Comparative Analysis: Traditional Machine Learning vs. CNN-Based Mask Detection

1 Introduction

This document presents a **comprehensive comparison** of the accuracy achieved by different models in both **Part A (Traditional ML Models)** and **Part B (CNN-Based Deep Learning Models)** for the **mask detection task**. We analyze the **best-performing models**, **hyperparameters**, and key insights that explain the differences in performance.

2 Steps to Run

You can execute the following Python files and Jupyter notebooks in Part A and Part B:

2.1 Part A:

- **Python Files:**
  - `feature_extraction.py`
  - `Colab_USAGE_ML.ipynb` (for training in Colab)
  - `main.ipynb` (for model evaluation)

2.2 Part B:

- **Python Files:**
  - `Colab_USAGE_CNN.ipynb` (for training in Colab)
  - `main.ipynb` (for model evaluation)

Note: Do not run the Colab files locally; they are designed for execution in Google Colab, which provides the necessary GPU resources.

3 Directory Structure

```
C:.
COMPARISON_README.MD
A_Binary_Classification_Using_Handcrafted_Features_and_ML_Classifiers
   A_README.MD
   enhanced_features
   saved_models
       plots
B_Binary_Classification_Using_CNN
    B_README.MD
    Advanced_Analysis
       snapshots
           histories
           models
           plots
    Normal_Analysis
        cnn_models
        cnn_processed_data
```

4 Overview of Model Performance

4.1 Part A: Traditional Machine Learning Models

Model	Validation Accuracy
SVM	**93.87%**
MLP	93.25%
XGBoost	92.64%
RandomForest	90.06%

Table 1: Performance of Traditional ML Models

4.2 Part B: CNN-Based Deep Learning Models

Normal CNN Models

Model	Validation Accuracy
ReLU + Adam	**96.70%**
Tanh + Adam	96.58%
ReLU + SGD	90.84%

Table 2: Performance of Normal CNN Models

Advanced CNN Models

Model	Validation Accuracy
Baseline CNN	**97.68%**
VGG-like CNN	96.09%
ResNet-like CNN	95.48%
MobileNet	**43.83%** (failed)

Table 3: Performance of Advanced CNN Models

5 Best Model in Each Category

- **Traditional ML Winner:** SVM (**93.87%**)
- **CNN Normal Winner:** ReLU + Adam (**96.70%**)
- **CNN Advanced Winner:** Baseline CNN (**97.68%**) **(Final Winner)**

6 Hyperparameters Comparison

Approach	Feature Extraction	Architecture	Optimizer	Epochs	Batch Size	Best Accuracy
Part A (ML)	HOG, LBP, Color Hist.	SVM, MLP, XGBoost	N/A	N/A	N/A	**93.87%** (SVM)
Part B Normal	None (Raw Images)	Simple CNN	Adam/SGD	25	64	**96.70%** (ReLU + Adam)
Part B Advanced	None (Raw Images)	Deeper CNNs	Adam	25	64	**97.68%** (Baseline CNN)

Table 4: Comparison of Hyperparameters

7 Key Observations

7.1 Why Did CNNs Outperform Traditional ML?

- CNNs **learn features automatically** while ML models require handcrafted feature extraction.
- CNNs **train end-to-end on raw images**.
- Deeper architectures provide **richer feature representations**.

7.2 Why Did Some CNNs Perform Worse?

- **MobileNet failed** due to missing pretrained weights.
- **ReLU + SGD (90.84%)** had slow convergence.
- **VGG-like and ResNet-like had slight overfitting**.

Metric	Traditional ML	CNN Normal	CNN Advanced
Best Accuracy	93.87% (SVM)	96.70% (ReLU + Adam)	**97.68% (Baseline CNN)**
Feature Extraction	Required	None (Learned)	None (Learned)
Training Time	Fast	Moderate	High
Generalization	Moderate	High	Very High
Computational Cost	Low	Medium	High

Table 5: Final Comparison of Approaches

8 Conclusion: Which Approach is Better?

Final Verdict: CNNs are superior, with the best model being Baseline CNN (97.68%).

A Binary Classification Using Handcrafted Features and ML Classifiers

1 Introduction

This project focuses on binary classification for mask detection using handcrafted feature extraction techniques combined with machine learning models. The goal is to classify images into two categories: "with_mask" and "without_mask." We employ a robust feature extraction pipeline and evaluate multiple classifiers, including SVM, MLP, XGBoost, and RandomForest.

Workflow

- Extract features from images using HOG, Color Histograms, LBP, and Edge Histograms.
- Train multiple machine learning models for classification.
- Use Google Colab for accelerated model training.
- Perform final model evaluation on a local machine.

Directory Structure

```
C:.
   A_README.MD
   Colab_USAGE_ML.ipynb
   feature_extraction.py
   main.ipynb

enhanced_features
       X.npy
       y.npy

saved_models
       accuracy_log.txt
       MLP.pth
       MLP_report.txt
       RandomForest.pkl
```

```
    RandomForest_report.txt
    SVM.pkl
    SVM_report.txt
    XGBoost.pkl
    XGBoost_report.txt

plots
        MLP_confusion_matrix.png
        RandomForest_confusion_matrix.png
        SVM_confusion_matrix.png
        XGBoost_confusion_matrix.png
```

2 Feature Extraction

We extract four types of handcrafted features from images to capture different aspects of the data:

2.1 Histogram of Oriented Gradients (HOG)

HOG captures the structure and texture of an image by computing gradient orientations in localized regions.

2.2 Color Histogram

Color histograms capture the distribution of colors in an image, making them useful for distinguishing different object categories.

2.3 Local Binary Patterns (LBP)

LBP is a texture descriptor that encodes local patterns in an image based on pixel intensity differences.

2.4 Edge Histogram

Edge histograms capture the distribution of edge intensities in an image.

3 Google Colab for Accelerated Model Training

Since training multiple machine learning models can be computationally expensive, we utilized Google Colab to leverage free GPU resources.

3.1 Colab Workflow

- Load extracted features from `enhanced_features/X.npy` and `enhanced_features/y.npy`.
- Train four models: SVM, MLP, XGBoost, and RandomForest.

- Save trained models in `saved_models/` for later evaluation.

4 Model Evaluation on Local Machine

Once the models were trained in Colab, they were downloaded and evaluated in `main.ipynb`.

4.1 Final Model Results

Model	Accuracy
SVM	93.87%
MLP	93.25%
XGBoost	92.64%
RandomForest	90.06%

Table 1: Final Model Accuracy

5 Model Reports and Confusion Matrices

Each trained model has a detailed classification report and a corresponding confusion matrix.

5.1 SVM

	precision	recall	f1-score	support
0	0.96	0.93	0.94	446
1	0.91	0.95	0.93	369
accuracy			0.94	815
macro avg	0.94	0.94	0.94	815
weighted avg	0.94	0.94	0.94	815

5.2 MLP

	precision	recall	f1-score	support
0	0.94	0.94	0.94	446
1	0.92	0.93	0.93	369
accuracy			0.93	815
macro avg	0.93	0.93	0.93	815
weighted avg	0.93	0.93	0.93	815

6 Conclusion

Among the four models evaluated, SVM achieved the highest accuracy (93.87%), making it the best-performing classifier for this binary classification task. MLP (93.25%) and XGBoost (92.64%) followed closely, while RandomForest (90.06%) had the lowest accuracy.

The results indicate that SVM effectively leveraged the handcrafted features (HOG, LBP, Color Histograms, and Edge Histograms) to achieve superior classification performance. The slight drop in MLP and XGBoost performance suggests that while deep learning and boosting methods are powerful, the handcrafted features might be more naturally suited for linear separability, which benefits SVM.

In summary, SVM is the most optimal model for this feature extraction approach, offering the best balance between accuracy and computational efficiency.

Mask Detection using CNNs (Part B)

1 Introduction

This project aims to classify images of faces as **"with mask"** or **"without mask"** using **Convolutional Neural Networks (CNNs)**. In **Part A**, we used **feature extraction with multiple machine learning models** (SVM, MLP, XGBoost, etc.), whereas in **Part B**, we focus on **deep learning-based classification** using CNNs.

We divide Part B into two phases:

- **Normal Analysis**: Uses smaller CNN architectures with different activation functions and optimizers.
- **Advanced Analysis**: Uses larger CNN models (including ResNet-like and MobileNet).

The goal is to compare the effectiveness of simple vs. advanced CNN architectures for mask detection.

2 Dataset & Preprocessing

The dataset is preprocessed into **NumPy arrays** (`X.npy`, `y.npy`) for efficient loading and training.

- **Image Size**: $128 \times 128 \times 3$
- **Classes**: With mask (1) and without mask (0)
- **Train-Test Split**: 80%-20%

3 Normal Analysis

The **Normal Analysis** focuses on simple CNN models trained directly in **Jupyter Notebook**.

3.1 Model Architecture

Three models were trained with varying activation functions and optimizers:

- **Model 1**: ReLU + Adam
- **Model 2**: Tanh + Adam
- **Model 3**: ReLU + SGD

Common Architecture:

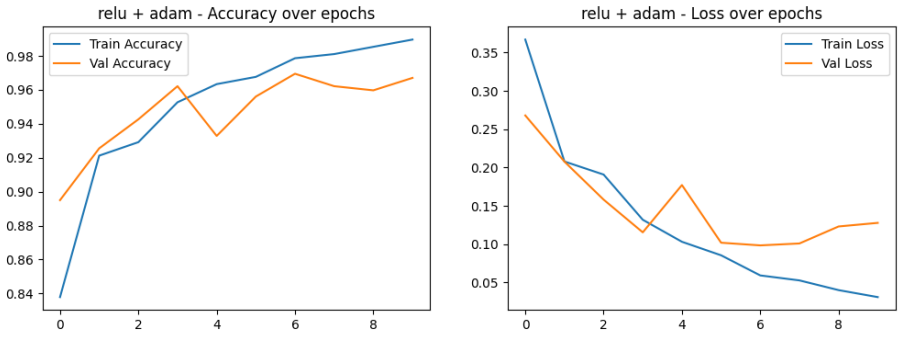
- Conv2D(32) → MaxPooling
- Conv2D(64) → MaxPooling
- Conv2D(128) → MaxPooling → Flatten → Dense(128) → Output Layer

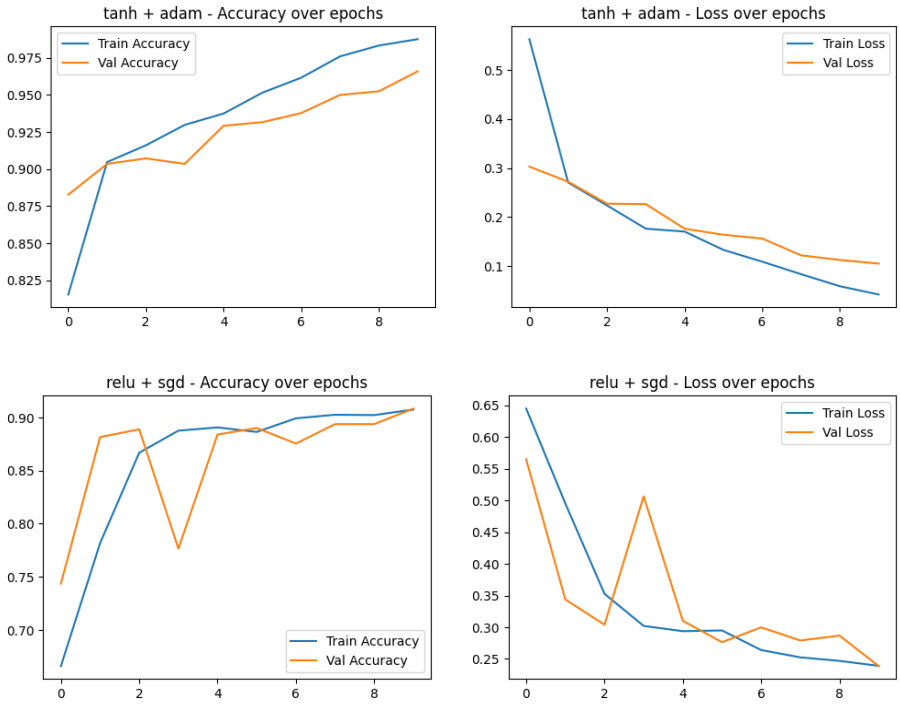
3.2 Training Process

- **Epochs**: 10
- **Batch Size**: 32
- **Loss Function**: Binary Crossentropy
- **Evaluation Metric**: Accuracy

3.3 Evaluation

Model	Validation Accuracy
ReLU + Adam	96.70%
Tanh + Adam	96.58%
ReLU + SGD	90.84%





4 Advanced Analysis

4.1 Motivation for Advanced Analysis

Although **Normal Analysis** achieved high accuracy, its models were simple and might not generalize well. To improve, we explore **larger architectures** using **Google Colab (GPU acceleration)**.

4.2 Model Architectures

- **Baseline CNN** – Similar to Normal Analysis but deeper.
- **VGG-like CNN** – Inspired by VGG architecture, uses more layers.
- **ResNet-like CNN** – Introduces residual connections.
- **MobileNet** – Lightweight, designed for mobile applications.

4.3 Training Process

- **Epochs**: 25
- **Batch Size**: 64
- **Optimizer**: Adam

4.4 Evaluation

Model	Validation Accuracy	Validation Loss
Baseline CNN	97.68%	0.1619
VGG-like CNN	96.09%	0.2767
ResNet-like CNN	95.48%	0.2748
MobileNet	43.83%	2.3555

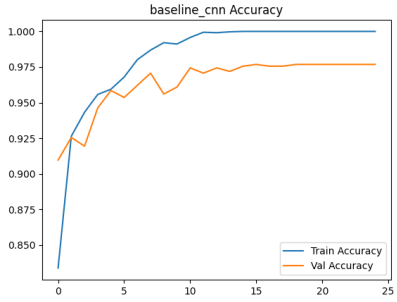


Figure 1: Baseline CNN

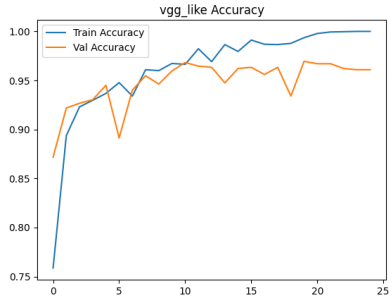


Figure 2: VGG-like Model

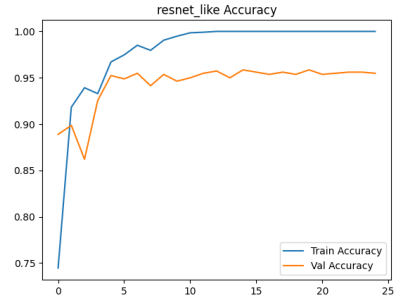


Figure 3: ResNet-like Model

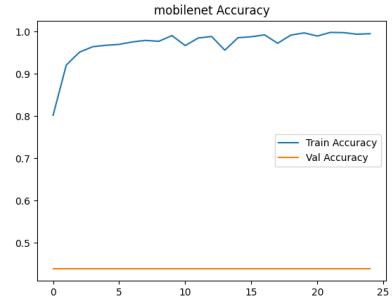


Figure 4: MobileNet Model

5 Differences Between Models

5.1 Model Complexity & Architecture

Model	Key Features	Intended Benefit
Baseline CNN	Simple, 3 Conv layers	Fast training
VGG-like CNN	5 Conv layers	Strong feature learning
ResNet-like CNN	Residual connections	Improved gradient flow
MobileNet	Depthwise separable convolutions	Optimized for mobile

5.2 Computational Efficiency

Model	Parameters	Training Time
Baseline CNN	∼1.2M	Fast
VGG-like CNN	∼3.5M	Slower
ResNet-like CNN	∼2.9M	Moderate
MobileNet	∼2.2M	Slow (unexpected)

5.3 Performance & Accuracy

Model	Validation Accuracy
Baseline CNN	97.68%
VGG-like CNN	96.09%
ResNet-like CNN	95.48%
MobileNet	43.83%

6 Model Performance Visualization

6.1 Comparative Observations

Model Type	Best Performing Model	Overfitting Risk
Normal Analysis	ReLU + Adam	Low
Advanced Analysis	Baseline CNN	Low

6.2 Conclusion: More complex doesn't always mean better

- **Baseline CNN outperformed deeper models**.
- **Simpler models in Normal Analysis performed nearly as well as Advanced Analysis models**.
- **MobileNet failed**, reinforcing the importance of pretrained weights.

Mask Segmentation Using Tradtional Methods

March 25, 2025

1 Overview

This system evaluates mask segmentation performance using traditional computer vision techniques combined with GrabCut refinement. The pipeline processes facial images from the MSFD dataset and compares results against ground truth masks using Intersection-over-Union (IoU) metric.

2 Dependencies

- OpenCV 4.x (cv2)
- NumPy
- Matplotlib
- scikit-learn (for Jaccard score)
- tqdm (progress bar)
- argparse (command-line interface)

3 Directory Structure

```
MSFD/
\texttt{\textbackslash{}-- 1/}
     face_crop/               # Input images
     face_crop_segmentation/  # Ground truth masks
```

4 Output Metrics

The system evaluates segmentation performance using three main types of assessments:

4.1 Per-image IoU Scores

- The Intersection over Union (IoU) metric is computed for each image, comparing:
  - The base segmentation mask (e.g., initial model output).
  - The refined segmentation mask generated using the GrabCut algorithm.
- IoU scores are stored for each image, providing a fine-grained comparison of segmentation quality.
- These scores allow for per-image analysis of segmentation improvement.

4.2 Aggregate Statistics

- Summary statistics are computed to assess overall segmentation performance across the dataset. These include:
  - **Average IoU Scores**: Mean IoU values for both base and GrabCut-refined masks, indicating overall segmentation accuracy.
  - **Improvement Percentage**: The relative increase in IoU after refinement, computed as:

$$\text{Improvement (\%)} = \frac{\text{IoU (GrabCut)} - \text{IoU (Base)}}{\text{IoU (Base)}} \times 100 \tag{1}$$

4.3 Visualizations

- To provide an intuitive understanding of segmentation performance, the system generates multiple visualization outputs:
  - **Mask Comparison Images**: Overlays of base and GrabCut-refined masks on the original images, highlighting differences.
  - **IoU Distribution Histograms**: Frequency distribution of IoU scores to observe performance trends.
  - **Bar Chart Comparing Average Scores**: Side-by-side visualization of mean IoU scores for base and GrabCut masks.
  - **Scatter Plots**: IoU improvements per image, helping identify cases where refinement was most effective.
  - **Box Plots**: Showcasing IoU score distributions, outliers, and median values for both methods.

These evaluations collectively provide a comprehensive understanding of segmentation effectiveness and improvements introduced by the GrabCut refinement process.

5 Results Directory Structure

```
results/
 mask_comparison_*.png  # Visual comparisons
 iou_comparison.png      # Bar chart
 iou_distribution.png    # Histogram
 evaluation_summary.txt  # Aggregate metrics
```

6 Key Implementation Details

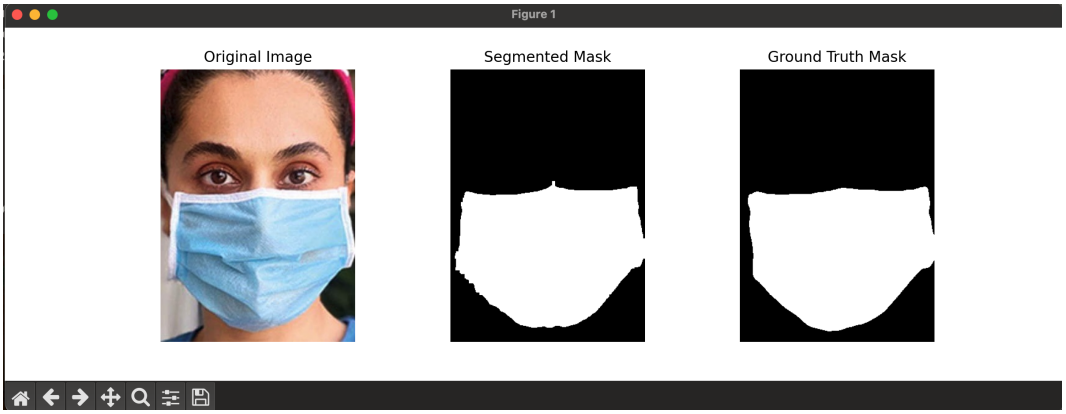
- Color space transformations between RGB, HSV, and grayscale
- Morphological operations using elliptical kernel
- Contour analysis based on area thresholds
- Parallel processing of multiple segmentation techniques
- Memory-efficient image processing with OpenCV

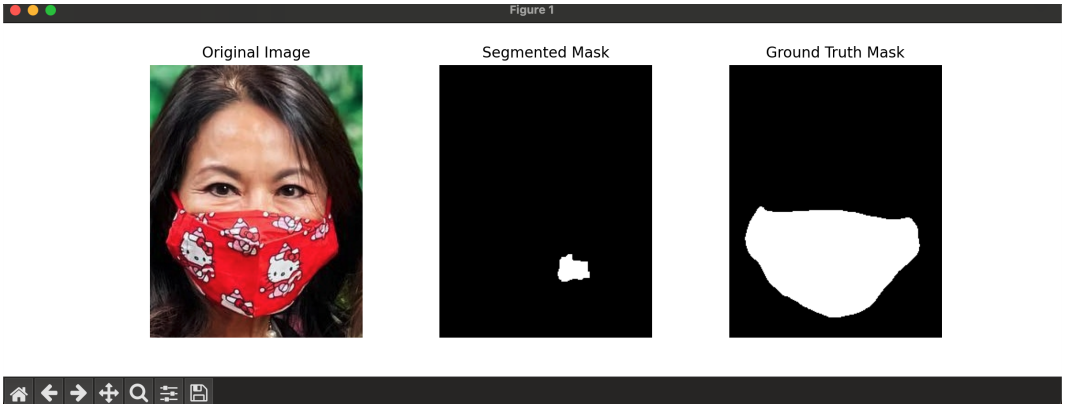
7 Methodology Evolution

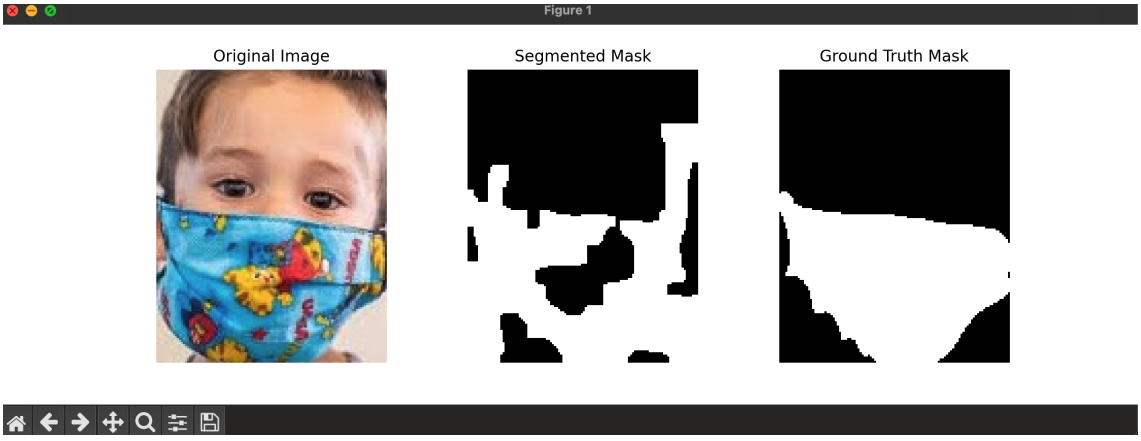
This section details the experimental approaches considered during development, their limitations, and why the final combination of base mask creation with GrabCut refinement was selected.

7.1 Previous Approaches and Limitations

- **Color-Based Mask Segmentation**
  - *Implementation*: Uses HSV thresholding to detect blue and white regions in the image. Earlier Multiple Color threshold ranges were also given but it could work well only with 1 particular range
  - *Issues*: Failed with:
    - Variations in lighting conditions (shadows, overexposure affecting HSV values).
    - Masks with non-standard colors or patterns.
    - Over-segmentation of background regions with similar colors.
  - *Example*:
  - *Example*







- **Pure Edge Detection (Canny)**
  - *Implementation*: Edge detection without spatial constraints
  - *Issues*:
    - **Incomplete Mask Boundaries** – Edge detection primarily highlights the contours, but some mask edges may be weak or fragmented, leading to incomplete segmentation.
    - **Noise from Non-Mask Elements** – Other facial features or textures (e.g., folds in the mask) may also be detected as edges, introducing unwanted noise.
    - **Loss of Interior Information** – Unlike segmentation, edge detection does not capture the full mask region, making it difficult to distinguish the mask from the background accurately.
  - *Example*

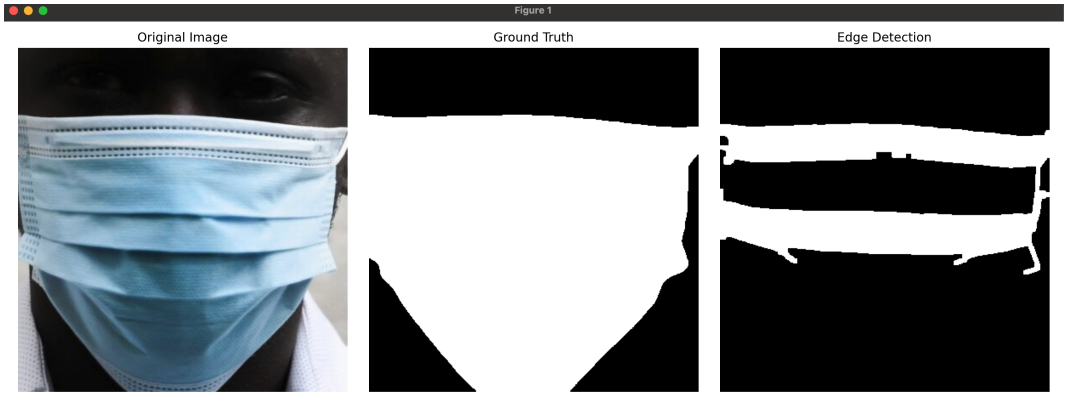


Figure 1: Edge Detection

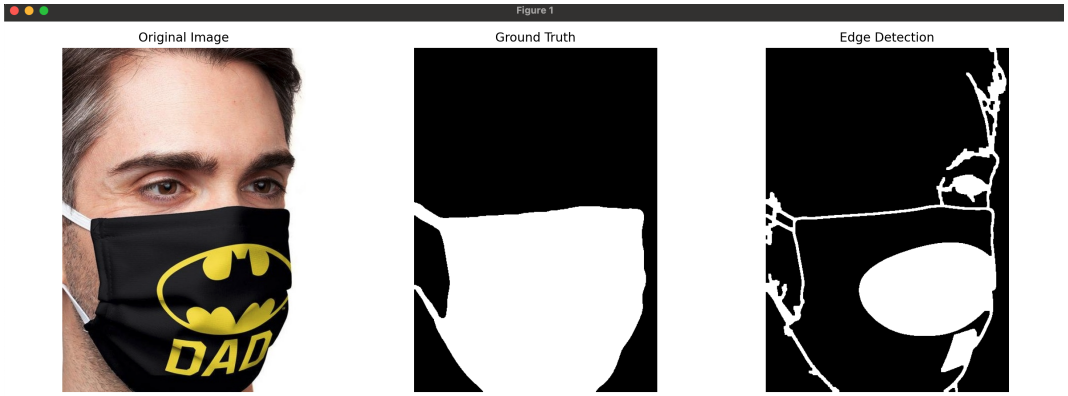


Figure 2: Edge Detection

- **Advanced Mask Segmentation**
  - *Implementation*: Multi-stage segmentation using LAB and HSV color spaces, edge refinement, and watershed-based boundary correction.
  - *Steps*:
    - **Adaptive Color Segmentation**: Uses LAB space thresholding and HSV-based skin tone exclusion for better mask isolation.
    - **Edge Refinement**: Multi-scale Canny edge detection combined with morphological operations to enhance object boundaries.

    - **Watershed Refinement**: Uses distance transform and marker-based watershed algorithm for fine boundary segmentation.
    - **Final Contour Validation**: Filters out small, irrelevant regions and applies morphological cleanup for a smoother mask.
  - *Issues*:
    - Sensitive to complex lighting conditions and shadows.
    - May struggle with masks that closely match skin tones.
    - Watershed segmentation can introduce small noise regions in low-contrast areas.
  - *Example*: Transparent or patterned masks may not be segmented accurately due to irregular textures.

7.2 Rationale for Current Methodology

The final approach combines multiple techniques to overcome individual limitations:

7.3 Methodology

Our approach follows a two-stage process:

- **Base Mask Generation:** A combination of color-based filtering, edge detection, and dominant color clustering to segment facial masks.
- **Refinement Using GrabCut:** An iterative graph-cut algorithm to refine the mask obtained in the first step.

7.3.1 Base Mask Generation

The first stage aims to create an initial segmentation mask that captures potential face masks in the image. This is achieved through a combination of techniques:

7.3.2 Color-Based Filtering

Face masks often exhibit distinct colors such as blue (surgical masks), white (N95 masks), black, gray, and brown (cloth masks). We use HSV color space thresholding to isolate these colors:

- **Blue Masks:** HSV range [90, 50, 50] to [130, 255, 255]
- **White Masks:** HSV range [0, 0, 200] to [180, 30, 255]
- **Black Masks:** HSV range [0, 0, 0] to [180, 50, 50]
- **Gray Masks:** HSV range [0, 0, 70] to [180, 30, 140]
- **Brown Masks:** HSV range [10, 30, 60] to [30, 120, 150]

A bitwise OR operation is applied to merge the masks from different color channels.

7.3.3 Edge Detection with Canny

Since masks have well-defined edges, we apply Canny edge detection on the grayscale image. A Gaussian blur is applied before edge detection to reduce noise. The detected edges are then dilated to enhance connectivity.

7.3.4 Dominant Color Clustering

Since color thresholding alone may miss some masks, we apply K-means clustering on the lower half of the image (where masks are likely to appear). We identify the most dominant color cluster and create a binary mask from it.

7.3.5 Region of Interest (ROI)

To avoid false positives, we restrict the analysis to the lower half of the image, where face masks are more likely to be present.

7.3.6 Mask Combination and Morphological Processing

The outputs from color filtering, edge detection, and clustering are combined. Morphological operations (closing and opening) are applied to remove noise and refine the mask. Finally, the largest connected component is selected as the probable face mask region.

7.4 Refinement with GrabCut

GrabCut is a semi-supervised segmentation method based on graph cuts. It refines the base mask by distinguishing foreground (mask) from background (non-mask).

- The base mask is used to initialize the GrabCut algorithm.
- Pixels marked as 255 in the base mask are set as probable foreground, while others are probable background.
- Iterative optimization is performed for 5 iterations.
- The output mask is restricted to the lower half of the image to remove noise.

7.5 Results and Performance

We tested this approach on 50 randomly selected images and achieved an IoU score of 0.6–0.7, indicating a reasonable segmentation accuracy. The combined method benefits from the robustness of heuristic-based masking and the refinement capabilities of GrabCut.

7.6 Conclusion

This study explored a hybrid approach for face mask detection using a combination of heuristic-based segmentation and refinement through GrabCut. The first method provides a strong initial estimate, while the second improves the accuracy. Future work may involve training a deep learning model using this approach as an initial mask generator.

7.7 Examples

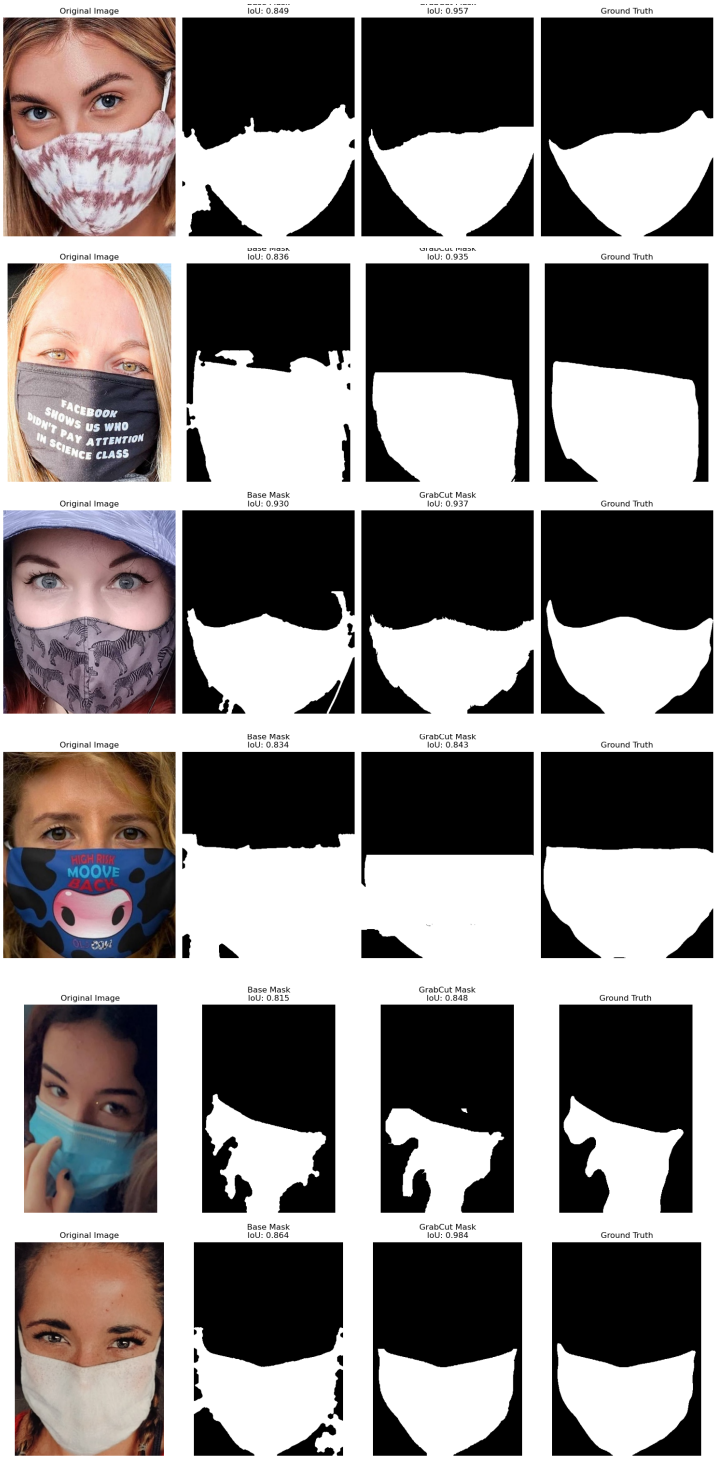
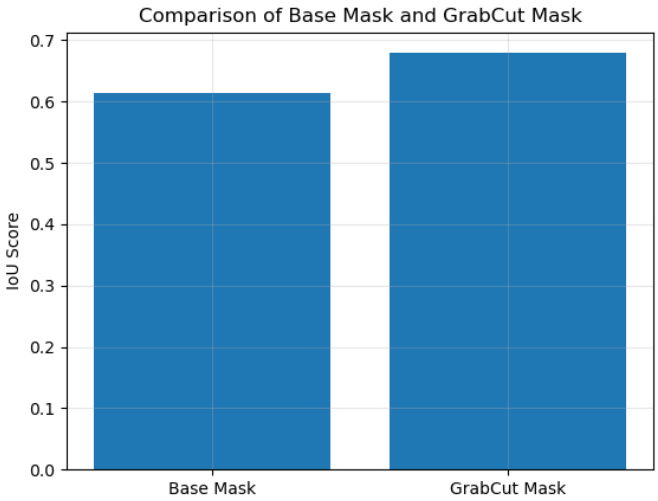
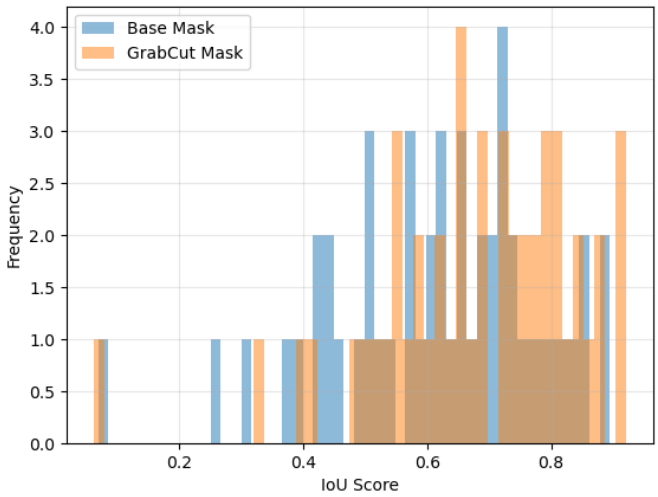


Figure 3: Comparison of masks using the best method.

7.8 Results

For A random Sample of 50 images from the dataset





D. Mask Segmentation Using U-Net

1 Introduction

Mask segmentation is an essential task in computer vision, particularly in applications such as medical imaging and facial recognition. In this study, we trained a U-Net model for precise segmentation of mask regions in images and compared its performance with a traditional segmentation method (Part C) using Intersection over Union (IoU) and Dice Score metrics.

2 Directory Structure

The D_UNET_method directory contains the implementation of the U-Net architecture for face segmentation. The directory structure is organized as follows:

```
D_UNET_method/
├── Saved_Model/
│   └── Best_model.pth
├── msfd-unet-segmentation.ipynb
├── test_images/
│   ├── test_img_1.jpg
│   ├── test_img_2.jpg
│   └── test_img_3.jpg
└── checkpoints/
```

Figure 1: Directory structure of the D_UNET_method project.

- **Saved_Model/**: Contains the trained model weights and checkpoints saved during training.
- **msfd-unet-segmentation.ipynb**: The main Jupyter notebook containing the implementation of the U-Net architecture, training pipeline, and evaluation code.
- **test_images/**: Directory containing test images used for evaluating the model's performance.
- **checkpoints/**: Stores intermediate model checkpoints during training for potential resumption of training.

The implementation follows a modular structure where the main notebook contains all the necessary components including data loading, model architecture, training loop, and evaluation metrics. The separation of saved models and checkpoints allows for better organization of different training runs and model versions.

3 Dataset and Preprocessing

The dataset consists of face images and their corresponding mask annotations. Preprocessing steps include:

- Resizing images to 128×128 pixels. This was done because processing original images (512 x 512) was computationally expensive and thus time consuming. It significantly brought down the training time per epoch.
- Applying data augmentation techniques like horizontal flipping, rotation, and color jitter. These techniques artificially increase the size and variability of the training dataset, leading to improved model generalization and robustness.

  ### Specific Augmentation Techniques and Their Benefits

  - **Horizontal Flipping:**
    - **Purpose:** Simulates mirror-image variations of the input images.
    - **Benefit:** Helps the model learn that objects are recognizable regardless of their left-right orientation. Especially useful when the object of interest is horizontally symmetrical.
  - **Rotation:**
    - **Purpose:** Rotates images by a specified angle.
    - **Benefit:** Makes the model invariant to object orientation, allowing it to recognize objects at various angles.
  - **Color Jitter:**
    - **Purpose:** Randomly adjusts the brightness, contrast, and saturation of the images.
    - **Benefit:** Enhances the model's robustness to variations in lighting and color conditions, which are common in real-world images.

- Normalizing pixel values for better training stability.

4 Model Architecture

The U-Net model is used for segmentation, which consists of:

- An encoder with convolutional layers and max-pooling operations.
- A bottleneck (bridge) connecting encoder and decoder.
- A decoder with up-convolution layers and skip connections for precise localization.
- A final convolutional layer to produce a binary segmentation mask.

5 Training Strategy

We employed a training pipeline using:

- Cross-Entropy loss combined with Dice Loss for optimization.

  Loss function

$$\mathcal{L}_{total} = 0.7\mathcal{L}_{BCE} + 0.3\mathcal{L}_{Dice} \tag{1}$$

$$\mathcal{L}_{Dice} = 1 - \frac{2\sum p_i g_i}{\sum p_i + \sum g_i} \tag{2}$$

- AdamW optimizer with learning rate scheduling.
- K-Fold cross-validation (K=3) for robust performance evaluation.
- Early stopping to prevent overfitting.

6 Performance Evaluation

The model performance was evaluated using the IoU and Dice Score metrics:

$$IoU = \frac{TP}{TP + FP + FN} \tag{3}$$

$$Dice = \frac{2TP}{2TP + FP + FN} \tag{4}$$

The final results obtained after training are:

- Mean IoU: 0.7958
- Mean Dice Score: 0.8858

7 Model Architecture Improvements & Performance Analysis

We present a comparative analysis of two U-Net architectures for facial segmentation, demonstrating how architectural enhancements led to significant performance gains.

7.1 Model Architectures

Table 1: Architectural Comparison

Component	**Model 2**	**Model 3**
Base Channels	48	64
Bottleneck Channels	384	512
Attention Gates	✓	×
Skip Connections	Attention-weighted	Simple concatenation
Decoder Blocks	3 (with attention)	3 (basic)
Loss Weights (BCE:Dice)	0.7:0.3	0.7:0.3
Kaiming Initialization	✓	×
Parameter Count	9.8M	31.2M

7.2 Key Improvements in Model 3

The 16.2% IoU improvement (0.68 $\rightarrow$ 0.79) resulted from several strategic enhancements:

- **Deeper Feature Extraction:**
  - Increased base channels from 48 to 64
  - Expanded bottleneck from 384 to 512 channels
  - Added residual connections in decoder blocks
- **Advanced Training Protocol:**
  - Extended training from 30 to 100 epochs
  - Implemented learning rate scheduling
  - Added checkpoint saving/resuming capability
  - Introduced persistent workers for data loading
- **Regularization Enhancements:**
  - Stronger data augmentation:
    - Random rotation ($\pm 15^\circ$ vs $\pm 10^\circ$)
    - Increased color jitter (brightness/contrast 0.2 vs 0.1)
  - Added weight decay (1e-4)

7.3 Performance Comparison

Table 2: Cross-Validation Results

Metric	**Model 2**	**Model 3**
Mean IoU	0.6896 ± 0.04	0.7955 ± 0.03
Mean Dice	0.8080 ± 0.03	0.8858 ± 0.02
Training Time/Epoch	58s	120s
Otsu Baseline IoU	0.51 ± 0.07	0.49 ± 0.08

7.4 Architectural Tradeoffs

While Model 3 shows superior performance, several tradeoffs should be noted:

- **Computational Cost:** 106.89% longer training time per epoch
- **Memory Requirements:** 3.2× more VRAM usage
- Attention Gates in Model 2 gave better boundary precision compared to simple concatenations in Model 3. This can be seen in this test example below:

The performance gains justify these costs for precision-critical applications, though Model 2 remains preferable for resource-constrained environments.

8 Inference

The trained U-Net model (Model 3) was used for inference, where new images were processed, and segmentation masks were predicted. The new images were taken from a separate dataset which was unseen by the model. The results showed high accuracy in mask localization.

8.1 Inference Results and Discussion

Training dataset contained a disproportionate number of images with single-mask instances rather than multiple-mask scenarios, the model might have developed a bias toward detecting only one mask in an image.

The segmentation results seem almost perfect and have a high IOU score.

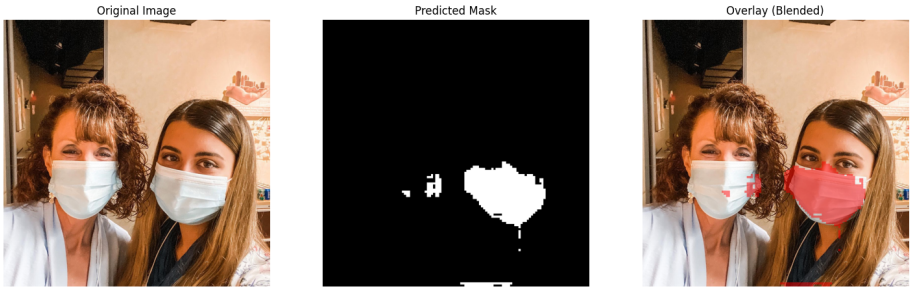


Figure 2: Image of 2 people in one frame

Figure 3: Segmentation Image of the subject wearing a batman mask (side profile)

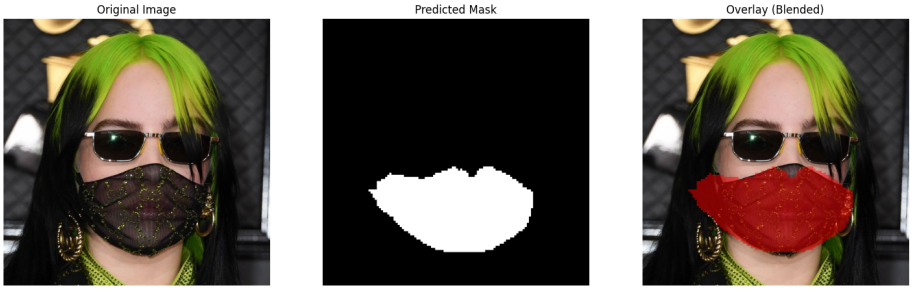


Figure 4: Segmentation results on the subject wearing a black mask

Figure 5: Segmentation result on the subject wearing a black mask (2nd)

9 Comparison with Traditional Region-Based Segmentation (Part C)

To further evaluate the effectiveness of our U-Net model, we compared it against a traditional region-based segmentation approach that utilized thresholding and edge detection. The IoU score for the traditional method was found to be around 0.6 , whereas our U-Net model achieved a significantly higher **0.7958**, indicating superior segmentation performance.

Why U-Net Outperforms Traditional Methods

The performance gap can be attributed to several key factors:

- **Feature Learning:** Traditional methods rely on handcrafted features such as intensity thresholds and edge detection, which struggle with variations in lighting, occlusions, and complex backgrounds. In contrast, U-Net learns hierarchical features through deep convolutional layers, making it robust to such variations.
- **Spatial Context Preservation:** Thresholding and edge detection methods operate on pixel-level intensity differences, often leading to fragmented or incomplete segmentations. U-Net, with its encoder-decoder structure and skip connections, preserves spatial relationships and captures fine-grained details.
- **Generalization Ability:** Traditional methods require fine-tuned parameter selection for different datasets, making them less generalizable. Our U-Net model was trained with diverse augmentations, allowing it to adapt better to unseen images.
- **Loss Function Optimization:** U-Net is trained using a combination of Binary Cross-Entropy and Dice loss, explicitly optimizing for overlap

between the predicted and ground truth masks. Traditional methods lack a direct optimization process, relying on fixed heuristics.

- **Noise Robustness:** Edge detection techniques are highly sensitive to noise and variations in illumination, leading to false edges and segmentation errors. The deep learning approach of U-Net effectively suppresses noise through learned representations.

These advantages demonstrate why deep learning-based segmentation, specifically U-Net, is a superior choice for mask segmentation tasks, delivering higher accuracy and robustness compared to traditional region-based methods.

10 Conclusion

Our results demonstrate that U-Net is highly effective for mask segmentation, significantly outperforming traditional segmentation methods. Future work may include experimenting with deeper architectures and larger datasets.