Large-Margin Saliency-aware Binarized CNN for Monkeypox Virus Image Classification

Debojyoti Biswas  
bishaldebojyoti@gmail.com

Texas State University  https://orcid.org/0000-0002-8842-0207

Jelena Tesic
Texas State University

Research Article

Keywords: Monkeypox Classification, Lightweight CNN, Binarized CNN, Power Efficient DNN, Deep Learning Approaches to Detect Mpox

Posted Date: February 27th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-3990337/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: The authors declare no competing interests.
Large-Margin Saliency-aware Binarized CNN for Monkeypox Virus Image Classification

Debojyoti Biswas\textsuperscript{1} and Jelena Tešić\textsuperscript{2}

\textsuperscript{1,2}Department of Computer Science, Texas State University, 601 University Dr, San Marcos, 78666, TX, USA.

Contributing authors: debojyoti_biswas@txstate.edu; jtesic@txstate.edu;

Abstract

The recent widespread increase of the Mpox (formerly monkeypox) virus infections in the South Asian and African countries has raised concerns among medical professionals regarding the potential emergence of another pandemic in those regions. With the number of available test kits surpassing the count of positive/probable cases, there is a pressing need to develop a robust and lightweight classifier model that can alleviate the burden of physical testing kits and expedite the detection process. The existing state-of-the-art primarily focuses on achieving high accuracy in modeling Mpox without considering factors such as modeling suitability, real-time inferencing, and adaptability to resource-constrained CPU-only mobile devices. In this research, we propose a novel lightweight binarized DarkNet53 model, referred to as BinaryDNet53, which is approximately $\sim 20 \times$ more computationally efficient and $\sim 2 \times$ more power-efficient than the current state-of-the-art. This model demonstrates smooth detection capabilities when deployed on small handheld or embedded devices. Our work introduces large-margin feature learning and weighted loss calculation to enhance results, particularly on complex samples. We conduct experiments using the latest MSLD v2.0 dataset, showcasing the superiority of the proposed model over state-of-the-art models based on classification and computational metrics, including Watt power consumption, required memory, and GFLOPS.

\textbf{Keywords:} Monkeypox Classification, Lightweight CNN, Binarized CNN, Power Efficient DNN, Deep Learning Approaches to Detect Mpox

1 Introduction

Recent advancements in artificial intelligence (AI) have solved many challenging computer vision problems, ranging from image classification \cite{1}, segmentation \cite{2}, object detection \cite{3, 4}, etc., in consumer, medical, and remote-sensing images. There has been significant improvement in medical science due to AI-powered devices and utilities. Deep Learning is a sub-field of artificial intelligence that can assist us in extracting important features/information from images without human participation. In the field of medical science, different kinds of imaging techniques are primarily used to assist medical professionals and clinics in making diagnoses of a wide variety of diseases, such as brain cancer \cite{5}, respiratory conditions like pneumonia \cite{6}, as well as the recent COVID-19 virus \cite{7}. Although the infection of COVID-19 is not entirely over yet, the Mpox virus has caused concern in many parts of the world. Mpox (formerly Mpsx) is announced as a medical emergency \cite{8} due to its rapid outbreak across 40 countries. From a clinical perspective, the Mpox virus poses
characteristics similar to chickenpox, cowpox, and smallpox. There is an urgent need to start building lightweight and computationally efficient detection models because there are increasing cases of minimal testing kits available in clinics. A deep learning-based mobile testing kit can help us address problems such as the inadequate number of clinic experts, the high cost of kits, and, most importantly, the prolonged waiting time involved in the virus detection process.

Recently, Kundu et al. used a vision transformer-based classification model for Mpox classification [1]. This work aimed to evaluate several machine learning and deep learning models toward achieving optimal results. The optimal result in that work came from the transformer and ResNet50 models. However, ResNet50 and Transformer models are heavy in computation, with the ResNet50 requiring around 86.47 Giga Floating-point Instructions Per Second (GFLOPS) and 25.6 Million learnable parameters, making it a poor choice for mobile devices.

**Contribution**

In this paper, we propose a lightweight and computationally efficient model for real-time Mpox detection whose effectiveness matches state-of-the-art performance with higher power consumption. The model’s low power consumption and faster detection rate make it a strong candidate for real-time virus detection in mobile devices. First, we propose to use a large-margin hybrid network with Maximum margin hyperplane learning to distinguish Mpox from Chickenpox correctly [9]. The examples of similar images with different diseases are provided in Figure 1. Second, we propose using the feature vector’s saliency to calculate the image’s complexity level. It has been shown that the amount of neuron activation is a good indicator of pixel diversity and texture variance [10]. We propose to use this information to calculate the image complexity and put more weight on more complex cases in the loss function. Third, we binarize all weights and biases of the DarkNet53 baseline model, as a CNN with binary weights and biases is \( \sim 20 \times \) more computationally efficient than an equivalent network with single-precision weights. Thus, all calculations use +, − operators only during the forward pass, significantly reducing the computations. The paper’s contributions are:

1. **Novel framework:** We propose a large-margin-based hybrid network to address the high inter-class similarity.

2. **Saliency-aware Weighting:** In this process, we put more weight on complex sample learning and less on easy examples. The complexity of the sample is calculated from the number of neuron activations. The experiments show that weighted loss calculation significantly improves the actual positive virus detection performance.

3. **Binarization of Weights and Biases**

   Inspired by the work of XNOR-Net [11], we convert the traditional DarkNet53 model as a binary CNN model. The Binary version of the DarkNet53 model requires only 8.82 GFLOPS and 0.087 Million learnable parameters, which is significantly less than traditional DenseNet201 and ResNet50 CNN models.

4. **Comprehensive Model Analysis**

   Apart from the classification metrics (Accuracy, Precision, and Recall), we include several other important evaluation criteria such as the amount of power (in Watt) needed for the model training and inference, the amount of GFLOPS required for the model computation, and the lastly the size of the model in system memory.
The rest of this article is organized as follows. Section 2 summarizes related work; Section 3 introduces the proposed Binary-DarkNet53 method, describing the large-margin feature learning approach and the saliency-aware loss function in the pipeline. Section 4 introduces the experimental setup and dataset. Section 5 evaluates the proposed framework, and the ablation study is presented using the latest virus detection benchmarks and computational performances. Finally, Section 6 summarizes the quantitative findings and outlines future works.

2 Related Work

Convolutional Neural Networks (CNN) have been playing a key role in extracting rich features from different types of images. Some deep CNNs like ResNets [12], DenseNet [13], DarkNet53 [3], and MobileNetv2 [14] showed promising results on different computer vision tasks. The success of CNN is also evident in the field of medical image classification [6, 15–17].

The initial use of CNNs in medical science was for classifying simple tasks, such as brain tumors or pneumonia, from X-ray and MRI reports. In early 2007, researchers often used the traditional Machine Learning approach for tumor classification. S. Chaplot et al. [18] proposed a novel technique for classifying magnetic resource images of the human brain, which utilizes wavelets to support vector machine and neural system self-organizing maps. Later, Usman et al. [19] proposed a similar wavelet-based multimodal classification of MRI images. Their work extracted feature intensity, intensity differences, local neighborhood, and wavelet texture from multimodal data. Next, a random forest classifier was integrated to perform classification tasks. However, after the improvement of deep learning networks, CNN became the standard feature extraction method for image data. Afshar et al. [20] developed a capsule network (CapsNet) model to classify brain tumors on the Figshare dataset efficiently. In this work, authors improve the performance by using the context information through the spatial relationships between the tumor and its surrounding tissues, which earlier CNN-based classification models ignored. On the other hand, Abiwinanda et al. [21] generated seven distinct CNN variations without segmentation for brain tumor classification. Compared to earlier models, the second variant of their model achieved the optimal training and testing accuracy of 98.51% and 84.19% on the Fighsare brain MRI dataset, respectively.

A recent successful application of CNN in medical science is the early detection of COVID-19 patients from Chest X-rays. It was found that many COVID-19 patients pose symptoms of COVID-19 viral pneumonia. As the number of Reverse-transcription polymerase chain reaction (RT-PCR) testing kits was in limited supply, it became urgent to develop a rapid screening method that is accurate, fast, and cheap. Toward this goal, Abbas et al. [22] utilized a CNN model known as DeTraC to classify COVID-19 using chest X-ray images. On the other hand, Narin et al. [23] integrated two deeper models for organizing COVID-19 and standard CXR images obtained from the public domain datasets. The authors used 100 images for the experiments, and the models achieved 97% and 87% accuracy from InceptionV3 and InceptionResNetV2, respectively. Moreover, to gain more improved results, Wang et al. [24] proposed an inception migration-learning model for the classification of 453 confirmed cases of COVID-19 with previously diagnosed traditional pneumonia.

Due to the success of CNNs in earlier applications, it was the first choice for researchers to perform Mpox classification. Ali et al. [16] used several pre-trained CNN feature extractors such as ResNet50, InceptionV3, and VGG-16 models for the classification of Mpox and other similar diseases on the Mpox Skin Lesion Dataset (MSLD). The MSLD dataset was developed and released by the authors of the same paper. Like the previous work, Islam et al. [25] also used pre-trained ResNet50, DenseNet121, MobileNet-V2, and several other models to examine the classification performance on Mpox datasets. The latest dataset released by Ahsan et al. [26] caught the researcher’s attention due to its completeness covering diverse virus-infected samples. In the work, Ahsan et al. proposed a modified VGG16 model, which included two separate studies. The proposed custom VGG16 achieved an accuracy of 97±1.8% and 88±0.8% for studies one and two, respectively. Additionally, they used Local InterpreTable Model-Agnostic Explanations (LIME) to
explain the model’s feature extraction and prediction strategy. Finally, Irmak et al. [27] use pre-trained CNN networks, like MobileNetV2 and VGG models, to categorize Mpox infection on the Mpox Skin Image Dataset (MSID) dataset. The optimal result was achieved from the MobileNetv2 model with an accuracy of 91.28%. Bala et al. propose MonkeyNet [28] with DenseNet201 detector of the Mpox virus, evaluating the proposal on the previously mentioned MSID dataset. The model achieved 93.19% of accuracy on the original test images from the dataset. However, the computational memory needed for running inference and the energy/power requirement for deploying the device were not reported. In summary, state-of-the-art relies on only pre-trained CNN networks for classification. They fail to address complex samples where inter-class similarity is an issue. Also, none of these works discusses the feasibility of deploying the models in a mobile device where low-power consumption is in demand.

3 Methodology

The success of deep learning algorithms lies in the fact that they can automatically extract hidden and complex features efficiently from images and videos, which is not often possible by technicians or experts. Several Convolutional Neural Network (CNN) architectures have been proposed for feature extraction, e.g., AlexNet, ResNet, MobileNet, DarkNet, etc. The deep learning model learns a large number of parameters from a large amount of training data. The advances in software and hardware support enabled deep neural network-driven solutions for breast and brain cancer [5], pre-COVID pneumonia detection [6], efficient automated creation of the MRI report [19], chickenpox classification [9]. It is important to note that pixel-level semantic information plays a critical role in medical and healthcare systems because subtle changes in pixel color can change the final decision (see Figure 1). Keeping that in mind, we carefully design a framework that preserves low-level semantic information efficiently and achieves optimal performance. In the following subsection, we will describe the baseline model we used for our proposed model.

3.1 Baseline Architecture

We propose to use DarkNet53 with the Cross Stage Partial (CSP) module for its proven performance in YOLOv3 architecture. CSP DarkNet53 [3] can detect small pixel information in more detail while preserving spatial information in later layers of CNN [10]. Our baseline architecture is illustrated in Figure 2, where we have three main parts: 1) Input, 2) Feature Extractor Backbone, and 3) Binary Loss Function. As input, we send original and augmented versions of the Mpox sample from the MSLD v2.0 dataset. The actual size of the data samples is ~ 224 × 224; we downsize the samples to 124 × 124 pixels before we send the samples to the feature extractor. Next, we perform image normalization using Eq. 1.
Normalized Image = \frac{X - \min(X)}{\max(X) - \min(X)} \quad (1)

CSP DarkNet53 backbone starts with a 2D Convolution (Conv2D) operation. Next, we have several Conv2D followed by CSP [29] module blocks (Conv2d+CSP Bottleneck). Adding CSP blocks in DarkNet53 architecture aims to represent complex features effectively by enabling richer gradient combinations with low computational costs. The architecture of the CSP block is illustrated in Figure 5. Here, we see that the feature map from the base layers gets split into two branches through the channel of the base feature. The former part of the channel blocks is directly connected to the partial transition block; however, the latter part goes through a dense block (such as Residual blocks with bottleneck). Inside the transition block, first, we perform some transition on the Residual block output, then we concatenate the transition output with the part 1 block output. Finally, another transition is performed on the combined feature and passes to the next block in the architecture.

We have four Conv2D+CSP blocks, one after one, in the bottom-up baseline architecture. Then, we have a Spatial Pyramid Pooling Fast (SPPF) layer following CSP blocks. The SPPF aims to improve the calculation speed by using a single pooling layer instead of multiple pooling layers in the SPP module and adding two pooling operations of different sizes to obtain more contextual information. After SPPF blocks, we start the top-down architecture by concatenating later layers of CSP blocks with the earlier layers of CSP blocks. The concatenation helps us to retain semantic and spatial features in the final feature vectors.

The output from the last layer of the CSP Bottleneck block is flattened and fed to a fully connected layer to reduce the feature dimension to 1024. The final 1024 dimension feature vector is fed into the sigmoid function, where we predict the class probability from 0 to 1. Finally, the probability scores and the ground truth class labels are provided for the Binary Cross-Entropy (BCE) loss function for calculating the training loss. The Eq. 2 shows the calculation for BCE loss, where \( y \) is the true label (0 or 1) and \( \hat{y} \) is the predicted probability.

\[
\text{BCE} (y, \hat{y}) = - (y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})) 
\] (2)

3.2 Proposed Architecture

Our goal is to outperform State-of-the-art classification performance with as low computational and power consumption as possible for Mpox with proposed lightweight CNN architecture with binarized weights and bias, as illustrated in Figure 6. ResNets, DenseNets, or DarkNets work fine for classification tasks, but they are computationally heavy and not suitable for resource-constrained devices [30]. The proposed architecture offers five significant improvements: 1) Binarized weights and bias, 2) Reduction of CSP Blocks, 3) Feature Fusion, 4) Image Complexity Score, and 5) Large Margin SVM classifier.

1. Binary Weights and Bias: Inspired by the work of Rastegari et al. [11], we incorporate
We propose to calculate the saliency-aware image complexity score, as introduced in [10, 31], to estimate the complexity of the image. Equation 5 estimates $I_{\text{sum}}$, the sum of all pixel intensity from the image feature vector. Next, to avoid the issue of gradient explosion, we restrict the value of the total image activation within a stable range. The total operation is denoted with $ICS$ in Eq. 5. The $ICS$ for each image is later used in the Large-Margin Loss calculation as a weight for the corresponding image. Here, $b_l$ and $u_b$ are the lower and upper bounds of the $ICS$ value.

$$ICS(I_{\text{sum}}) = \min(\max(I_{\text{sum}}, b_l), u_b)$$ (5)

### 4. Image Complexity Score (ICS)

**2. Reduction of CSP Block:** We removed the CSP blocks from the deeper part of the model, focusing on generating high-level features. The high-level feature represents the abstract view of the object at different scales. For healthcare image samples, the challenge is to preserve the pixel-level features because the disease symptoms heavily depend on the color, texture, and shapes. These critical features are mainly generated by CNN’s surface layers or low-level features. From Figure 4, we can see very little diversity at the abstract level, so extracting features at a less abstract level should show optimal performance compared to the baseline model. We also removed several residual connections; it is evident that the scale variation is not very high for Mpox-infected areas in the samples, so it is not very useful to extract multi-scale feature extraction.

#### 3. Feature Fusion helps us to focus more on high-intensity foreground objects and reject the background noise from non-infected skin areas. To perform feature fusion, we first pass CSP block outputs to the Conv2D block to make the feature dimension consistent. Next, we use FC layers with ReLU activation to make the length of each feature the same. We use element-wise multiplication as the feature fusion technique. The fusion operation is $F_{1024} \approx F_1_{1024} \ast F_{2024}$; $F_1$ and $F_2$ are two outputs from FC layers, $\ast$ is element-wise multiplication, and $F'$ denotes the output combined feature.

$$ICS = \sum_{dim=1}^D f_{dim}(I) \Rightarrow$$

$$ICS(I_{\text{sum}}) = \min(\max(I_{\text{sum}}, b_l), u_b)$$ (5)

### 5. Weighted Large-Margin Loss

Instead of binary cross-entropy loss for calculating...
classification similarity error. Figure 1 shows that inter-class similarity exists in our dataset. Many data points lie very close to the decision boundary, and there is a high chance of getting misclassified with a low-margin decision boundary. As shown in Figure 7, margin-based SVM hinge loss can help us to create large margins near the decision boundary. Figure 7 shows the mechanism for the SVM margin where the samples with a probability near 0.5 are at risk of misclassification. So, the SVM Margin creates space around it to afford some error and keep the margin high for overall improved performance. The loss function can be defined as follows in Eq. 6, where $C$ is the number of classes, $N$ is the mini-Batch Size, $P$ is the Penalty parameter, $\Delta$ is the margin parameter, and $y$ is the index ($0 \leq y \leq C - 1$) of the true class label for a sample.

$$M_{Loss_{x,y}} = \sum_{i=0}^{C-1} \max(0, \Delta - x[y] + x[i])^p$$  \hspace{1cm} (6)$$

Here, $x$ is the 2D logits/probability scores vectors derived from the softmax function. So, the dimension of the vector $x$ is $N \times C$ and $i \neq y$. The above Eq. 6 shows the operation for a single example in a mini-batch of size $N$.

$$WM_{Loss_i} = ICS_i \times M_{Loss_i}$$  \hspace{1cm} (7)$$

Next, we multiply the loss for each image w.r.t the corresponding Image Complexity Score $ICS$ for the weighted large-margin loss. The final weighted loss becomes as in Eq. 7, where $i$ is the $i^{th}$ example in the mini-batch.

### 4 Proof Of Concept Setup

#### 4.1 Dataset

In our experiment, we choose the latest publicly open Mpx dataset known as Mpx Skin Lesion Dataset Version 2.0 (MSLD v2.0) [32, 33]. This dataset is the extended version of MSLD 1.0; in the earlier version, there were only two classes: Mpx and Others. In the other class, there were chickenpox and Measles samples. The latest version has six classes: Mpx, Chickenpox, Measles, Cowpox, Hand-foot-mouth disease, and Healthy. Figure 4 illustrates a few samples from each class. Different augmentation techniques, such as rotation, translation, reflection, brightness jitter, noise, etc., were performed on the original dataset to increase the size of the dataset. The class distribution of the original and augmented dataset is presented in Table 1. The dataset folder includes actual and increased folded images (Train, Test, and Validation) with a proportion of 70:10:20. Each of the Train, Test, and Validation folders
was divided into six folders containing six different classes. We kept Mpox as the positive class for our training and grouped all the folders to create the negative class. In this way, we had 14,354 positive Mpox samples and 22,690 negative samples for training. To make the train-val dataset, we first merged training and validation sets. Then, we followed 75:25 random splitting for training and validation set creation. We used the 20% test set given with the MSLD v2.0 dataset for testing.

<table>
<thead>
<tr>
<th>Infection Class</th>
<th># of Original Images</th>
<th># of Augmented Images</th>
<th># of Unique Patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chickenpox</td>
<td>75</td>
<td>3598</td>
<td>62</td>
</tr>
<tr>
<td>Cowpox</td>
<td>66</td>
<td>3220</td>
<td>41</td>
</tr>
<tr>
<td>Healthy</td>
<td>114</td>
<td>5654</td>
<td>104</td>
</tr>
<tr>
<td>HFMD</td>
<td>161</td>
<td>7982</td>
<td>144</td>
</tr>
<tr>
<td>Measles</td>
<td>55</td>
<td>2618</td>
<td>46</td>
</tr>
<tr>
<td>Mpox</td>
<td>284</td>
<td>14070</td>
<td>143</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>755</strong></td>
<td><strong>37044</strong></td>
<td><strong>540</strong></td>
</tr>
</tbody>
</table>

Table 1 Instance distribution statistics of the presented Mpox Skin Lesion Dataset (MSLD) v2.0

### 4.2 Implementation Details

We have used Python with PyTorch as the deep learning framework to implement the project. First, we create a custom data loader to load all samples from a folder. Next, we perform several transformations on the image, such as resizing, normalization, horizontal flip, and color jitter. Next, we use the Darknet53 as the backbone as it is much lighter than dense residual networks and has been shown to preserve semantic information well from low-level features. We use the pytorch `LeakyReLU` activation unit and `Conv2d` for 2D convolution operation throughout the deep layers of the DarkNet53. At the output layer, we use `Softmax` as the activation. For calculating the ICS, we use the PyTorch `sum` operation for summing all neuron activation. Next, we use the torch `Clamp` method to restrict the ICS within [0.7, 1.2]. Finally, for large-margin loss calculation, we use PyTorch `MultiMarginLoss`, where we pass the softmax class output probability logits and the ground truth class id. The weights and biases were updated using `Adam` optimizer with a weight decay of 0.003 as a regularization. We also reduce the learning rate by a factor of 0.01 every seven epochs.

### 4.3 Hyper-parameter Settings

In our proposed network, the load image size was 256 × 256. To train our model, we have resized all images to 128 × 128 pixels and set 64 as the mini-batch size in each epoch. The learning rate for training all models was set to 0.003. We put the 70 : 20 ratio for the training and validation set. Next, the Margin and Penalty in the `MultiMarginLoss` loss function were set to 10.0 and 1, respectively. The system specification for experimental devices is presented in Table 2. We have used the workstation presented in Table 2 for all training and fine-tuning. We run the inferences on both the workstation and embedded Jaston TX2; this gives us the difference between power consumption in high GPU and limited GPU devices.

### 5 Experiments

This work aims to propose a model that can produce optimal performance compared to SOTA, and the model is very lightweight in terms of power consumption, memory consumption, and
<table>
<thead>
<tr>
<th>Spec.</th>
<th>Workstation</th>
<th>Nvidia Jetson TX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel® CoreTM i9-11900K</td>
<td>Dual-Core NVIDIA Denver 2 64-Bit</td>
</tr>
<tr>
<td>CPU Core</td>
<td>3.50GHzx16</td>
<td>Quad-Core ARM® Cortex®-A57 MPCore</td>
</tr>
<tr>
<td>RAM</td>
<td>167GB 128-bit LPDDR4</td>
<td>8GB 128-bit LPDDR4</td>
</tr>
<tr>
<td>GPU</td>
<td>GP102 TITAN Xp</td>
<td>256-core NVIDIA Pascal™ GPU</td>
</tr>
</tbody>
</table>

Table 2 System Specifications.

Fig. 8 Nvidia Jetson TX2 power measurement with P4400 p3 Kill-A-Watt meter.

GFLOPS. Toward this goal, we divide our discussion of experimental results into two parts. In the first part, we focus on classification performance. Here, we use several performance metrics, such as Precision, Recall, and Accuracy, to show the superiority of our model. We also compare our model with several SOTA classification pipelines to prove the consistency. Below in Eq. 8, we define the classification performance measurement metrics.

The calculation of precision ($P$), recall ($R$), and accuracy score ($AC$) is determined through the Eq. 8. True positives ($TP$) denote instances accurately predicted by the model, false positives ($FP$) represent instances wrongly predicted as a positive case by the model, and false negatives ($FN$) indicate instances incorrectly predicted as a negative case by the model and ($TN$) denotes the correctly predicted negative-cases. Precision ($P$) represents the proportion of relevant instances recovered by the model, while recall ($R$) is the fraction of relevant instances correctly identified by the model among all relevant instances. The accuracy score provides an overall measure of correct classification by the model.

$$ P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}, $$

$$ AC = \frac{TP + TN}{TP + TN + FP + FN} \quad (8) $$

On the other hand, we use metrics such as GFLOPS and Num. of Layers, model size, and power consumption (in Watt) for computational expense measurement. Based on this information, we analyze and compare models with our proposed model and discuss its suitability for deployment in embedded resource-constrained devices. Following a previous work [34], we use a P4400 p3 Kill-A-Watt meter (See Figure 8) for calculating the power consumption. We connect the power meter to the CPU line and take the baseline power consumption reading to run the workstation. We carry several readings and then average the numbers for an accurate and stable baseline result. Next, we run the model for inference and see how much power consumption increases due to the model inference via GPU and CPU.

5.1 Classification Performance Comparisons:

The experimental dataset was released recently, so almost no open-source work is available for model comparison. Due to that, we implemented several comparing models following the latest research [16, 17, 33, 35] on this topic. We will use several popular classifiers for performance comparisons, such as ResNet50, MobileNetv2, DenseNet121, and XceptionNet. We use DarkNet53 as the baseline and show the improvements through our BinaryDarkNet53 model.

Figure 9 shows the confusion matrix for the baseline and the proposed model. Here, we can see that the model performs reasonably well on the test dataset, but there are still some FP and FN cases. As we see in 1, there are samples where inter-class similarity and intra-class
diversity exist. To solve the problems, we implemented Image Complexity Scores (ICS) and the Large-Margin Loss function. For ICS calculation in Eq. 5, we use Lower Bound = 0.7 and Upper Bound = 1.2 to represent easy and complex examples, respectively. Using our proposed model, we resolved the challenges and recorded optimal results with only 7 FN cases and 11 FP cases. The overall performance was entirely satisfactory, considering the lightweight characteristics of the model.

In Table 3, we compare different SOTA classification methods with our proposed method. For the Mpox classification task, the Recall metric is crucial. If we fail to detect a positive Mpox case correctly, the patient may get released from the hospital and infect more healthy people who are in contact with them. From Table 3, ResNet50 gives the least recall approx. 86%, XceptionNet and DarkNet53 provide very close recall 92% and 90%, respectively. However, the computational cost for XceptionNet is significantly higher than the DarkNet53 model. We show our proposed model’s performance with and without ICS+LM modules. It is worthy to notice that the Binarized DarkNet53 performs inferior to the original DarkNet53 implementation. The reason behind this is that, due to binary weights and bias, the model misses some precision values after the decimal points, which makes the learning slightly noisy. However, after incorporating the ICS and LM, we could compensate for the noise and achieve optimal results approx. 95.05% of accuracy and 94.40% of recall on the test set. Our closest competitor in classification performances was the DenseNet121 model, with 93.73% of accuracy and 89.94% of recall.

**Table 3** Performance comparisons with several latest classification models. Here, ICS= Image Complexity Score, LM= Large Margin

<table>
<thead>
<tr>
<th>Feature Extractor</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>84.38</td>
<td>86.40</td>
<td>85.38</td>
<td>89.84</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>92.00</td>
<td>89.84</td>
<td>90.91</td>
<td>93.73</td>
</tr>
<tr>
<td>MobileNetv2</td>
<td>87.40</td>
<td>88.80</td>
<td>88.10</td>
<td>91.76</td>
</tr>
<tr>
<td>XceptionNet</td>
<td>88.46</td>
<td>92.00</td>
<td>90.20</td>
<td>91.76</td>
</tr>
<tr>
<td>DarkNet53</td>
<td>86.26</td>
<td>90.40</td>
<td>88.28</td>
<td>91.76</td>
</tr>
<tr>
<td>BinaryDNet53 W/O ICS+LM</td>
<td>89.90</td>
<td>92.05</td>
<td>90.96</td>
<td>90.11</td>
</tr>
<tr>
<td>BinaryDNet53</td>
<td>91.47</td>
<td>94.40</td>
<td>92.91</td>
<td>95.05</td>
</tr>
</tbody>
</table>

**Table 4** Multi-factor computational cost comparison between our proposed BinaryDNet53 and recent SOTA methods. Power consumption is measured in the Watt unit.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Jatson TX2 Workstation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet50</td>
<td>25.6M</td>
<td>86.47</td>
<td>110 ± 5</td>
<td>195 ± 5</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>6.9M</td>
<td>929.61</td>
<td>201 ± 8</td>
<td>291 ± 8</td>
</tr>
<tr>
<td>MobileNetv2</td>
<td>2.34M</td>
<td>52.94</td>
<td>89 ± 5</td>
<td>178 ± 5</td>
</tr>
<tr>
<td>XceptionNet</td>
<td>20.81M</td>
<td>1164.93</td>
<td>236 ± 8</td>
<td>331 ± 8</td>
</tr>
<tr>
<td>DarkNet53</td>
<td>36.4M</td>
<td>149.33</td>
<td>121 ± 7</td>
<td>210 ± 5</td>
</tr>
<tr>
<td>BinaryDNet53</td>
<td>0.37M</td>
<td>7.52</td>
<td>67 ± 2</td>
<td>138 ± 3</td>
</tr>
</tbody>
</table>

5.2 Computational Performance Comparisons:

The most crucial aspect of our work is to provide a model with a shallow power consumption requirement and low GFLOPS. Table 4 shows that our proposed model is 20× lightweight than the baseline model in terms of GFLOPS. All other models with decimal weight and bias require significantly more GFLOPS and learnable parameters than the proposed model. Although performed well, DenseNet201 and XceptionNet are significantly more computationally expensive than other state-of-the-art when evaluated on the benchmark dataset. The XceptionNet and DenseNet201 require 1163.93 and 929.61 GFLOPS and 20.81 and 6.9 Million parameters, respectively, for a batch size = 64. These heavy models are unsuitable for small devices in real-life scenarios, and the detection rate does not meet real-time requirements. On the other hand, MobileNetv2 was found to be very convincing, with 91.76% accuracy and 52.94 GFLOPS.

We calculate the power consumption during the test data inference. To calculate the power consumption, we run inference on the workstation and a small embedded device, Nvidia Jetson TX2 (See Figure 8). The power consumption for our top
three performers, DensNet121, MobilNetv2, and XceptionNet on workstation inference, are 291, 178, and 331 watts, respectively. On the other hand, the embedded devices require 201, 89, and 236 watts, respectively. Our goal was to design a model that will require ≈ 65 watts to use a mobile charger of 65 watts and get continuous throughput without issues. Our BinaryDNet53 can lower the power consumption to 67 watts on the embedded device, which is very close to our goal. Even in the workstation inference, our proposed model shows superiority by consuming the lowest power(≈ 139) compared to all SOTA methods. The memory consumption from the SOTA models varied between 140 to 270 MB. Our proposed model requires only 84 MB of space, which is reasonable for any small device.

6 Conclusion

In this study, we proposed a lightweight Mpox classification model for resource-constrained mobile or embedded devices. Here, we show that using the binary weights and bias can make the model significantly lighter. It is to be noted that the binary weights and bias can reject some precision values after decimal points, which should be carefully addressed for smooth learning. We also see we have challenges such as intra-class dissimilarity and inter-class similarities in the MSLD v2.0 dataset. To resolve the issues, we proposed Image Complexity Scoring (ICS) based on saliency information to focus more on complex examples. Also, we offered a large-margin SVM loss for separating confusing samples near the decision boundary. The ICS scores for each image were used as the weight during the loss calculation. Our Proposed model outperformed all SOTA methods regarding classification and computational performance. Our BinaryDNet53 achieved 94.40% recall and 95.05% accuracy. Also, we were able to reduce the memory and power Consumption to 84 MB and 67 watts, respectively. Finally, our model is ≈ 20× lightweight in terms of GFLOPS and ≈ 2× power efficient than the baseline model.

Declarations

Ethical Approval

Not applicable.

Availability of supporting data

Data used in this article were obtained from the following Kaggle (dataset link).

Funding

The authors did not receive support from any organization for the submitted work.

References


