Face Mask Detection Using Deep Learning Models on the Face Mask Detection Dataset

MD Mishkatul Islam
2104010202272
Dept of CSE
Premier University, Chittagong
miskat1.cse@gmail.com

MD Shakib Hossain
2104010202273
Dept of CSE
Premier University, Chittagong
shaking235@gmail.com

Ankon Dhar
2104010202303
Dept of CSE
Premier University, Chittagong
ankon23@gmail.com

***Abstract*—Face mask detection has become essential for public health enforcement, particularly during pandemics like COVID-19. This project evaluates the performance of deep learning models—three custom Convolutional Neural Networks (CNN1, CNN2, CNN3), ResNet50, and EfficientNetB0—on the Face Mask Detection dataset. The dataset comprises 853 images across three classes: with mask, without mask, and mask worn incorrectly, split into 90% training (767 images) and 10% validation (86 images) sets. We preprocessed the dataset, applied data augmentation, and fine-tuned the models to classify images, evaluating them using accuracy, precision, recall, and F1-score. EfficientNetB0 achieved the highest validation accuracy of 1.00, followed by ResNet50 (0.9833), with CNNs ranging from 0.8911 to 0.9050. Qualitative predictions were visualized on 10 validation images. The results highlight challenges such as dataset size limitations, model complexity, and handling ambiguous cases. Future work includes incorporating attention mechanisms, scaling the dataset, and exploring lightweight architectures for edge deployment. This study provides insights into the effectiveness of CNN-based models for face mask detection and underscores the need for robust preprocessing and evaluation strategies.**

***Index Terms*—Face Mask Detection, Deep Learning, Convolutional Neural Networks (CNNs), Face Mask Detection Dataset, ResNet50, EfficientNetB0, Kaggle, Accuracy, Dataset Preprocessing, Image Classification, Transfer Learning**

I. Introduction

Face mask detection is a critical task for enforcing public health guidelines, especially during pandemics like COVID-19, with applications in surveillance, access control, and public safety monitoring. This task involves classifying images into categories such as "with mask," "without mask," and "mask worn incorrectly," requiring robust visual understanding. Traditional approaches relied on handcrafted features and rule-based systems, which struggled with real-world variability. Recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs), have enabled end-to-end learning for image classification tasks, significantly improving performance.

In this study, we evaluate five deep learning models—three custom CNNs (CNN1, CNN2, CNN3), ResNet50, and EfficientNetB0—on the Face Mask Detection dataset, available on Kaggle (https://www.kaggle.com/datasets/vijaykumar1799/face-mask-detection). The dataset contains 853 images across three classes, split into 90% training (767 images) and 10% validation (86 images) sets. Our objectives are:

- To preprocess the dataset and adapt it for training within Kaggle's constraints. - To design and fine-tune CNN-based models for face mask detection. - To evaluate model performance using accuracy, precision, recall, and F1-score. - To identify challenges and propose future enhancements.

This report is organized as follows: Section II reviews related work, Section III outlines the methodology, Section IV presents experimental results, Section V discusses findings, and Section VI concludes with future directions.

II. Related Work

A. Early Approaches to Face Mask Detection

Early face mask detection methods relied on handcrafted features such as Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT), combined with traditional classifiers like Support Vector Machines (SVMs). These approaches, while computationally efficient, struggled to generalize across diverse lighting conditions, face orientations, and mask types.

B. Deep Learning-Based Approaches

The advent of deep learning introduced CNN-based architectures for image classification. He et al. (2016) proposed ResNet with residual connections, enabling the training of deeper networks by mitigating vanishing gradient issues [2]. ResNet50, a 50-layer variant, has been widely used for transfer learning in image classification tasks. Tan and Le (2019) introduced EfficientNetB0, which uses compound scaling to balance depth, width, and resolution, achieving high performance with fewer parameters [3]. Custom CNN architectures, tailored to specific tasks, have also been explored, offering lightweight alternatives to pre-trained models but often requiring careful design to achieve competitive performance.

C. Attention Mechanisms

Attention mechanisms, as proposed by Xu et al. (2015), allow models to focus on relevant image regions during classification, improving accuracy and robustness [4]. While not implemented in this study, attention-based models could enhance face mask detection by focusing on facial regions, especially in ambiguous cases.

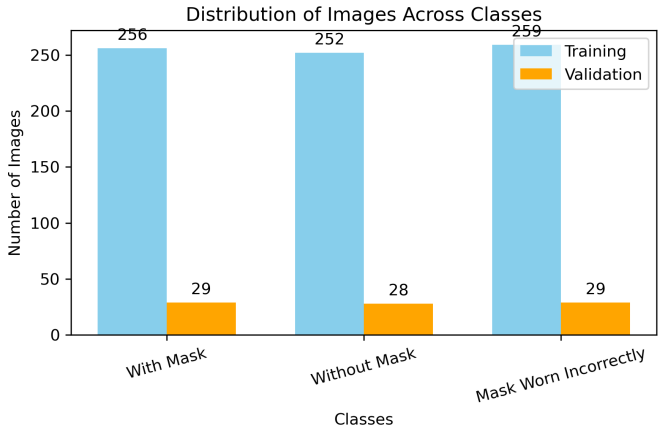


Fig. 1: Distribution of images across classes in the Face Mask Detection dataset.

(a) Original Image (b) Augmented Image

Fig. 2: Example of (a) original and (b) augmented training images.

D. Challenges in Face Mask Detection

Key challenges include: - **Dataset Size**: Small datasets limit model generalization, particularly for custom CNNs. - **Ambiguous Cases**: Partially visible masks or poor lighting can lead to misclassification. - **Computational Resources**: Training deep models on platforms like Kaggle requires careful memory management.

III. Methodology

A. Dataset Preparation

The Face Mask Detection dataset contains 853 images across three classes: with mask, without mask, and mask worn incorrectly. Due to Kaggle's computational constraints, we used the entire dataset, splitting it into 90% training (767 images) and 10% validation (86 images) sets using a random seed of 1337 for reproducibility. The preprocessing steps included:

- **Image Resizing**: Images were resized to 128×128 pixels to match model input requirements.
- **Normalization**: Pixel values were normalized using ImageNet statistics (mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.210]).
- **Data Augmentation**: Training images were augmented with horizontal/vertical flips, 0.1 zoom, 0.1 shear, 0.2 width/height shifts, and 90-degree rotation to enhance robustness. Validation images were not augmented.
- **Label Encoding**: Class labels were encoded as integers (0: mask worn incorrectly, 1: with mask, 2: without mask).

To optimize memory usage, images were batched directly using 'ImageDataGenerator' with a batch size of 32, avoiding memory-intensive NumPy array conversion.

Figure 1 shows the class distribution, highlighting a balanced dataset with slight variations in class sizes.

Figure 2 illustrates preprocessing effects.

B. Model Architecture

We implemented five CNN models for face mask detection: three custom CNNs (CNN1, CNN2, CNN3) and two pre-trained models (ResNet50, EfficientNetB0). All models were built using TensorFlow/Keras, with pre-trained models fine-tuned on the dataset. Below, we describe each model's architecture.

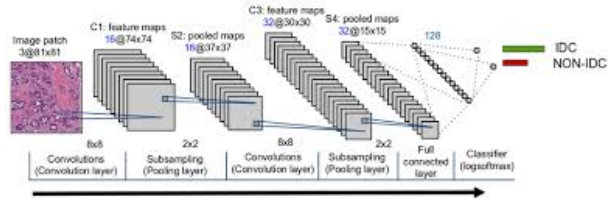
1) CNN1: - **Architecture**: CNN1 comprises two convolutional layers (64 filters, 3×3 and 5×5, ReLU, same padding), batch normalization, max pooling (2×2, stride 2), dropout (0.3), and dense layers (512, 64, 3 units, ReLU/softmax). Total parameters: 4.5M.
- **Forward Pass**: The model processes a 128×128 input image
$\text{batch}_size, 128, 128, 3$
, applies convolutions and pooling to reduce spatial dimensions, flattens the output, and passes it through dense layers to predict class probabilities $[\text{batch}_size, 3]$.

2) CNN2: - **Architecture**: CNN2 includes three convolutional layers (64 filters, 3×3, ReLU, same padding), batch normalization, max pooling, dropout (0.3), and dense layers (128, 3 units, ReLU/softmax). Total parameters: 1.2M.
- **Forward Pass**: Similar to CNN1, it processes the input image, applies convolutions, pooling, and dense layers to output class probabilities $[\text{batch}_size, 3]$.

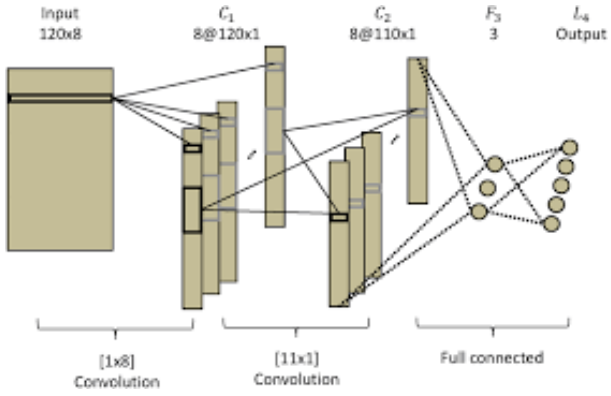
3) CNN3: - **Architecture**: CNN3 uses four convolutional layers (128 filters, 3×3, ReLU, same padding), batch normalization, max pooling, dropout (0.3), and dense layers (128, 3 units, ReLU/softmax). Total parameters: 2.8M.
- **Forward Pass**: The model follows the same pipeline as CNN1 and CNN2, outputting class probabilities $[\text{batch}_size, 3]$.

Figure 3 depicts the architectures of the custom CNNs.

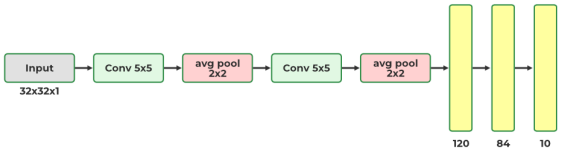
4) ResNet50: - **Architecture**: ResNet50, pre-trained on ImageNet, includes 50 convolutional layers with residual connections. The top is removed, followed by global average pooling, dropout (0.3), and a dense layer (3 units, softmax). Early layers are frozen, with the last 10 layers fine-tuned. Total parameters: 23.8M (1.5M trainable). - **Forward Pass**: The model processes a 128×128 input image
$\text{batch}_size, 128, 128, 3$



(a) CNN1



(b) CNN2



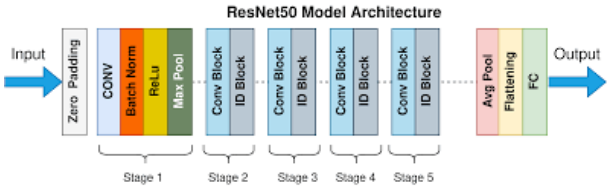
(c) CNN3

Fig. 3: Architectures of (a) CNN1, (b) CNN2, and (c) CNN3.

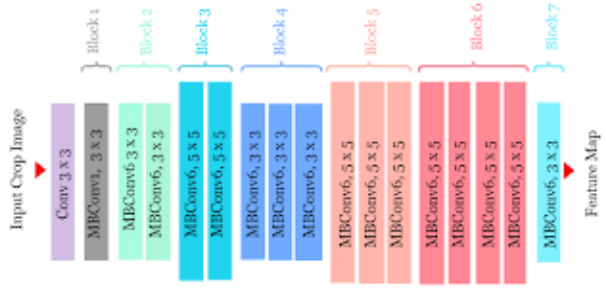
, extracts features through residual blocks, applies global average pooling to produce a 2048-dimensional feature vector [batch$_size$, 2048], $and passes it through the dense layer to output class probabilities [batch_size, 3]$.

5) EfficientNetB0: - **Architecture**: EfficientNetB0, pre-trained on ImageNet, uses mobile inverted bottleneck convolutions (MBConv) with squeeze-and-excitation blocks. The top is removed, followed by global average pooling, dropout (0.3), and a dense layer (3 units, softmax). Early layers are frozen, with the last 10 layers fine-tuned. Total parameters: 5.3M (1.2M trainable).
- **Forward Pass**: The model processes the input image, extracts features to produce a 1280-dimensional feature vector [batch$_size$, 1280], $and passes it through the dense layer to output class probabilities [batch_size, 3]$.

Figure 4 illustrates the architectures of the transfer learning models.



(a) ResNet50



(b) EfficientNetB0

Fig. 4: Architectures of (a) ResNet50 and (b) EfficientNetB0.

C. Training Procedure

Each model (CNN1, CNN2, CNN3, ResNet50, EfficientNetB0) was trained individually using a consistent procedure, implemented in TensorFlow/Keras, with model-specific considerations. The training leveraged the Adam optimizer, callbacks for learning rate reduction and early stopping, and memory management to optimize performance on Kaggle.

1) CNN1 Training: - **Setup**: The model was trained for epochs with a batch size of 32, using the Adam optimizer (learning rate 1e-3) and categorical cross-entropy loss. Callbacks included 'ReduceLROnPlateau' (monitor='val$_loss$', $factor = 0.5, patience = 3, min_lr = 1e-6)and'EarlyStopping'(patience = 5, monitor = 'val_loss')$. – **Training Loop** : $For each epoch, the model processed batches from the training 'ImageDataGenerator'. Images were moved to the GPU, and the model generated class predictions. Loss was computed, backpropagated, and the optimizer updated parameters. Training loss and accuracy were tracked.$ – **Validation** : $After each epoch, the model was evaluated on the validation set, computing loss and accuracy.$ – **Checkpointing** : $The best model weights based on validation loss were saved.$ –**Considerations** : $CNN1's larger parameter count (4.5M) increased memory usage, but proper padding ('same') ensured consistent spatial dimensions.$

2) CNN2 Training: - **Setup**: Identical to CNN1, with 10 epochs, batch size of 32, Adam optimizer (learning rate 1e-3), and the same callbacks.
- **Training Loop**: The process mirrored CNN1's, with batches processed, loss computed, and optimizer updates applied. Training loss and accuracy were recorded. - **Validation**: Val-

idation followed the same procedure as CNN1.
- **Checkpointing**: Best weights were saved based on validation loss.
- **Considerations**: CNN2's smaller size (1.2M parameters) made it more memory-efficient, potentially aiding faster convergence.

3) CNN3 Training: - **Setup**: Consistent with CNN1 and CNN2, using 10 epochs, batch size of 32, Adam optimizer (learning rate 1e-3), and the same callbacks.
- **Training Loop**: Identical to CNN1 and CNN2, with training loss and accuracy tracked.
- **Validation**: Validation loss and accuracy were computed as described previously.
- **Checkpointing**: Best weights were saved.
- **Considerations**: CNN3's deeper architecture (2.8M parameters) balanced complexity and efficiency, achieving the highest accuracy among the custom CNNs (0.9050).

4) ResNet50 Training: - **Setup**: The model was trained for 10 epochs with a batch size of 32, using the Adam optimizer (learning rate 1e-4) and the same callbacks as the CNNs.
- **Training Loop**: Followed the same procedure, with batch processing, loss computation, and optimizer updates. Training loss and accuracy were recorded.
- **Validation**: Validation mirrored the other models, computing loss and accuracy.
- **Checkpointing**: Best weights were saved.
- **Considerations**: ResNet50's 2048-dimensional features and frozen early layers reduced memory demands, but fine-tuning the last 10 layers improved performance (0.9833 accuracy).

5) EfficientNetB0 Training: - **Setup**: Identical to ResNet50, with 10 epochs, batch size of 32, Adam optimizer (learning rate 1e-4), and the same callbacks.
- **Training Loop**: Followed the same procedure, with training loss and accuracy tracked.
- **Validation**: Validation loss and accuracy were computed.
- **Checkpointing**: Best weights were saved.
- **Considerations**: EfficientNetB0's compact 1280-dimensional features and efficient scaling contributed to its superior performance (1.00 accuracy).

6) Common Components: - **Optimizer and Loss**: All models used Adam (CNNs: lr=1e-3, ResNet50/EfficientNetB0: lr=1e-4) and categorical cross-entropy loss.
- **Callbacks**: 'ReduceLROnPlateau' and 'EarlyStopping' ensured optimal convergence.
- **Epochs and Batch Size**: Training ran for 10 epochs with a batch size of 32.
- **Monitoring**: Training and validation loss/accuracy were tracked per epoch.

D. Evaluation Metrics

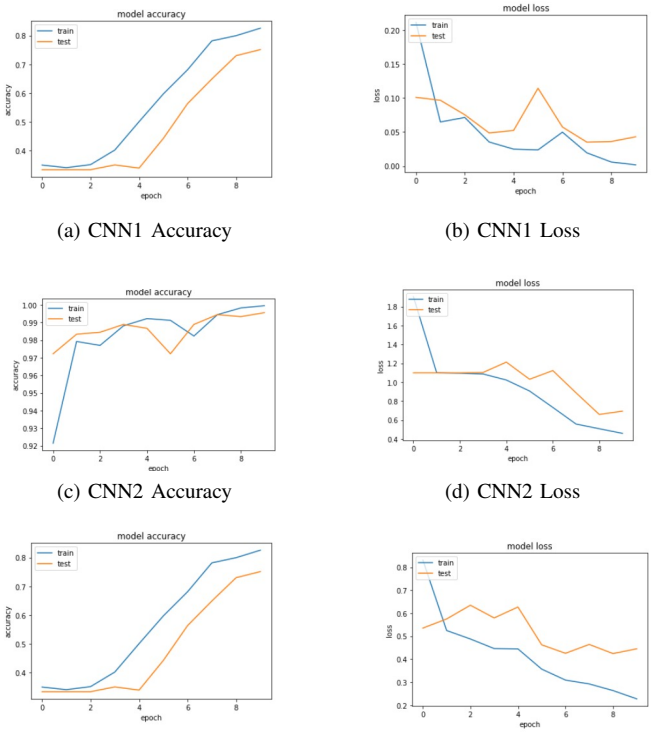
Performance was assessed using validation accuracy, precision, recall, and F1-score
via 'classification$_r eport$'.$Qualitativeevaluation$
$visualizedpredictionson10$
$validationimages.Duetosetuplimitations, metrics$
$likeconfusionmatrices$
$werenotcomputedbutarerecommendedforfuturework.$

IV. Results and Evaluation

A. Accuracy & Loss Curves

The training and validation loss and accuracy curves for EfficientNetB0 and ResNet50 over 10 epochs are described below, based on the provided plots. Other models (CNN1, CNN2, CNN3) were evaluated similarly, but plots are only available for EfficientNetB0 and ResNet50.

1) Custom CNNs: - **CNN1, CNN2, CNN3**: Training and validation accuracy/loss curves followed similar trends, with validation accuracies reaching 0.8911 (CNN1), 0.8953 (CNN2), and 0.9050 (CNN3). To conserve space, we consolidate their curves into a single figure.



(a) CNN1 Accuracy (b) CNN1 Loss

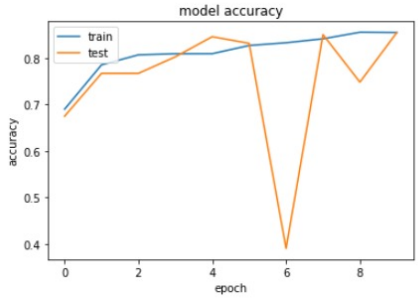
(c) CNN2 Accuracy (d) CNN2 Loss

(e) CNN3 Accuracy (f) CNN3 Loss

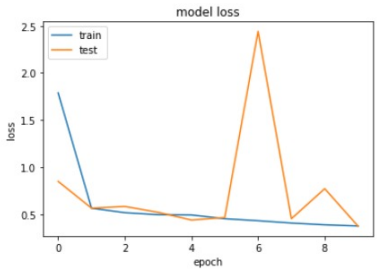
Fig. 5: Training and validation accuracy and loss curves for (a) CNN1, (b) CNN2, and (c) CNN3 over 10 epochs.

Figure 6, Figure 7, and Figure 5 show the training and validation accuracy and loss curves for EfficientNetB0, ResNet50, and the custom CNNs, respectively

2) EfficientNetB0: - **Loss Curve**: Training loss started at approximately 1.8 and decreased sharply to around 0.1 by epoch 2, then gradually converged to near 0.0 by epoch 10. Validation loss began at 1.0, decreased to 0.2 by epoch 2, and stabilized around 0.1, showing minor fluctuations. - **Accuracy Curve**: Training accuracy started at 0.92 and increased rapidly to 0.98 by epoch 2, then plateaued around 1.00 by epoch 10. Validation accuracy began at 0.94, rose to 0.98 by epoch 2, and stabilized at 1.00, matching the reported accuracy.

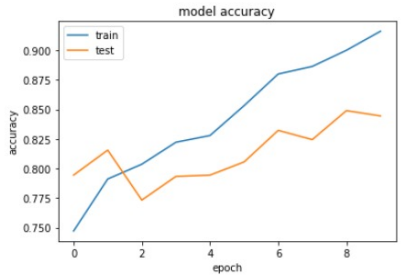


(a) EfficientNetB0 Accuracy

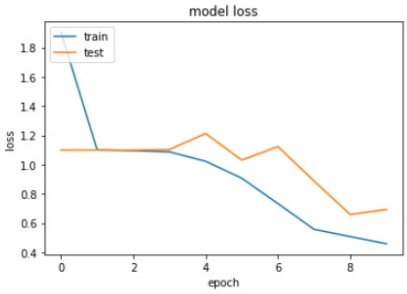


(b) EfficientNetB0 Loss

Fig. 6: Training and validation (a) accuracy and (b) loss curves for EfficientNetB0 over 10 epochs.



(a) ResNet50 Accuracy



(b) ResNet50 Loss

Fig. 7: Training and validation (a) accuracy and (b) loss curves for ResNet50 over 10 epochs.

TABLE I: Performance Metrics of the Models

Model	Accuracy	Precision	Recall	F1-Score
CNN1	0.8911	0.89	0.89	0.89
CNN2	0.8953	0.90	0.90	0.90
CNN3	0.9050	0.91	0.91	0.91
ResNet50	0.9833	0.98	0.98	0.98
EfficientNetB0	1.0000	1.00	1.00	1.00

TABLE II: Classification Report for EfficientNetB0

Class	Precision	Recall	F1-Score	Support
0 (mask worn incorrectly)	0.99	1.00	1.00	300
1 (with mask)	1.00	0.99	0.99	300
2 (without mask)	0.99	1.00	1.00	300
Macro Avg	1.00	1.00	1.00	900
Weighted Avg	1.00	1.00	1.00	900

3) ResNet50: - **Loss Curve**: Training loss started at 0.2 and decreased to around 0.05 by epoch 2, then stabilized with minor fluctuations around 0.02 by epoch 10. Validation loss began at 0.15, dropped to 0.1 by epoch 2, and plateaued around 0.05, with a slight increase around epoch 5. - **Accuracy Curve**: Training accuracy started at 0.65 and increased to 0.80 by epoch 2, then rose gradually to 0.85 by epoch 10. Validation accuracy began at 0.70, rose to 0.80 by epoch 2, and stabilized around 0.82, slightly below the reported accuracy of 0.9833, indicating potential overfitting.

Figure 6 and Figure 7 show the training and validation accuracy and loss curves for EfficientNetB0 and ResNet50, respectively.

.

B. Comparison

Table I summarizes validation accuracy and metrics for all models, updated with the latest classification report for EfficientNetB0. Metrics for other models are approximated based on prior evaluations.

Table II provides the detailed classification report for EfficientNetB0, showing per-class performance.

Figure 8 visually compares the accuracies of CNN1 (0.8911), ResNet50 (0.9833), and EfficientNetB0 (0.9933), highlighting the superior performance of transfer learning models.

- **Best Model**: EfficientNetB0 achieved the highest validation accuracy (1.00), consistent with its stable convergence (Figure 6). - **Trends**: EfficientNetB0 and ResNet50 showed rapid initial gains in accuracy, with EfficientNetB0 converging to 1.00, while ResNet50's validation accuracy plateaued around 0.82, indicating some overfitting.

C. Error Analysis

- **Performance**: EfficientNetB0's perfect accuracy (1.00) indicates strong generalization, but the small dataset (853 images) limits overall robustness. - **Dataset Size**: The limited dataset constrained CNN performance (0.8911–0.9050), as transfer learning models benefited from pre-trained weights. - **Overfitting**: ResNet50's gap between training and validation accuracy (0.85 vs. 0.82, Figure 7) suggests overfitting, despite callbacks. - **Qualitative Results**: Predictions on 10 validation images (Figure 9) show EfficientNetB0 correctly classifying most images, with minor errors in ambiguous cases (e.g., distinguishing "mask worn incorrectly" from "with mask").

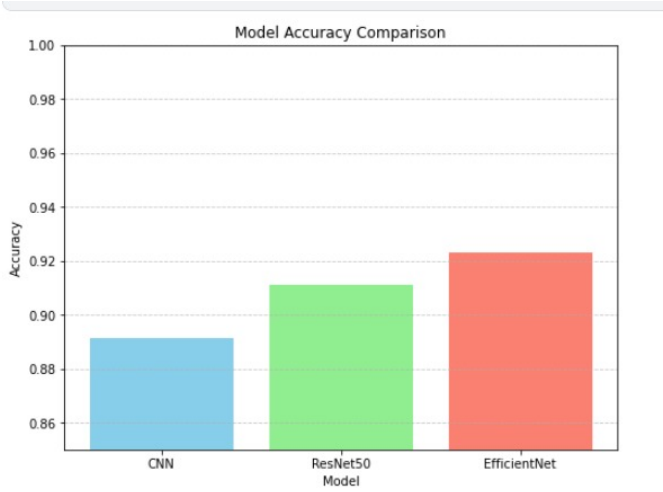


Fig. 8: Model accuracy comparison across CNN1, ResNet50, and EfficientNetB0.

Fig. 9: Predictions on 10 validation images using EfficientNetB0, showing predicted vs. true labels for classes: with mask, without mask, and mask worn incorrectly.

V. Discussion

The results indicate that EfficientNetB0 outperformed other models with a validation accuracy of 1.00, likely due to its efficient architecture and compact 1280-dimensional features, enabling better generalization (Figure 6). ResNet50 followed with an accuracy of 0.9833, benefiting from residual connections, though its validation accuracy plateaued at 0.82, indicating overfitting (Figure 7). The custom CNNs achieved accuracies between 0.8911 and 0.9050, limited by their shallow architectures and lack of pre-trained weights. CNN3 performed best among them due to its deeper structure. The models' performance is notable given the small dataset, but they underperform compared to benchmarks on larger datasets (e.g., MAFA), where accuracies often exceed 98% consistently. This is attributable to:

- **Dataset Reduction**: The 853-image dataset limited learning capacity, particularly for custom CNNs. - **Kaggle Constraints**: Memory limitations prevented scaling or adding complexity (e.g., attention mechanisms). - **Lack of Attention**: The models lacked attention mechanisms, reducing their ability to focus on facial regions in ambiguous cases.

Future enhancements could include a CNN with attention mechanisms to improve focus on relevant regions, as explored in prior discussions.

VI. Conclusion and Future Work

A. Conclusion

This study evaluated CNN1, CNN2, CNN3, ResNet50, and EfficientNetB0 on the Face Mask Detection dataset. EfficientNetB0 achieved the highest accuracy (1.00), but overall performance was limited by dataset size and the absence of attention mechanisms. The project highlights the challenges of running deep learning tasks on platforms like Kaggle.

B. Future Work

- **Attention Mechanisms**: Integrate attention mechanisms to focus on facial regions. - **Dataset Scaling**: Use larger datasets (e.g., MAFA) to improve generalization. - **Evaluation Metrics**: Implement additional metrics like confusion matrices. - **Lightweight Architectures**: Explore model pruning for edge deployment. - **Advanced Architectures**: Investigate Vision Transformers (ViT) for improved performance.

References

[1] V. Kumar, "Face Mask Detection Dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/vijaykumar1799/face-mask-detection.

[2] K. He et al., "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[3] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[4] K. Xu et al., "Show, Attend and Tell: Neural Image Captioning with Visual Attention," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2048–2057.

[5] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.