image
    Resize the gray-scale image into 100 x 100
    Normalize the image and convert it into 4 dimensional array
end
for building the CNN model do
    Add a Convolution layer of 200 filters
    Add the second Convolution layer of 100 filters
    Insert a Flatten layer to the network classifier
    Add a Dense layer of 64 neurons
    Add the final Dense layer with 2 outputs for 2 categories
end
Split the data and train the model
```

*Figure 11: Algorithmfor face mask detection [1]*

Saini et al. (2022) proposed a real-time face mask detection system using deep learning, designed to monitor mask compliance in public spaces during the COVID-19 pandemic. The system leverages OpenCV, TensorFlow, and Keras, and is built on the lightweight MobileNetV2 architecture with Single Shot Detector (SSD) for efficient and accurate face detection. The model was trained on a dataset of 3,835 images—1,916 with masks and 1,919 without—sourced from Kaggle and RMFD. The dataset was split into 75% training and 25% testing. The network architecture includes a 128-unit fully connected layer with ReLU, 0.5 dropout, and a softmax output for binary classification. Key preprocessing included noise reduction, histogram normalization, and HAAR Cascade for face localization. Principal Component Analysis (PCA)[2] and Sobel edge detection were used for feature enhancement and dimensionality reduction. The system achieved 99% accuracy and was optimized for low-resource devices like Raspberry Pi and Google Coral, ensuring portability. A mobile application, CheckYourMask[2], was also proposed for self-monitoring. While effective, the study notes the need for larger, more diverse datasets and improved handling of partial or incorrect mask usage in future work.

Kavitha et al. (2022) proposed, a CNN method which has been used which obtains less accuracy. To improve the accuracy, the MTCNN Deep Learning algorithm is applied for this detection. It is divided into three stages: P-Net, R-Net, and O-Net. The pre-trained models InceptionV3 and VGG 16 are used in this algorithm for detection. it is observed that dataset is collected then, splited into train and test and MTCNN is fine-tuned with pre-trained models and dataset is loaded with pre trained models and finally, masks are detected. In this, 0 indicates without mask, 1 indicates with mask and 2 indicates wearing mask improperly. The available dataset contains 853 images which has been taken from Kaggle to train the deep learning architecture. Then it's divided into two parts: train and test. The dataset has been taken as multiple person in single image. 90% of the images are used for training, with the remaining 10% used for testing. The persons face has been identified using bounding box regression. Then, MTCNN algorithm has been applied for identifying with and without mask persons. Applying Several pre-trained models include Mobile Net, InceptionV3, ImageNet, VGG-16. Fine tuning the model

MTCNN -MTCNN, a prominent technique, addresses numerous problems of face identification, such as significant posture changes, severe human emotions, and varying illumination conditions. Here, MTCNN has been imported for detecting face masks. It contains three stages namely P-Net(Personal Network), R Net(Refine Network) and O-Net (Output Network).[3]

| METHODS | PRE-TRAINED MODEL | ACCURACY |
|---|---|---|
| CNN | | 98.2 |
| MTCNN | InceptionV3 | 97.58 |
| | VGG 16 | 98.52 |

*Figure 12: Performance of MTCNN [3] with VGG16 offers better accuracy*

Anirudh et al. (2022) utilized the Kaggle medical mask dataset[4], which contains 10,000 images evenly divided between "With Mask" and "Without Mask" classes, accompanied by XML annotation files. The dataset was split into 8,000 training and 2,000 testing/validation images, with input images standardized to 224×224×3 dimensions. The study compared three classification models: a Convolutional Neural Network (CNN), a Support Vector Machine (SVM), and a hybrid Principal Component Analysis with SVM (PCA+SVM).

The CNN model, featuring convolutional and pooling layers followed by dense layers and a softmax classifier, achieved the highest accuracy of 98.6%. The SVM, configured with a polynomial kernel and regularization parameter C=100C=100C=100, attained 95% accuracy, while the PCA+SVM model, which reduced feature dimensionality before classification, resulted in a lower accuracy of 90.5%—likely due to loss of information during dimensionality reduction. All models were evaluated using 5-fold cross-validation to ensure reliability. The authors concluded that CNN outperformed the other models, demonstrating superior capability in hierarchical feature learning for face mask detection.

In paper Abbas et al. (2023), face recognition (FR) using a Convolutional mixer (AFR-Conv) algorithm [5] is developed to handle face occlusion problems. A novel AFR -Conv architecture is designed by assigning priority-based weight to the different face patches along with residual connections and an AdaBoost classifier for automatically recognizing human faces. The AFR-Conv also leverages the strengths of pre-trained CNNs by extracting features using ResNet-50, Inception-v3, and DenseNet-161. The AdaBoost classifier combines these features' weighted votes to predict labels for testing images. To develop this system, we use the data augmen tation method to enhance the number of datasets using human face images. The AFR-Conv method is then used to extract robust features from images. Finally, to recognize human identity, an AdaBoost classifier is utilized. For the training and evaluation of the AFR-Conv model, a set of face images is collected from online data sources. The experimental results of the AFR-Conv approach are presented in terms of precision (PR), recall (RE), detection accuracy (DA), and F1-score metrics. Particularly, the proposed approach attains 95.5% PR, 97.6% RE, 97.5% DA, and 98.5% of F1-score on 8500 face images. The experimental results show that our proposed scheme outperforms advanced methods for face classification.[5]

Sethi et al. (2021),To contribute towards communal health, this paperaims to devise a highly accurate and real-time technique that can efficiently detect non-mask faces in public and thus, enforcing to wear mask. The proposed technique is ensemble of one-stage and two-stage detectors to achieve low inference time and high accuracy. We start with ResNet50 as a baseline and applied the concept of transfer learning to fuse high-level semantic information in multiple feature maps. In addition, we also propose a bounding box trans formation to improve localization performance during mask detection. The experiment is conducted with three popular baseline models viz. ResNet50,

AlexNet and MobileNet. We explored the possibility of these models to plug-in with the proposed model so that highly accurate results can be achieved in less inference time. It is observed that the proposed technique achieves high accuracy (98.2%) when implemented with ResNet50. Be sides, the proposed model generates 11.07% and 6.44% higher precision and recall in mask detection when compared to the recent public baseline model published as RetinaFaceMask detector. The outstanding perfor mance of the proposed model is highly suitable for video surveillance devices.[6]

### *1. Masked Face Detection Datasets*

- These datasets are used for detecting and recognizing faces, particularly under conditions where faces might be partially or fully occluded, such as by masks.
- ***FDDB (Face Detection Data Set and Benchmark):*** A dataset with face images for generic face detection but does not include masked faces.
- ***MALF (Multi-Attribute Labelled Faces):*** Includes faces with occlusions but does not explicitly focus on masked faces.
- ***CelebA (CelebFaces Attributes Dataset):*** A large dataset of celebrity faces with various attributes but without masked images.
- ***WIDERFACE:*** A dataset containing a diverse range of face images, including occluded ones.

### *2. Face Masked Datasets*

These datasets specifically contain masked face images, making them useful for tasks like face recognition with masks and masked face detection.

- ***MAFA (Masked Face Dataset):*** One of the largest datasets containing masked face images, specifically designed for masked face detection.
- ***RMFRD (Real-world Masked Face Recognition Dataset):*** Focuses on real-world masked face images, useful for face recognition tasks.
- ***SMFRD (Simulated Masked Face Recognition Dataset):*** A dataset with simulated masked faces to study the impact of face masks on recognition systems.[6]

| Type of Datasets | Dataset | Scale | #Faces | #masked face images | Occlusion |
|---|---|---|---|---|---|
| Masked face detection Datasets | FDDB [31] | 2845 | 5171 | - | - |
| | MALF [32] | 5250 | 11931 | - | ✓ |
| | celebA [33] | 200000 | 202599 | - | - |
| | WIDERFACE [34] | 32203 | 194000 | - | ✓ |
| Face masked datasets | MAFA [35] | 30811 | 37824 | 35806 | ✓ |
| | RMFRD [36] | 95000 | 9200 | 5000 | ✓ |
| | SMFRD [36] | 85000 | 5000 | 5000 | ✓ |
| | MFDD [36] | 500000 | 500000 | 24771 | ✓ |

*Figure 13: Different Categories of Datasets.*

Sheikh et al. (2023) proposed a Rapid Real-Time Face Mask Detection System (RRFMDS)[7] designed to monitor face mask compliance during the COVID-19 pandemic. The system uses a Single Shot MultiBox Detector (SSD) for face detection and a fine-tuned MobileNetV2 model for face mask classification. It is lightweight and compatible with existing CCTV infrastructure, enabling real-time detection with an average frame processing time of 0.142 seconds. The model was trained on a custom dataset of 14,535 images categorized into three classes: with mask, without mask, and incorrectly worn mask. The dataset includes various scenarios such as masks worn on the chin, covering only the mouth, and both simple and complex face masks or occluded faces. The system achieved an accuracy of 99.15% on training data and 97.81% on testing data. The dataset is publicly available on Kaggle.

| Component | Models considered |
|---|---|
| Data augmentation | ImageDataGenerator class of tensorflow |
| Face detection | SSD (single-shot multi-box detector) based on ResNet 50 |
| Face mask classifier/detection | Transfer learning of MobileNetV2 |

*Figure 14: Models used for each component*

| Performance metrics | Class | Result (1000 test samples) | Macro average | Weighted average |
|---|---|---|---|---|
| Precision | Incorrect mask | **0.98** | **0.98** | **0.98** |
| | Unmasked | **0.98** | | |
| | With mask | **0.98** | | |
| Recall | Incorrect mask | **1.00** | **0.98** | **0.98** |
| | Unmasked | **0.97** | | |
| | With mask | **0.98** | | |
| F1-score | Incorrect mask | **0.99** | **0.98** | **0.98** |
| | Unmasked | **0.97** | | |
| | With mask | **0.98** | | |
| Average accuracy (testing data) | – | **97.81%** | – | – |

*Figure 15: Performance of RRFMDS[7]*

Hussain et al. (2022) proposed an automated face mask detection system to combat the spread of COVID-19 using deep learning techniques. The study employed a Deep Convolutional Neural Network (DCNN) [8] and a MobileNetV2-based transfer learning approach for effective mask detection. The models were evaluated on two datasets: a custom dataset with 2,500 real-world images (dataset-1) and another sourced from PyImageSearch Reader Prajna Bhandary and other online sources (dataset-2). MobileNetV2 achieved 98% and 99% accuracy on dataset-1 and dataset-2, respectively, outperforming the DCNN model, which yielded 97% accuracy on both datasets. The results indicate that MobileNetV2 is a highly effective model for real-world face mask detection tasks.

| Datasets | With mask | Without mask | Total images |
|---|---|---|---|
| Dataset-1 | 1250 | 1250 | 2500 |
| Dataset-2 | 2220 | 2216 | 4436 |

*Figure 16: Dataset Used*

| Datasets | Precision | Recall | Specificity | F1-score | Accuracy | Error rate | Kappa coefficient |
|---|---|---|---|---|---|---|---|
| Dataset-1 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.014 | 0.96 |
| Dataset-2 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.017 | 0.98 |

*Figure 17: The performance of MobileNetV2 for the test datasets*

| Datasets | Precision | Recall | Specificity | F1-score | Accuracy | Error rate | Kappa coefficient |
|---|---|---|---|---|---|---|---|
| Dataset-1 | 0.96 | 0.98 | 0.95 | 0.97 | 0.97 | 0.02 | 0.94 |
| Dataset-2 | 0.96 | 0.98 | 0.95 | 0.97 | 0.97 | 0.02 | 0.94 |

*Figure 18: The performance of DCNN for the test datasets.*

Kumar Shukla et al. (2022) presented a deep learning-based face mask detection system aimed at promoting safety in public and industrial settings during the COVID-19 pandemic. Utilizing Python, OpenCV, and Keras, the system performs real-time face detection through webcam feeds. The model, based on Convolutional Neural Networks (CNN), was trained on a dataset containing 1,915 images of individuals with and without masks, using 80% for training and 20% for testing. The architecture includes multiple convolutional and dense layers to extract features and classify mask usage. Despite achieving promising results, the system faces limitations in recognizing faces obscured by hands and struggles with detection in crowded environments. Additionally, effective deployment at scale would require extensive CCTV infrastructure and human monitoring. [9]

| Classifier | Epochs | Train/test | Optimizer | Train loss | Train accuracy | Test loss | Test accuracy |
|---|---|---|---|---|---|---|---|
| Mobilenet V2 | 20 | 90/10 | ADAM | 0.0090 | 0.9981 | 0.0071 | 1.0000 |
| | | | ADAGRAD | 0.2454 | 0.9148 | 0.1811 | 0.9855 |
| | | | SGD | 0.1549 | 0.9502 | 0.0216 | 0.9855 |

*Figure 19: Model Performance*

## 3. PROBLEM STATEMENT

***A. Existing work [1] has the following limitations:***

- **Limited Dataset Size**: Uses only 1,376 + 853 images from two datasets, which may not be diverse enough.
- **Lower Image Resolution**: Uses 100×100 grayscale images, potentially missing important color and texture features.
- **Basic Preprocessing**: Only grayscale conversion, normalization, and resizing used, without augmentation techniques for better generalization.
- **Limited Network Depth**: Uses only two convolutional layers with 200 and 100 filters.
- **Overfitting Risk Not Addressed**: No explicit mention of dropout or other regularization techniques.
- **Unknown Test Performance**: No test accuracy reported, making it unclear how well the ***model*** generalizes.

***B. Existing work [2] has the following limitations:***

***1. Dataset-Related Limitations***

- ***Limited Dataset Size****:* Only 3,835 images, which may not be sufficient for training a robust deep learning model, especially for real-world variations (e.g., different face orientations, lighting conditions, and occlusions).
- ***Imbalanced Data Collection****:* The dataset is sourced from multiple sources (Kaggle and RMFD), which may introduce biases due to inconsistent image quality, variations in labeling standards, and dataset-specific features.
- ***Lack of Augmentation****:* No mention of data augmentation techniques (such as rotation, flipping, zooming, or contrast changes) to improve model generalization.

***2. Hardware & Computational Constraints***

- ***Lack of GPU/TPU Utilization****:* The work does not specify using high-performance hardware (GPU/TPU), which may limit training efficiency and real-time processing capability, especially for large-scale deployment.
- ***Memory Usage Not Addressed****:* The model reports "99% memory usage," but no mention of optimizations like model pruning, quantization, or efficient deployment strategies for low-resource environments.

***3. Preprocessing & Feature Extraction Concerns***

- ***Over-Reliance on PCA:*** PCA reduces dimensionality but may remove essential features critical for distinguishing between masked and unmasked faces. It may not be the best choice for deep learning-based feature extraction.
- ***Sobel Edge Detection in CNN****:* While edge detection can help highlight boundaries, using it directly in a CNN pipeline may not be necessary since CNNs naturally learn edge features in early convolutional layers.
- ***Face Localization Using HAAR Cascade****:* The HAAR Cascade classifier is outdated compared to modern deep-learning-based face detection techniques like MTCNN or RetinaFace, which are more robust to variations in lighting, pose, and occlusion.

***4. Methodology & Model Limitations***

- ***Overfitting Risk:***

  The training accuracy is reported as **100%**, which strongly suggests

overfitting. The validation/test accuracy is not mentioned explicitly, making it unclear how well the model generalizes.

Dropout (0.5) is applied, but additional regularization techniques like L2 weight decay or batch normalization could further help mitigate overfitting.

- ***MobileNetV2*** as Backbone:

  MobileNetV2 is efficient but may not be the best choice for fine-grained mask detection, as it is designed for lightweight applications.

  Alternative models like EfficientNet or ResNet could be explored for better feature extraction while maintaining efficiency.

- ***Limited Training Strategy****:* No details on hyperparameter tuning (e.g., learning rate schedules, optimizer choices, or batch sizes), which are crucial for improving model performance.

***5. Lack of Comprehensive Evaluation***

- ***No Mention of Loss Analysis****:* Loss values (both training and validation) are not provided, making it difficult to assess convergence and overfitting risks.
- ***No Generalization Metrics****:* The model is tested on training data but lacks evaluations on unseen test datasets or real-world scenarios. Metrics like Precision, Recall, F1-score, and AUC-ROC should be included for better performance assessment.
- ***No Comparison with Baseline Models****:* The model's performance is not compared with other architectures (e.g., ResNet, EfficientNet, or traditional ML models like SVM or Random Forest).

***6. Deployment & Real-World Challenges***

- ***No Mention of Real-Time Performance****:* While it states that the model can process video streams, no details are given on FPS (frames per second), latency, or computational efficiency in real-world deployment.
- ***Embedded System Constraints Not Addressed****:* The study mentions suitability for embedded systems but does not discuss optimizations like model quantization (e.g., TensorFlow Lite) for edge devices.
- Limited Robustness Against Variations:
- No details on handling occlusions (e.g., sunglasses, scarves).

No discussion of ethnic diversity, age range, or different mask types (e.g., cloth masks, surgical masks).

***C. Existing work [3] has the following limitations:***

- ***Dataset Size*** – The dataset contains only 853 images, which may not be sufficient for robust generalization.
- ***Multiple Faces in a Single Image*** – The dataset includes multiple persons per image, increasing the complexity of detection and classification.
- ***Fine-Tuning Challenges*** – Even though MTCNN is adjusted (fine-tuned) with pre-trained models, it still struggles with issues like poor lighting, different face angles, and facial expressions, which can reduce detection accuracy.
- ***Bounding Box Limitations*** – The method uses a box to detect faces, but sometimes it may not perfectly capture the face, which can lead to errors in identifying whether a person is wearing a mask or not.

- ***Pre-Trained Model Dependency*** – The accuracy of the model depends a lot on the pre-trained models (like InceptionV3, VGG16, etc.), and it may be worth testing other models to see if they work better for this task.

## 4. PROPOSED WORK

The proposed work is to develop, compare, and optimize different CNN architectures for Face Mask Detection while exploring various deep learning techniques. The key objectives are:

### *1. Performance Comparison of CNN Architectures*

- CNN1, CNN2, and CNN3 follow different architectures and training strategies.
- The goal is to compare their accuracy, loss, training time, and efficiency.
- Understanding how architectural changes (e.g., layer depth, dropout, optimizer settings) impact performance.

### *2. Exploring Data Augmentation & Preprocessing*

- CNN3 specifically introduces image augmentation (rotation, width shift, height shift, zoom, and flipping) to improve generalization.
- CNN1 and CNN2 rely on basic normalization and resizing.
- The objective is to assess how data augmentation affects model performance and robustness.

### *3. Hyperparameter Tuning and Optimization*

- CNN1 and CNN2 use different batch sizes, epochs, and dropout values.
- CNN3 uses a lower learning rate (0.0001) with Adam optimizer to improve stability.
- The goal is to experiment with these hyperparameters and determine the best-performing configuration.

### *4. Real-world Application & Deployment Readiness*

- The models are trained to classify people as "With Mask" or "Without Mask."
- The final trained model can be deployed in security systems (e.g., public places, workplaces) to ensure compliance with mask mandates.
- The ability to take new images as input and make real-time predictions is critical for practical applications.

### *5. Efficient Model Training Using GPU*

- The project includes using Google Colab with a T4 GPU for faster training.
- Learning how to use the Kaggle API to import large datasets efficiently.
- Understanding how dataset size and GPU acceleration impact training time.

### *6. Understanding Transferability and Scalability*

### *A. CNN1 – CNN2 DATA COLLECTION*

### *1. Face Mask Detection Using Convolutional Neural Network*

- Collecting Kaggle Dataset (Contains 2Set of Images People Wearing Mask and People Not Wearing Mask)
- Performed Binary Classification
- Image Preprocessing
- Train Test Splitting
- Feed Data to CNN
- Evaluate CNN Model
- Build a Predicting System

***2. For Deep Learning Project We Get GPU Access so that Training Happen Quickly.***

- Runtime -> Change Runtime Type -> T4 GPU. If you run your file without GPU it will run by CPU, it will take lot time.

***3. When the Size of the Image Dataset is very Large.***

- Instead of downloading the complete dataset, import the dataset from API
- You need to add API Token, it is basically json file downloaded
- (Kaggle-> Setting-> API -> Create New Token -> Download Kaggle. Json File)
- Steps to Download, Set Up, and Extract a Dataset from Kaggle in Google Colab
    - Install the Kaggle package to enable access to Kaggle datasets.
    - Upload the Kaggle API key (kaggle.json) to authenticate access to Kaggle.
    - Configure the Kaggle API by placing the API key in the correct directory and setting appropriate permissions.
    - Download the face mask dataset from Kaggle using its unique identifier.
    - Check if the dataset already exists to avoid redundant downloads unless a redownload is necessary.
    - Extract the downloaded ZIP file to make the dataset accessible.
    - Verify the extracted dataset to ensure it is ready for machine learning or deep learning tasks like image classification and face mask detection.

***B. CNN-1 ALGORITHM***

***BEGIN***

***1. Dataset Preparation:***

- Download dataset from Kaggle (containing "Mask" and "No Mask" images).
- Import necessary dependencies (TensorFlow, Keras, OpenCV, NumPy,Matplotlib, PIL, etc.).
- Extract images from dataset.
- Assign labels:
    - If "Mask", label = 1
    - If "No Mask", label = 0
- Convert images to RGB format and resize to 128x128 pixels.
- Convert images and labels to NumPy arrays.

***2. Data Preprocessing:***

- Split dataset into training (80%) and testing (20%) subsets.
- Normalize or Scale Down the pixel values by dividing each pixel by 255.

***3. Model Construction:***

- Define a Sequential CNN model.
- Add convolutional layers with ReLU activation.
- Add max pooling layers to reduce dimensions.
- Apply dropout layers to prevent overfitting.
- Flatten the output and add dense (fully connected) layer with activation='sigmoid'
- Compile the model using:
    - ***Optimizer:*** Adam
    - ***Loss Function:*** Sparse Categorical Cross entropy
    - ***Metric:*** Accuracy

***4. Model Training:***

- Train CNN model using training data.
- Set batch size = 64 and epochs = 20.
- Monitor accuracy and loss on validation data.

***5. Model Evaluation and Saving:***

- Evaluate the model performance using test data.
- Save the trained model for deployment.

***6. Model Deployment and Prediction System:***

- Load the trained model.
- Process new images:
  - Convert to RGB format and resize to 128x128 pixels.
  - Normalize pixel values.
  - Predict class using trained model.
- Display results:
  - IF predicted class = 1 THEN Output "Mask Detected"
  - ELSE Output "No Mask Detected"

END

| Layer (type) | Depth (Filters/Neurons) | Height | Width | Filter Height | Filter Width |
|---|---|---|---|---|---|
| Input Layer | 3 (RGB Channels) | 128 | 128 | N/A | N/A |
| Conv2D (32 filters, 3×3) | 32 | 126 | 126 | 3 | 3 |
| MaxPooling2D (2×2) | 32 | 63 | 63 | 2 | 2 |
| Conv2D (64 filters, 3×3) | 64 | 61 | 61 | 3 | 3 |
| MaxPooling2D (2×2) | 64 | 30 | 30 | 2 | 2 |
| Flatten | 57600 | N/A | N/A | N/A | N/A |
| Dense (128 neurons, ReLU) | 128 | N/A | N/A | N/A | N/A |
| Dropout (0.5) | 128 | N/A | N/A | N/A | N/A |
| Dense (64 neurons, ReLU) | 64 | N/A | N/A | N/A | N/A |
| Dropout (0.5) | 64 | N/A | N/A | N/A | N/A |
| Dense (2 neurons, Sigmoid) | 2 | N/A | N/A | N/A | N/A |

*Figure 20: CNN – 1 LAYERS*

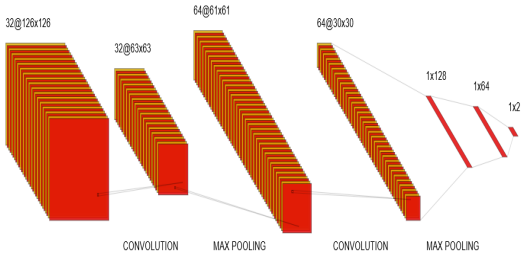


*Figure 21:CNN – 1 ARCHITECTURE*

***C. CNN-2 ALGORITHM***

***BEGIN***

***1. Import Required Libraries***

- Import tensorflow, keras, cv2, numpy, and matplotlib.pyplot.

***2. Define CNN Model***

- Create a Sequential model.
- Add Conv2D layer with 32 filters, kernel size (3×3), activation ReLU, and input shape (128,128,3).
- Add MaxPooling2D layer with pool size (2×2).
- Add another Conv2D layer with 64 filters, kernel size (3×3), activation ReLU.
- Add another MaxPooling2D layer with pool size (2×2).
- Flatten the output from convolutional layers.
- Add Dense layer with 64 neurons, activation ReLU.
- Apply Dropout (0.5) for regularization.
- Add final Dense layer with 1 neuron, activation Sigmoid (for binary classification).

***3. Compile the Model***

- Set optimizer as Adam.
- Use Binary Cross-Entropy as the loss function.
- Use Accuracy as the evaluation metric.

### *4. Train the Model*

- Fit the model using x_train_scaled and y_train.
- Set 10% validation split.
- Train for 10 epochs.
- Use batch size 64 for efficiency.
- Set verbose=1 to display training progress.

### *5. Evaluate the Model*

- Evaluate performance using x_test_scaled and y_test.
- Print Test Accuracy.

### *6. Plot Training and Validation Metrics*

- Plot Training Loss vs. Validation Loss.
- Plot Training Accuracy vs. Validation Accuracy.

### *7. Make Predictions on a New Image*

- Get image path as input from the user.
- Read the image using cv2.imread.
- Display the image using cv2_imshow.
- Resize image to (128,128,3).
- Normalize pixel values (divide by 255).
- Reshape image to (1,128,128,3) for model prediction.
- Predict class probability using modelNew.predict.
- Convert probability to class label using argmax.
- Print predicted class label.

END

| Layer Type | Depth (Filters/Neurons) | Height | Width | Filter Height | Filter Width |
|---|---|---|---|---|---|
| Conv2D | 32 | 126 | 126 | 3 | 3 |
| MaxPooling2D | 32 | 63 | 63 | 2 | 2 |
| Conv2D | 64 | 61 | 61 | 3 | 3 |
| MaxPooling2D | 64 | 30 | 30 | 2 | 2 |
| Flatten | 57600 | Not Applicable | Not Applicable | Not Applicable | Not Applicable |
| Dense | 64 | Not Applicable | Not Applicable | Not Applicable | Not Applicable |
| Dropout | 64 | Not Applicable | Not Applicable | Not Applicable | Not Applicable |
| Dense | 1 | Not Applicable | Not Applicable | Not Applicable | Not Applicable |

*Figure 22: CNN-2 LAYERS*

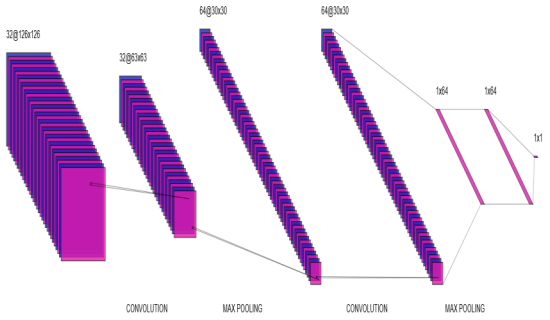


*Figure 23: CNN-2 ARCHITECTURE*

### *D. CNN-3 DATA COLLECTION*

The purpose of this task is to set up the dataset for a face mask detection project using TensorFlow and Keras in Google Colab. It involves uploading, extracting, and preparing the dataset for further processing.

### *Steps Involved:*

1. Import TensorFlow and Print Version
   - Ensures TensorFlow is installed and displays its version.
2. Install Keras
   - Installs Keras if it is not already available.
3. Upload the Dataset (data.zip) from the Local System
   - Prompts the user to select and upload the data.zip file from their device.
4. Extract (Unzip) the Uploaded Dataset
   - Unzips data.zip so that its contents are accessible for model training and analysis.

5. Delete the ZIP File After Extraction

- Removes data.zip to free up storage space, as it is no longer required.

### *C. CNN-3 Algorithm*

***BEGIN***

***1: Install Required Libraries***

- Install TensorFlow, Keras, and other necessary libraries.

***2: Data Preparation***

- Load dataset (data.zip) containing two classes:
- "With Mask" (5012 images)
- "Without Mask" (5003 images)
- Image resolution: 70x70 pixels
- Unzip dataset and organize directories for classification.

***3: Image Data Augmentation***

***Initialize `ImageDataGenerator` with:***

- Rescale pixel values between 0 and 1.
- Apply augmentation techniques:
- Rotation
- Width shift
- Height shift
- Zoom
- Horizontal flip

***3.2. Split dataset into:***

- 80% training data
- 20% validation data

***4: Model Definition***

***Create CNN model using `Sequential`:***

- ***Layer 1:*** Convolution (32 filters, 3×3 kernel, ReLU) → MaxPooling (2×2)
- ***Layer 2:*** Convolution (32 filters, 3×3 kernel, ReLU) → MaxPooling (2×2)
- ***Layer 3:*** Convolution (64 filters, 3×3 kernel, ReLU) → MaxPooling (2×2)
- Flatten the output.
- Fully connected Dense layer (64 neurons, ReLU).
- Output Dense layer (1 neuron, Sigmoid activation for binary classification).

***5: Model Compilation***

***Compile model with:***

- Optimizer: Adam (learning rate = 0.0001)
- Loss function: Binary Cross-Entropy
- Metric: Accuracy

***6: Model Training***

- Train model using `train_generator` for 30 epochs.
- Validate model using `validation_generator`.
- Monitor training and validation loss and accuracy.

***7: Plot Training and Validation Loss***

- Use `matplotlib` to plot:
  - Training loss vs. Validation loss
  - Training accuracy vs. Validation accuracy

***8: Prediction***

- Load user-specified image.
- Preprocess image (resize, rescale, convert to array).
- Predict using trained model:
  - If output > 0.5 → "Without Mask"
  - Else → "With Mask"
- Display prediction result.

END

| Layer Type | Depth (Filters/Neurons) | Height | Width | Filter Height | Filter Width |
|---|---|---|---|---|---|
| Conv2D | 32 | 70 | 70 | 3 | 3 |
| MaxPooling2D | 32 | 35 | 35 | 2 | 2 |
| Conv2D | 32 | 35 | 35 | 3 | 3 |
| MaxPooling2D | 32 | 17 | 17 | 2 | 2 |
| Conv2D | 64 | 17 | 17 | 3 | 3 |
| MaxPooling2D | 64 | 8 | 8 | 2 | 2 |
| Flatten | Reshaped to 1D vector | Not Applicable | Not Applicable | Not Applicable | Not Applicable |
| Dense | 64 Neurons | Not Applicable | Not Applicable | Not Applicable | Not Applicable |
| Dense | 1 Neuron | Not Applicable | Not Applicable | Not Applicable | Not Applicable |

*Figure 24: CNN-3 LAYERS*

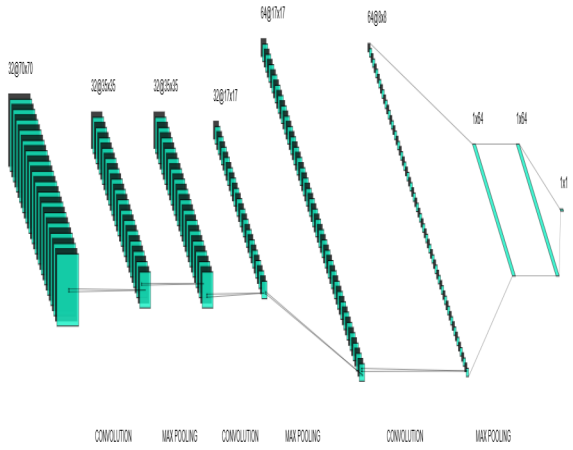


*Figure. 25: CNN-3 ARCHITECTURE*

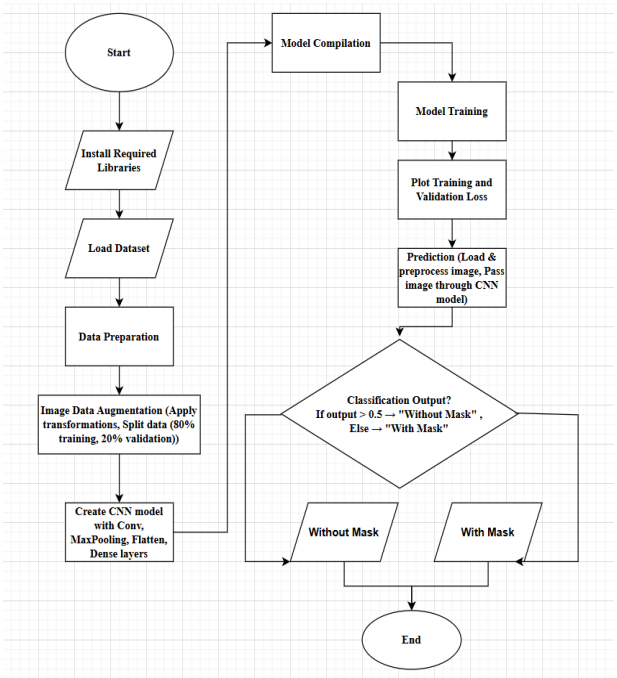


*Figure 26: Proposed Work Flow (CNN 3)*

## 5. IMPLEMENTATION

This section describes the implementation details of three different Convolutional Neural Network (CNN) models (CNN1, CNN2, and CNN3) for face mask detection using different datasets and architectures.

### *A. CNN1 Implementation*

#### *Dataset Details*

***Dataset URL:*** Kaggle Face Mask Dataset (https://www.kaggle.com/datasets/omkargurav/face-mask-dataset)


*Figure 27: Dataset 1 Images with Mask*

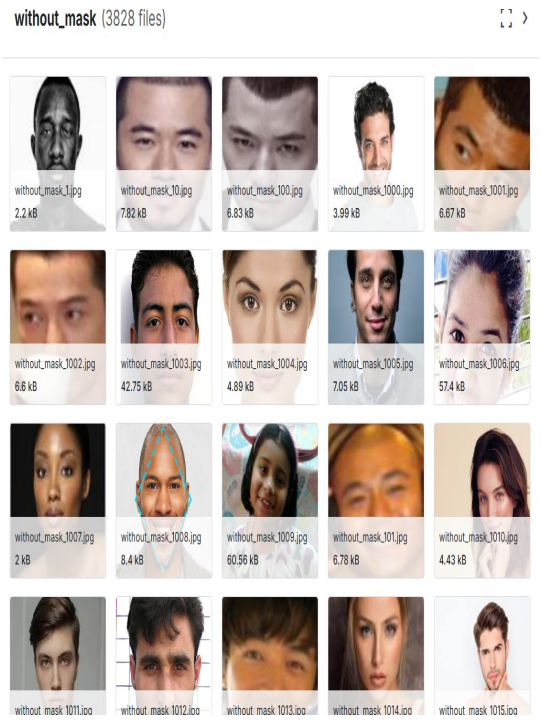


*Figure 28: Dataset 1 Images without Mask*

***Total Number of Images: 7553***

- Images with Mask: 3725 images
- Images without Mask: 3828 images

***Input Image Size***

- ***Original Image Size:*** The original images are resized to 128x128 pixels to standardize the input for the CNN model.

***After Preprocessing:***

- ***Resize:*** All images are resized to 128x128 pixels.
- ***Color Format:*** Converted to RGB color format (3 channels).
- ***Scaling:*** The pixel values are scaled from the range of 0-255 to 0-1 (i.e., divided by 255).

***Train-Test Split***

- Training Set Size: 6042 images (80% of the dataset)
- Test Set Size: 1511 images (20% of the dataset)

***CNN Model Structure***

***The CNN model has the following architecture:***

- ***Input Layer:*** Shape: (128, 128, 3) for RGB images of size 128x128
- ***Conv2D Layer 1:*** 32 filters with a kernel size of (3, 3), ReLU activation, Output shape: (126, 126, 32)
- ***MaxPooling2D Layer 1:*** Pool size: (2, 2), Output shape: (63, 63, 32)
- ***Conv2D Layer 2:*** 64 filters with a kernel size of (3, 3), ReLU activation, Output shape: (61, 61, 64)
- ***MaxPooling2D Layer 2:*** Pool size: (2, 2), Output shape: (30, 30, 64)
- ***Flatten Layer:*** Output shape: (57600)
- ***Dense Layer 1:*** 128 units with ReLU activation
- ***Dropout Layer 1:*** 50% dropout
- ***Dense Layer 2:*** 64 units with ReLU activation
- ***Dropout Layer 2:*** 50% dropout
- ***Output Layer:*** 2 units with a sigmoid activation function (for binary classification: mask vs. no mask)
- ***Model Compilation: Optimizer:*** Adam, Loss Function: Sparse Categorical Crossentropy, Metrics: Accuracy Model Training and Evaluation, Epochs: 20, Batch Size: 64, Visualization of Model Accuracy and Loss, Predictive System

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 126, 126, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 63, 63, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 61, 61, 64) | 18,496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 30, 30, 64) | 0 |
| flatten (Flatten) | (None, 57600) | 0 |
| dense (Dense) | (None, 128) | 7,372,928 |
| dropout (Dropout) | (None, 128) | 0 |
| dense_1 (Dense) | (None, 64) | 8,256 |
| dropout_1 (Dropout) | (None, 64) | 0 |
| dense_2 (Dense) | (None, 2) | 130 |

*Figure 29: CNNN-1 Model Implemented Layers*

**Total params:** 22,202,120 (84.69 MB)
**Trainable params:** 7,400,706 (28.23 MB)
**Non-trainable params:** 0 (0.00 B)
**Optimizer params:** 14,801,414 (56.46 MB)

***B. CNN2 Implementation***

***Dataset Details***

- Dataset URL: Kaggle Face Mask Dataset (https://www.kaggle.com/datasets/omkargurav/face-mask-dataset)

***Total Number of Images: 7553***

- Images with Mask: 3725 images
- Images without Mask: 3828 images

***Input Image Size***

- ***Original Image Size:*** The original images are resized to 128x128 pixels to standardize the input for the CNN model.

***After Preprocessing:***

- ***Resize:*** All images are resized to 128x128 pixels.
- ***Color Format:*** Converted to RGB color format (3 channels).
- ***Scaling:*** The pixel values are scaled from the range of 0-255 to 0-1 (i.e., divided by 255).

***Train-Test Split***

- ***Training Set Size:*** 6042 images (80% of the dataset)
- ***Test Set Size:*** 1511 images (20% of the dataset)

***Import Necessary Libraries***

- Import TensorFlow and Keras to build and train the model.

***C. CNN Model Structure***

- ***Input Shape***: (128, 128, 3) → 128x128 RGB images.
- ***First Conv Layer:*** 32 filters, (3×3) kernel, ReLU activation.
- ***First MaxPooling:*** (2×2) pool size.
- ***Second Conv Layer:*** 64 filters, (3×3) kernel, ReLU activation.
- ***Second MaxPooling:*** (2×2) pool size.
- ***Flatten Layer:*** Converts feature maps into a 1D vector.
- ***Fully Connected Layer***: 64 neurons, ReLU activation.
- ***Dropout Layer:*** 50% dropout rate to prevent overfitting.
- ***Output Layer:*** 1 neuron, sigmoid activation (for binary classification).
- ***Compile the Model:*** Optimizer: Adam, Loss Function: Binary Crossentropy (since it's a binary classification problem), Metrics: Accuracy
- ***Display Model Summary: Total Parameters:*** 3,705,921 (~14.14 MB), Trainable Parameters: 3,705,921, Non-Trainable Parameters: 0
- ***Train the Model:*** Training Data: x_train_scaled, y_train (images and binary labels)., Validation Split: 10% of training data used for validation, Epochs: 10 (can be increased for better performance), Batch Size: 64 (balanced for memory and efficiency), Verbose: 1 (to display training progress).
- ***Evaluate the Model:*** Test Data: x_test_scaled, y_test, Output: Test accuracy and loss.
- ***Make Predictions:*** New Image Input Shape: (1, 128, 128, 3), Threshold for Classification: 0.5, Predicted Class: 0 = Mask, 1 = No Mask

Model: "sequential_4"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_8 (Conv2D) | (None, 126, 126, 32) | 896 |
| max_pooling2d_8 (MaxPooling2D) | (None, 63, 63, 32) | 0 |
| conv2d_9 (Conv2D) | (None, 61, 61, 64) | 18,496 |
| max_pooling2d_9 (MaxPooling2D) | (None, 30, 30, 64) | 0 |
| flatten_4 (Flatten) | (None, 57600) | 0 |
| dense_12 (Dense) | (None, 64) | 3,686,464 |
| dropout_8 (Dropout) | (None, 64) | 0 |
| dense_13 (Dense) | (None, 1) | 65 |

*Figure 30: CNNN-2 Model Implemented Layers*

**Total params:** 3,705,921 (14.14 MB)
**Trainable params:** 3,705,921 (14.14 MB)
**Non-trainable params:** 0 (0.00 B)

***C. CNN3 Implementation***

***Dataset Overview:***

https://github.com/The-Assembly/Build-A-Face-Mask-Detector-With-TensorFlow/blob/main/FaceMask%20detection/data.zip

*Figure 31: Dataset 2 Images with Labels*

***Dataset Overview & Preprocessing***

- The dataset contains 10,015 images:
- 5,012 images of people wearing masks
- 5,003 images of people without masks
- Data is split into training (80%) and validation (20%)

***Image size: ----***

- Original: 70x70 pixels
- Rescaled to 70x70 pixels during preprocessing
- Data Augmentation & Data Splitting
- Data Augmentation is applied using ImageDataGenerator:
- Rescaling pixel values to 0-1
- Rotation, width/height shift, zooming, and horizontal flipping

***Data Loading:***

- train_generator: loads training data
- validation_generator: loads validation data

***D, CNN Model Structure***

- ***Convolutional Layers:*** 3 Conv2D layers (filters: 32, 32, 64, kernel size: (3,3), activation: ReLU), 3MaxPooling2D layers (pool size: (2,2))
- ***Flatten & Dense Layers:*** Flatten() layer, Dense layer with 64 neurons (ReLU), Output layer with 1 neuron (Sigmoid)
- ***Model Compilation & Training: Optimizer:*** Adam (learning rate = 0.0001), Loss Function: Binary Crossentropy, Metric: Accuracy, Training:, batch_size = 8, epochs = 30
- ***Model Evaluation:*** Training and validation loss/accuracy monitored after each epoch
- ***Making Predictions:*** Load and preprocess an input image, Use model.predict() to classify if the person is wearing a mask or not, Display prediction results with colored outputs (Green for masked, Red for no mask)

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 70, 70, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 35, 35, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 35, 35, 32) | 9,248 |
| max_pooling2d_1 (MaxPooling2D) | (None, 17, 17, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 17, 17, 64) | 18,496 |
| max_pooling2d_2 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| flatten (Flatten) | (None, 4096) | 0 |
| dense (Dense) | (None, 64) | 262,208 |
| dense_1 (Dense) | (None, 1) | 65 |

*Figure 32: CNNN-2 Model Implemented Layers*

**Total params:** 290,913 (1.11 MB)
**Trainable params:** 290,913 (1.11 MB)
**Non-trainable params:** 0 (0.00 B)

## 6. RESULTS

This chapter presents the performance evaluation of the three CNN models developed for the given classification task. It outlines the training, validation, and testing outcomes in terms of accuracy and loss over multiple epochs. The goal is to compare the effectiveness of each model and analyze their generalization capability based on the obtained metrics.

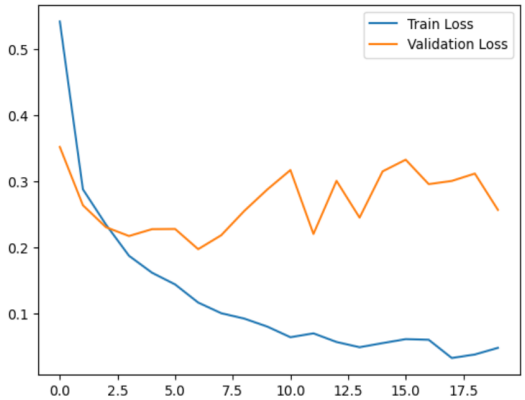


*Figure 33: CNN 1 Training & Validation Loss*

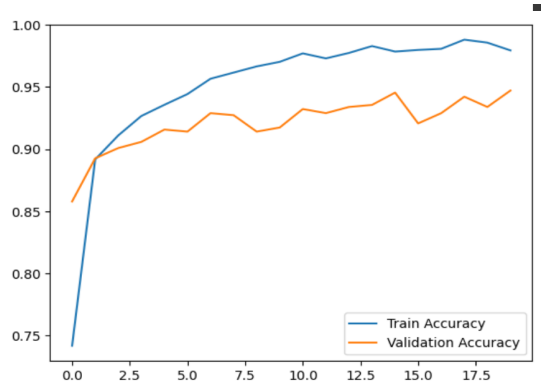


*Figure 34: CNN 1 Training & Validation Accuracy*


*Figure 35: CNN 2 Training & Validation Loss*

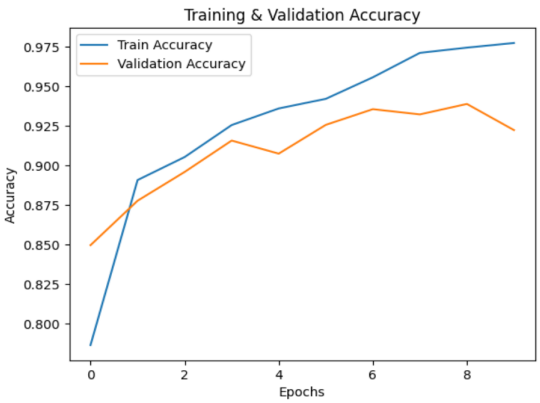


*Figure 36: CNN 2 Training & Validation Accuracy*

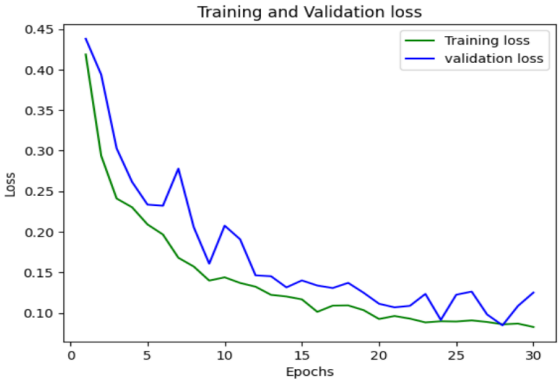


*Figure 37: CNN 3 Training & Validation Loss*

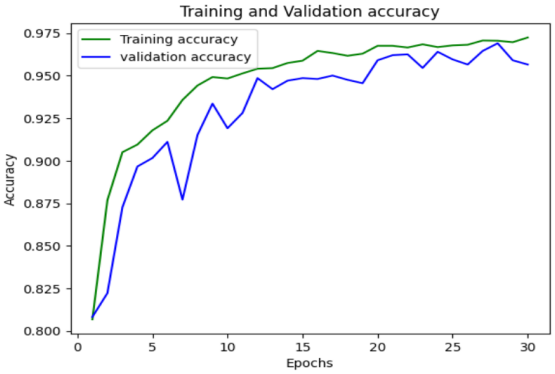


*Figure 38: CNN 3 Training & Validation Accuracy*

**Table 1: Dataset Description (CNN-wise)**

| Feature | CNN1 | CNN2 | CNN3 |
|---|---|---|---|
| Dataset Name | Kaggle Face Mask Dataset | Kaggle Face Mask Dataset | Github Build-A-Face-Mask-Detector-With-TensorFlow |
| Total Images | 7,553 (With Mask - 3,725, With out Mask 3,828) | 7,553 (With Mask - 3,725, With out Mask 3,828) | 10013 (With Mask - 5,011 With out Mask 5,002) |
| Image Resolution | 128×128×3 RGB (3 channels) | 128×128×3 RGB (3 chan-nels) | 70×70×3 RGB (3 channels) |
| Number of Classes | 2 (Binary) | 2 (Binary) | 2 (Binary) |
| Train-Validation-Test Split | Splitting: 80% Training (6042 images), 20% testing (1511 images). | Splitting: 80% Training (6042 images), 20% testing (1511 images). | Splitting: 80% Training (8011 images), 20% Validation (2002 images). |

**Table 2: Neural Network Structure**

| Feature | CNN1 | CNN2 | CNN3 |
|---|---|---|---|
| Input Shape | 128×128×3 | 128×128×3 | 70×70×3 |
| Conv Layers | 2 (32, 64 filters) | 2 (32, 64 filters) | 3 (32, 32, 64 filters) |
| Pooling Layers | 2 MaxPooling (2×2) | 2 MaxPooling (2×2) | 3 MaxPooling (2×2) |
| Fully Connected Layers | 128 → Dropout(0.5) → 64 → Dropout(0.5) | 64 → Dropout(0.5) | 64 → Output |
| Output Layer | Dense(2, Sigmoid) | Dense(1, Sigmoid) | Dense(1, Sigmoid) |
| Loss Function | Sparse Categorical Crossentropy | Binary Crossentropy | Binary Crossentropy |
| Optimizer | Adam | Adam | Adam (LR=0.0001) |
| Batch Size | 64 | 64 | 8 |
| Epochs | 20 | 10 | 30 |
| Image Generator | No | No | from tensorflow.keras.preprocessing.image import ImageDataGenerator |

**Table 3: Accuracy Comparison**

| Metric | CNN1 | CNN2 | CNN3 | Best Model |
|---|---|---|---|---|
| Initial Training Accuracy | 0.6246% | 67.37% | 0.7412% | CNN3 |
| Final Training Accuracy | 0.9853% | 97.54% | 0.9724% | CNN1 |
| Best Validation Accuracy | 0.9873% (Epoch 19) | 93.88% (Epoch 9) | 0.9690% (Epoch 28) | CNN1 |
| Final Validation Accuracy | 0.9471 | 0.9223 | 0.9565 | CNN3 |
| Test Accuracy | 0.9212% | 93.18% | Raw Prediction Probability: 0.9999 prediction on one image | CNN3 |

**Table 4: Final Performance Evaluation**

| Metric | CNN1 | CNN2 | CNN3 | Best Model |
|---|---|---|---|---|
| Initial Training Loss | 0.7696 | 0.7425 | 0.5132 | CNN3 |
| Final Training Loss | 0.0369 | 0.0560 | 0.0809 | CNN1 |
| Best Validation Loss | 0.1979 (Epoch 7) | 0.1857 (Epoch 7) | 0.0848 (Epoch 28) | CNN3 |
| Test Loss | 0.3349 | 0.2112 | - | CNN2 |
| Training Time per Epoch | ~180.5 seconds | ~157 sec | ~25 seconds | CNN3 |
| Total Training Time | ≈60.17 minutes | ~26 min | ~12.5 minutes | CNN3 |
| Overfitting Risk | Yes (after Epoch 10) | Yes (after Epoch 9) | No | CNN3 |

**Table 5: Comparison Table:**

| Feature | Existing Work | CNN1 | CNN2 | CNN3 | Better Model(s) |
|---|---|---|---|---|---|
| Dataset Size | 1,376 + 853 images (2 datasets) | 7,553 images | 7,553 images | 10,013 images | CNN3 (Larger dataset) |
| Image Resolution | 100×100 (Grayscale) | 128×128×3 (RGB) | 128×128×3 (RGB) | 70×70×3 (RGB) | CNN1, CNN2 (Higher resolution) |
| Preprocessing | Grayscale conversion, normalization, resizing | RGB, resizing, scaling | RGB, resizing, scaling | RGB, resizing, Dat Augmentation | CNN3 (More advanced preprocessing) |
| Train-Test Split | 90% train, 10% test, 20% validation from training | 80% train, 20% test | 80% train, 20% test | 80% train, 20% validation | CNN3 (More structured split) |
| Data Augmentation | Not mentioned | No | No | Yes (ImageDataGenerator) | CNN3 (Better generalization) |
| Conv Layers | 2 (200, 100 filters) | 2 (32, 64 filters) | 2 (32, 64 filters) | 3 (32, 32, 64 filters) | CNN3 (More convolutional layers) |
| Pooling Layers | 2 (3×3) | 2 (2×2) | 2 (2×2) | 3 (2×2) | CNN3 (More pooling layers) |
| Dropout | 1 (50%) | 2 (50%) | 1 (50%) | None | CNN1 |
| Fully Connected Layers | 64 neurons | 128 → 64 | 64 | 64 | CNN1 (More neurons) |
| Loss Function | Categorical Crossentropy | Sparse Categorical Crossentropy | Binary Crossentropy | Binary Crossentropy | CNN2, CNN3 (More suitable for binary classification) |
| Optimizer | Adam | Adam | Adam | Adam (LR=0.0001) | CNN3 (Fine-tuned learning rate) |
| Batch Size | Not specified | 64 | 64 | 8 | CNN3 (Better generalization) |
| Epochs | 20 | 20 | 10 | 30 | CNN3 (More training for better learning) |
| Training Accuracy | 95.77%, 94.58% | 98.53% | 97.54% | 97.23% | CNN1 (Highest accuracy) |
| Validation Accuracy | Not mentioned | 94.71% | 92.23% | 95.65% | CNN3 (Best generalization) |
| Test Accuracy | Not mentioned | 92.12% | 93.18% | Raw Prediction Probability: 0.9999 prediction on one image | CNN3 (Best generalization) |
| Loss Monitoring | Yes (Model Checkpoint) | Yes | Yes | Yes | All CNNs (Proper monitoring) |
| Overfitting Risk | Not explicitly mentioned | Yes (after Epoch 10) | Yes (after Epoch 9) | No | CNN3 (Best generalization) |

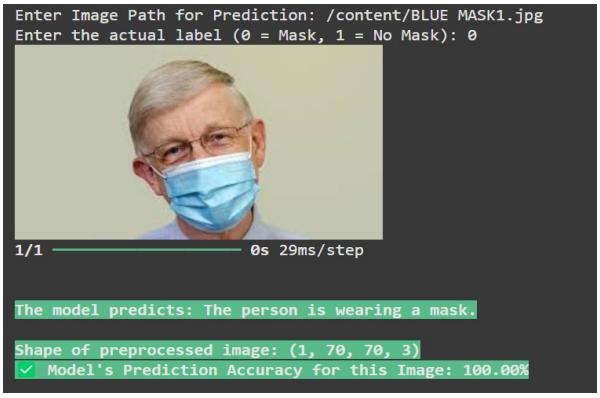


*Figure 39 : Model Correctly Identified Person Waering a Mask*

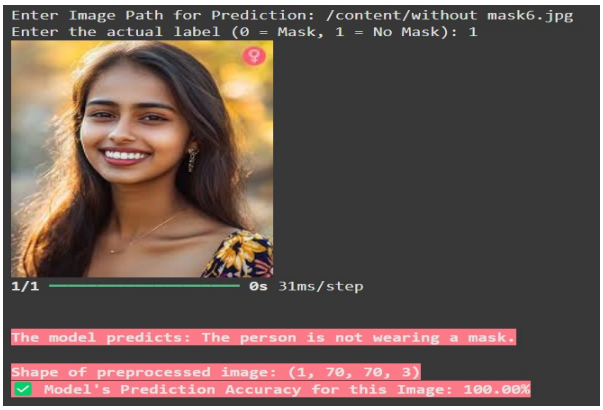


*Figure 40: Model Correctly Identified Person Not Waering a Mask*

## 7. CONCLUSION AND FUTURE WORK

The proposed work successfully developed and compared different CNN architectures for face mask detection, focusing on performance, data augmentation, and hyperparameter optimization. Among the three models (CNN1, CNN2, and CNN3), CNN3 demonstrated the best performance in terms of validation and test accuracy, with a final Validation Accuracy of 95.65%. This improvement was attributed to the use of advanced data augmentation techniques, such as rotation, zoom, and shifting, which enhanced the model's ability to generalize. Hyperparameter tuning, including a lower learning rate and optimized batch size, further contributed to CNN3's superior performance. This research also highlighted the importance of leveraging GPU acceleration for faster model training and efficient dataset handling using the Kaggle API. CNN3, with its robust design, is well-suited for real-world deployment in security systems for mask detection in public places and workplaces. Moreover, the models developed are highly transferable and can be adapted to other image classification tasks, showcasing the scalability of CNN architectures for various applications. Overall, CNN3 emerged as the optimal model, offering excellent generalization and deployment readiness.

Future advancements in face mask detection can be significantly enhanced by focusing on key areas that improve both performance and real-world applicability. Expanding datasets to include a broader range of lighting conditions, facial orientations, and various mask types will enhance model robustness and generalization. Incorporating advanced deep learning architectures like ResNet, DenseNet, or EfficientNet, along with transfer learning from large-scale datasets, can boost accuracy and training efficiency, particularly when data is limited. Real-time deployment on mobile and embedded devices through optimization techniques such as quantization and pruning will make these systems more practical and responsive. Moving beyond binary classification to include classes like "Mask Incorrectly Worn" will offer a more nuanced understanding of compliance. Additionally, the adaptability of face mask detection models to other domains such as healthcare diagnostics and security systems broadens their utility. Finally, prioritizing user privacy through privacy-preserving techniques like federated learning will be crucial in ensuring ethical deployment, particularly in surveillance scenarios. Collectively, these future directions pave the way for more accurate, efficient, and ethically responsible face mask detection systems.

## REFERENCE:

[1] Arjya Das, Mohammad Wasif Ansari, Rohini Basak, "Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV", 2020 IEEE 17th India Council International Conference (INDICON) | 978-1-7281-6916-3/20/

$31.00 ©2020 IEEE | DOI: 10.1109/ INDICON49873.2020.9342585.

[2] Kavita Saini, Shubham Bharadwaj, Vasu Gupta, "Face Mask Detection: A Deep Learning Concept", 2022 3rd International Conference on Intelligent Engineering and Management (ICIEM) | 978-1-6654-6756-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICIEM54221.2022. 9853014, 978-1-6654-6756-8/22/ $31.00 ©2022 IEEE.

[3] M.N.Kavitha, N. Kanimozhi2, S.S.Saranya3, S.Janani Sri4, V.Kalpana5, K.Jayavarthiniy6, "Face Mask Detection Using Deep Learning", 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS) | 978-1-6654-0052-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAIS53314.2022. 9742825, 978-1-6654-0052-7/22/ $31.00 ©2022 IEEE. Kavitha et al. 2022

[4] Kallakuri Anirudh, Anirudh Ravi, Vecha Sri Charan, Vijayshri Chaurasiya, "FACE MASK DETECTION USING MACHINE LEARNING", 2022 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS) | 978-1-6654-1418-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/SCEECS54111.2022. 9740913, 978-1-6654-1418-0/22/ $31.00 ©2022 IEEE. Anirudh et al. 2022

[5] Qaisar Abbas, Talal Saad Albalawi, Ganeshkumar Perumal and M. Emre Celebi, "Automatic Face Recognition System Using Deep Convolutional Mixer Architecture and AdaBoost Classifier", Abbas, Q.; Albalawi, T.S.; Perumal, G.; Celebi, M.E. Automatic Face Recognition System Using Deep Convolutional Mixer Architecture and AdaBoost Classifier. Appl. Sci. 2023, 13, 9880. https:// doi.org/ 10.3390/app13179880, Abbas et al. 2023

[6] Shilpa Sethi, Mamta Kathuria, Trilok Kaushik, "Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread", https:// doi.org/10.1016/j.jbi.2021.103848, Received 28 September 2020; Received in revised form 10 June 2021; Accepted 20 June 2021 Available online 24 June 2021. Sethi et al. 2021.

[7] Burhan ul haque Sheikh1 · Aasim Zafar1, "RRFMDS: Rapid Real-Time Face Mask Detection System for Effective COVID-19 Monitoring", Received: 8 July 2022 / Accepted: 15 February 2023 / Published online: 27 March 2023 © The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2023. Sheikh et al. 2023

[8] Dostdar Hussain, Muhammad Ismail, Israr Hussain, Roobaea Alroobaea, Saddam Hussain, and Syed Sajid Ullah, "Face Mask Detection Using Deep Convolutional Neural Network and MobileNetV2-Based Transfer Learning", Received 11 February 2022; Revised 13 March 2022; Accepted 22 March 2022; Published 4 May 2022. Hussain et al. 2022

[9] Dr. Santosh Kumar Shukla, Pradeep Singh, Diksha Haswani, Sarthak Khare, Atul Gupta, "Face Mask Detection Using Machine Learning", International Journal for Modern Trends in Science and Technology, 8 (06): 132-136, 2022 Copyright © 2022 International Journal for Modern Trends in Science and Technology ISSN: 2455-3778 online DOI: https://doi.org/10.46501/ IJMTST0806019. Kumar Shukla et

al. 2022

[10] Burhan ul Haque Sheikh1 Aasim Zafar1, "RRFMDS: Rapid Real-Time Face Mask Detection System for Effective COVID-19 Monitoring", SN Computer Science (2023) 4:288 https://doi.org/10.1007/s42979-023-01738-9.