Mask Detection Using Transfer Learning

Yuyao Wang

Abstract

This report details the development of a mask detection system using deep learning models, specifically focusing on transfer learning techniques applied to MobileNetV2 and ResNet architectures. The aim is to distinguish between individuals wearing masks and those not wearing masks in real-time environments. The system was trained using a dataset of labeled images, utilizing techniques such as convolutional neural networks (CNNs), data augmentation, and optimization algorithms. In this report, we provide a detailed explanation of the methodologies and algorithms involved, with a focus on the mathematical foundations behind transfer learning, CNN architectures, and model fine-tuning.

Contents

1 Introduction

The COVID-19 pandemic has highlighted the importance of mask-wearing as a public health measure. This project aims to develop a deep learning-based system to detect whether individuals are wearing masks in real time. The project is particularly relevant in settings such as airports, hospitals, and offices where compliance with mask regulations is crucial.

The challenge is to train a model that can classify images into two categories: wearing a mask or not wearing a mask. This involves adapting pre-trained models using transfer learning and fine-tuning them for the binary classification task. In this report, we will delve into the mathematical methodologies used, including CNN architectures, transfer learning, and the optimization techniques applied to fine-tune the model.

2 Methodology

2.1 Problem Definition

We define the mask detection task as a binary classification problem. Let the input be an image represented as a matrix $\mathbf{X} \in \mathbb{R}^{m \times n \times 3}$, where m and n are the height and width of the image, and 3 denotes the RGB color channels. The goal is to predict a binary label $y \in \{0, 1\}$, where:

$$y = \begin{cases} 1 & \text{if the individual is wearing a mask,} \\ 0 & \text{if the individual is not wearing a mask.} \end{cases}$$

Given a training dataset $\mathcal{D} = \{(\mathbf{X}_i, y_i)\}_{i=1}^{N}$, where N is the number of samples, the task is to find a model $f(\mathbf{X}; \theta)$ with parameters θ that minimizes the classification error:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{X}_i; \theta), y_i)$$

where $\mathcal{L}$ is the loss function (cross-entropy loss, defined below).

2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are at the core of the mask detection model. The basic idea behind CNNs is to automatically extract features from the input image using convolutional layers. Each convolutional layer applies a set of filters to detect features such as edges, textures, and shapes.

Mathematically, a convolution operation is defined as:

$$\mathbf{X}^{(l+1)} = \sigma(\mathbf{W}^{(l)} * \mathbf{X}^{(l)} + \mathbf{b}^{(l)})$$

where:

- $\mathbf{X}^{(l)}$ is the input at layer l,
- $\mathbf{W}^{(l)}$ is the convolution filter (or kernel),
- $\mathbf{b}^{(l)}$ is the bias term,
- σ is the activation function (typically ReLU), and
- $*$ denotes the convolution operation.

The output of a convolution operation is passed through an activation function, typically the Rectified Linear Unit (ReLU):

$$\sigma(x) = \max(0, x)$$

which introduces non-linearity into the model, allowing it to learn more complex features.

To reduce the spatial dimensions of the feature maps, pooling layers are used. The most common pooling operation is max-pooling, which selects the maximum value from a patch of the feature map:

$$\mathbf{X}^{(l+1)} = \text{max-pool}(\mathbf{X}^{(l)})$$

Pooling layers help in reducing the dimensionality, making the model more computationally efficient and less prone to overfitting.

2.3 Transfer Learning

Transfer learning allows us to leverage the knowledge gained by a model trained on a large dataset (such as ImageNet) for a new task with a smaller dataset (mask detection). We use pre-trained models such as MobileNetV2 and ResNet.

Mathematically, transfer learning can be expressed as:

$$\theta = \theta_{pre} + \Delta\theta$$

where θ_{pre} are the parameters learned from the pre-trained model, and $\Delta\theta$ are the updates learned from fine-tuning on the mask detection dataset.

In practice, we freeze the early layers of the model (which have learned general image features such as edges and textures) and only update the parameters in the final few layers. This reduces the risk of overfitting and accelerates the training process.

2.4 Loss Function: Cross-Entropy

The binary cross-entropy loss function is used to train the model. It is defined as:

$$\mathcal{L}(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

where:

- y is the true label (0 or 1),
- $\hat{y}$ is the predicted probability of the label being 1 (wearing a mask).

Minimizing this loss function encourages the model to predict probabilities that are close to the true labels.

2.5 Data Augmentation

To improve the model's generalization ability, we apply various data augmentation techniques during training. These transformations help simulate real-world variability in the data, such as different head orientations and lighting conditions. The key augmentation techniques include:

- Random Rotation: Rotate the image by a random angle $\theta \in [-15°, 15°]$.
- Horizontal Flip: Flip the image horizontally with a probability of 0.5.
- Color Jittering: Apply random changes to brightness, contrast, and saturation.
- Normalization: Normalize the image pixel values using the mean and standard deviation of the ImageNet dataset.

These augmentations can be mathematically described as transformations T applied to the input image $\mathbf{X}$:

$$\mathbf{X}' = T(\mathbf{X})$$

where T is the set of augmentations applied during training.

3 Algorithm

The complete training algorithm is described below:

Algorithm 1 Training Procedure for Mask Detection Model

Input: Pre-trained model $f(\mathbf{X}; \theta_{pre})$, dataset $\mathcal{D}$, learning rate α, number of epochs E
Initialize: Replace final fully connected layer with a binary classification layer
for epoch $e = 1$ to E **do**
    for each mini-batch $(\mathbf{X}_b, y_b)$ from $\mathcal{D}$ **do**
        Forward pass: $\hat{y}_b = f(\mathbf{X}_b; \theta)$
        Compute loss: $\mathcal{L} = \frac{1}{|b|} \sum_{i \in b} \mathcal{L}(\hat{y}_i, y_i)$
        Backward pass: Compute gradients $\nabla_\theta \mathcal{L}$
        Update weights: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
    end for
    Evaluate model on validation set
end for
Return: Trained model $f(\mathbf{X}; \theta)$

The model is trained using mini-batch gradient descent. For each mini-batch, we compute the forward pass, calculate the loss, perform the backward pass to compute gradients, and update the model parameters using the gradients.

4 Optimization: Adam

In this project, we used the Adam (Adaptive Moment Estimation) optimizer to update the model's weights during training. Adam is a popular optimization algorithm in deep learning because it combines the advantages of two other commonly used optimizers: AdaGrad and RMSProp. It adapts the learning rate for each parameter, making it particularly effective for handling sparse gradients and noisy data.

Adam Update Rules

Adam computes individual adaptive learning rates for each parameter based on the first and second moments of the gradients. The update equations are as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta \mathcal{L}(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\nabla_\theta \mathcal{L}(\theta_t)\right)^2$$

where:

- m_t is the first moment estimate (the exponentially decaying average of past gradients),
- v_t is the second moment estimate (the exponentially decaying average of past squared gradients),
- $\nabla_\theta \mathcal{L}(\theta_t)$ is the gradient of the loss function $\mathcal{L}$ with respect to the parameters θ at time step t,
- β_1 and β_2 are hyperparameters controlling the decay rates for the moment estimates.

Next, bias-corrected estimates of the moments are computed as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

These bias-corrected estimates are used to mitigate the effects of the initialization of m_t and v_t close to zero during early stages of training.

Finally, the parameter updates are performed using the following equation:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where:

- α is the learning rate,
- ϵ is a small constant added to prevent division by zero (typically $\epsilon = 10^{-8}$).

4.1 Benefits of Adam

Adam offers several advantages over simpler optimization algorithms like Stochastic Gradient Descent (SGD): 1. Adaptive Learning Rates: Adam computes individual learning rates for each parameter, which helps in faster convergence and better handling of varying gradient magnitudes. 2. Momentum: The first moment estimate (m_t) adds momentum to the gradient updates, allowing the optimizer to move more smoothly through the loss landscape and avoid getting stuck in local minima. 3. Bias Correction: The bias-corrected moment estimates ($\hat{m}_t$ and $\hat{v}_t$) provide more accurate estimates of the gradient moments, especially during the early phases of training. 4. Robust to Noisy Data: The combination of the first and second moments allows Adam to be more robust to noise in the gradient, leading to more stable updates.

4.2 Hyperparameter Tuning

The default values of Adam's hyperparameters are typically sufficient for most tasks: - $\beta_1 = 0.9$: Controls the decay rate of the moving average for the gradient. - $\beta_2 = 0.999$: Controls the decay rate for the squared gradient. - $\epsilon = 10^{-8}$: Prevents division by zero.

However, in this project, the learning rate α was adjusted depending on the performance during training. Through experimentation, we set the learning rate to $\alpha = 0.001$, which provided a good balance between fast convergence and stable training.

4.3 Adam's Performance in Mask Detection

In the mask detection problem, Adam played a key role in efficiently optimizing the parameters of the deep learning model (MobileNetV2 and ResNet). The adaptive nature of Adam was particularly beneficial in the later stages of training, where the gradients became smaller, and a dynamically adjusted learning rate allowed the model to continue improving without oscillating around local minima.

Additionally, Adam's built-in momentum allowed it to navigate the complex loss landscape efficiently, finding better optima in fewer iterations compared to basic SGD. This led to faster convergence and better generalization, as observed by the improvement in validation accuracy over epochs.

5 Training and Convergence

During the training process, the model was optimized using the Adam optimizer over a span of 20 epochs. The training loss decreased steadily, while the validation accuracy improved. The combination of the Adam optimizer with data augmentation helped mitigate overfitting, ensuring that the model generalizes well to unseen data.

We monitored the training loss and validation accuracy at every epoch, ensuring that the model was converging towards an optimal solution. Convergence was achieved after 15 epochs, and training was stopped after 20 epochs to prevent overfitting.

6 Conclusion

This report presents the development of a deep learning-based mask detection system using transfer learning and optimization techniques. By leveraging pre-trained models such as MobileNetV2 and ResNet, and optimizing them using the Adam optimizer, we achieved high accuracy in distinguishing between individuals wearing masks and those without. Through the combination of data augmentation, transfer learning, and efficient optimization, the model demonstrates strong generalization ability, making it suitable for real-world deployment in public spaces.