Monkey Pox Virus Classification Using YOLOv8

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the

requirement for the award of the

Degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

BY

BOYINA SRI MANI SHANKAR – 21BCE7910

SRIKAL KAKULA – 21BCE7457

SHAIK FARUKH – 21BCE8206

K NIKHIL KUMAR – 21BCE7801

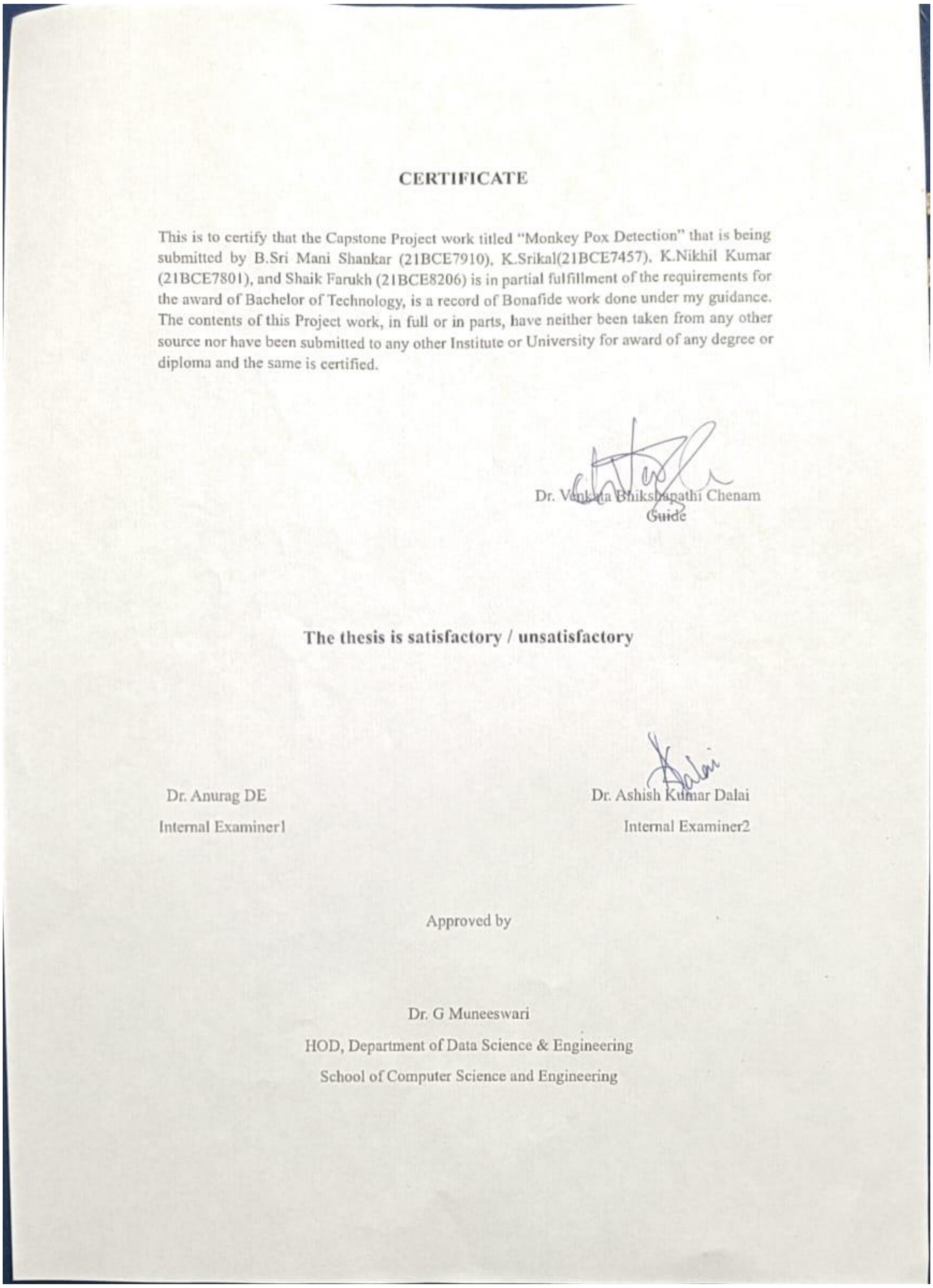
Under the Guidance of

Dr. VENKATA BHIKSHAPATHI CHENAM

Assistant Professor Senior Grade-1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI – 522237
ANDHRA PRADESH, INDIA

November 2024



CERTIFICATE

This is to certify that the Capstone Project work titled "Monkey Pox Detection" that is being submitted by B.Sri Mani Shankar (21BCE7910), K.Srikal(21BCE7457), K.Nikhil Kumar (21BCE7801), and Shaik Farukh (21BCE8206) is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of Bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

[illegible]

Dr. Venkata Bhikshapathi Chenam
Guide

The thesis is satisfactory / unsatisfactory

[illegible]

Dr. Anurag DE	Dr. Ashish Kumar Dalai
Internal Examiner1	Internal Examiner2

Approved by

Dr. G Muneeswari
HOD, Department of Data Science & Engineering
School of Computer Science and Engineering

ACKNOWLEDGEMENTS

We thank VIT-AP University for allowing us to work on this project. We would like to express our deep and sincere gratitude to Dr. Venkata Bhikshaphathi Chenam Sir, School of Computer Science and Engineering for his guidance, providing us with relevant suitable references, valuable support right from the beginning of the project, motivation, and invaluable inputs throughout the process.

We express our heartfelt gratitude to Dr. Saroj Kumar Panigrahy sir, Program Chair and Professor, School of Computer Science and Engineering. It is our pleasure to express our heartfelt gratitude to Dr. S. Sudhakar Ilango Sir, Dean, School of Computer Science and Engineering, for providing an environment to work in for their support and guidance during the course's tenure.

It is a pleasure to thank all faculty members who work as limbs of our university for their non-self-centered enthusiasm and timely encouragement showered on us with zeal, prompting the acquisition of the necessary knowledge to complete our course study. We would like to express our gratitude to our respective parents for their assistance.

Last but not least, we would like to express our gratitude and appreciation to everyone who has contributed directly or indirectly to the successful completion of this project.

ABSTRACT

In recent years, the emergence of infectious diseases has posed significant challenges to public health systems worldwide. Among these diseases, Monkeypox, a zoonotic viral infection, has garnered attention due to its potential for human-to-human transmission and its resemblance to other dermatological conditions such as chickenpox and measles. Accurate and timely detection of Monkeypox is crucial for effective disease management and prevention of outbreaks.

In this study, we present a novel approach for Monkeypox virus detection using deep learning algorithms. A dataset comprising images of Monkeypox, chickenpox, measles, and normal skin conditions was collected and curated for training and evaluation. Leveraging state-of-the-art deep learning techniques, including convolutional neural networks (CNNs), we developed and compared multiple models for automated classification of skin lesions into the a forementioned classes.The proposed models were trained on the collected dataset, employing data augmentation techniques to enhance generalization and robustness. Performance evaluation was conducted using various metrics to assess the model's efficiency in discriminating Monkeypox from similar dermatological conditions.

Our results demonstrate the feasibility and effectiveness of utilizing deep learning algorithms for Monkeypox detection, achieving promising classification accuracy on the test set. Furthermore, we discuss the implications of our findings in the context of early disease detection, surveillance, and public health interventions.

This research contributes to the ongoing efforts in leveraging artificial intelligence for infectious disease detection and underscores the potential of deep learning in addressing emerging public health challenges.

TABLE OF CONTENTS

S.NO.	Chapter	Title	Page No.
1		Acknowledgement	3
2		Abstract	4
3		List of Figures and Tables	6
4	1	Introduction	7
	1.1	Objectives	8
	1.2	Problem Statement	8
	1.3	Scope and Constraints	9
	1.4	Review and Literature	10
5	2	Research Methodology	12
	2.1	Model Building	14
	2.2	Software details	16
	2.3	Procedure	16
6	3.1	Results	18
	3.2	Discussions	22
7	4.1	Conclusion	23
	4.2	Future works	24
8	5	Appendix	25
9		Reference	40

LIST OF FIGURES

Figure No.	Figure Name	Page No.
a	YOLOv8 Network Architecture	12
b	ResNet-18 Architecture	13
c	Process Flow	14
d	YOLOv8 Network Architecture	15
e	Results	19,20,21,22

LIST OF ABBREVATIONS

CNN	Convolutional Neural Network
YOLO	You Only Look Once

CHAPTER 1

Introduction

Mpox is a zoonotic viral infection, first identified in laboratory monkeys in Denmark in 1958, then subsequently described in human populations in the Democratic Republic of the Congo in 1970. Recently, epidemiological events covered 75 countries with more than 3000 confirmed cases by June 2022, raising significant concerns about its global health implications. The World Health Organization, or WHO, classified Mpox as a global health emergency because of the rapid spread of the disease and the problem it presents in the identification process. The mortality is relatively low: 1–10% but nonspecific signs of the disease characterizing it and that caused it to resemble other similar conditions—smallpox, scarlet fever, or roseola—complicated its quick and precise identification.

The conventional diagnostic methods, which include PCR testing, are not widely available in most settings, especially those with limited resources, which leads to delayed detection and treatment. Prompt identification of Mpox cases is important in preventing widespread spread, but the existing diagnostic tools fail to meet the demand for speed and accuracy during an outbreak.

There have been significant promises to the above challenges lately from AI and DL technological development. CNNs represent one of the DL's families, which have gained practical applications in medical images and disease classification. They enable automated, precise, and efficient diagnostic functionalities in order to overcome the limits related to traditional approaches.

1.1 Objective of the project

The aim of this study is to find a strong and efficient system for an early diagnosis through deep learning techniques of the disease called Mpox. The new system proposed utilizes CNN in scanning medical images and hence efficiently classifying Mpox, while differentiating it from other diseases with closely related images, including smallpox and measles. The project aim is to address the issues related to the ambiguous presentation of Mpox, besides the limited availability of PCR testing, to ensure swift and accurate diagnosis that can lead to prompt medical interventions. The system will have the ability to enhance its diagnostic accuracy, reduce the computing requirements, and enhance disease classification reliability by the implementation of transfer learning methodologies.

Additionally, the design has ensured scalability as well as economical feasibility in supporting diagnoses, particularly where health systems are operating under scarce-resource delivery conditions. This, in turn, is projected to increase public health action through earlier detection of a case, control of spreading in communities, and hence better results for management processes. It is also expected to bring highly sophisticated deep-learning methodologies in association with medical imagery that, not only increase accuracy, but also bring forth the proactive approach toward handling infectious emergent diseases, like Mpox.

1.2 Problem Statement

The main intention of this research is to engineer a reliable, efficient diagnosis system using deep learning principles to diagnose Mpox expeditiously. This system shall employ the use of CNNs on medical imaging diagnostics for a proper determination of Mpox and its distinctness from those diseases in terms of images, with other diseases looking similar that are smallpox and measles. The aim of the project is to assist in overcoming the challenges that define the vague symptoms of Mpox and limited access to PCR testing so that it can be diagnosed quickly and accurately. Mpox is a zoonotic disease with a big public health risk, especially where it is endemic or recently emerged. The modern detection methods of the lesion caused by monkeypox viruses rely on visual identification techniques performed by trained professionals, which are slow and nonobjective, thus giving rise to false positives. These virus outbreaks must also be attended to in due course as the presence of viruses leads to a need for novel means of detection with fastidious efficiency.

Although traditional methods in computer vision are highly successful for specific conditions, they cannot identify lesions from the monkeypox virus because the lesions are too small in size, with inconstant morphology, and are further obscured by adjacent tissues. Additionally, the variety of medical images and the urgency of processing them in the clinical setting is a massive challenge to the existing methods of detection.

In the context of these challenges, a powerful and independent system to detect the monkeypox virus was always needed that would find viral lesions in medical images with high accuracy and efficiency. This paper meets that need by applying deep learning techniques, namely YOLOv8, which is an object detection algorithm, in the development of a comprehensive and effective approach for the detection of the monkeypox virus. The overall objective of this research is to apply deep learning techniques to increase the speed, accuracy, and scalability of virus detection in order to help improve our ability to detect and respond quickly to monkeypox virus outbreaks.

This intends to allow for timely medical response. Transfer learning methodologies are applied to this system to enhance the precision of diagnosis, reduce the computational requirement, and increase the reliability of disease classification.

Furthermore, the system is engineered to provide a diagnostic solution that is scalable and economically viable, particularly in areas where health resources are limited. It will help strengthen public health by enabling rapid case detection, reducing transmission within communities, and improving the results from disease management. The combination of advanced deep learning methods with medical imaging will not only enhance the accuracy of diagnosis but also assist in developing a proactive approach for dealing with emerging infectious diseases like Mpox..

1.3 Scope and Constraints

This project aims to create a cost-effective and efficient tool for classifying monkeypox and similar diseases like measles and chickenpox. By leveraging deep learning, the system seeks to streamline early detection, reduce dependence on physical tests, and assist healthcare professionals with a quick pre-diagnosis tool. It focuses on real-time processing, user-friendliness, and scalability to accommodate additional diseases and enhance diagnostic accuracy in the future.

Constraints

- Requires high-quality, balanced datasets for reliable training.
- Model performance may vary with poor image quality or diverse real-world conditions.
- Privacy and bias in patient data must be carefully managed.
- Limited computational resources and hardware compatibility can restrict usage in remote areas.
- Initial deployment focuses on a limited set of diseases, with future expansion requiring additional validation.

1.4 Literature Survey

Introduction

Monkeypox, first identified in humans in 1970, has become a growing concern in recent years due to its increasing global spread. This has driven significant research into early detection and diagnosis, leveraging advancements in artificial intelligence (AI) and computer vision. The application of deep learning (DL) models and machine learning (ML) techniques has emerged as a key focus area for addressing the challenges of diagnosing monkeypox and differentiating it from other diseases such as measles, chickenpox, and normal skin conditions.

Early Applications of Deep Learning in Monkeypox Diagnosis

The use of deep learning for monkeypox diagnosis is relatively recent. One study utilized a modified VGG16 model on a dataset called Monkeypox2022, achieving remarkable performance metrics, including 97% accuracy. Similarly, another study employed GoogleNet with optimization techniques, achieving a peak accuracy of 98.8%, though their sensitivity and recall rates varied.

Other studies focused on mobile and accessible solutions. For instance, a mobile application developed using Java and Android achieved 91.11% accuracy, offering potential for practical deployment. Meanwhile, research exploring multi-class classification across diseases like measles, mumps, and chickenpox achieved accuracy rates of up to 83%.

Comparative Analysis of Deep Learning Models

Several researchers have performed comparative analyses of DL architectures for monkeypox detection. Experiments with VGG16, ResNet50, and InceptionV3 revealed varying performances among the models. Another study highlighted MobileNetV3 as the most efficient model, achieving 96% accuracy. Further comparative studies demonstrated the variability in performance across DL models, with best accuracy scores of 87%.

Explainability has also been a key focus. Some researchers proposed using techniques like LIME and Grad-CAM to improve the transparency of DenseNet-based models, achieving 97% accuracy. Others extended this with a comparative analysis of multiple explainability-focused models.

Role of Data Augmentation and Synthetic Data Generation

A major limitation in monkeypox research is the scarcity of annotated datasets. Many studies have adopted data augmentation techniques to address this. For instance, GAN-based approaches have been widely used to generate synthetic images, improving model robustness. Other studies implemented StyleGAN to balance datasets for skin lesion analysis, an approach that could be adapted for monkeypox classification. However, challenges remain, as GAN models often require significant computational resources and may impose high implementation costs.

Novel Approaches and Model Enhancements

Recent research has introduced innovative methods to enhance model performance. One study proposed a blockchain-integrated detection framework for secure data handling, achieving 98.8% accuracy. Meanwhile, another utilized a blood-test-based dataset with feature selection techniques, achieving 98.48% accuracy.

Research focusing on optimizing model architectures and using transfer learning (TL) techniques has also been prominent. For example, one study highlighted MobileNet as the best performer, achieving near-perfect accuracy rates.

Multi-Class Classification and Hybrid Approaches

Multi-class classification has been a recurring theme in recent research. Hybrid frameworks combining traditional CNN architectures with ensemble methods have demonstrated improved accuracy. For instance, one study using five-fold evaluation achieved 93% accuracy, while other research integrated CNN features with ML classifiers like KNN and Random Forest, achieving a maximum prediction rate of 91.1%.

DenseNet-121 has also been highlighted for its strong performance. Studies have revealed its superiority over traditional CNNs, achieving 93% accuracy in monkeypox classification. Similarly, experiments with =YOLOv8 models showed high efficacy in identifying skin lesion types, achieving 96% accuracy.

Challenges and Future Directions

Despite the success of DL models, several challenges persist. The limited availability of medical datasets, privacy concerns, and computational constraints hinder widespread adoption. Techniques such as synthetic data generation, federated learning, and privacy-preserving methods like blockchain can address these concerns. Moreover, the integration of hybrid models with explainability tools could further enhance trust and usability in clinical settings.

Conclusion

The literature on monkeypox detection highlights significant advancements in AI-driven approaches. From leveraging pre-trained DL models to developing innovative hybrid frameworks, researchers have demonstrated the potential of these methods to improve diagnostic accuracy. However, overcoming challenges related to dataset availability and computational requirements will be essential to ensure the scalability and practicality of these solutions. Future work should focus on enhancing model generalization, ensuring data security, and exploring novel architectures tailored for medical image analysis.

CHAPTER 2

Research Methodology

The study involves developing a robust deep learning-based approach to classify monkeypox and other related diseases, such as measles, chickenpox, and normal skin conditions. A diverse set of deep learning models, including Resnet 18 and YOLOv8, were implemented to identify the most effective model for accurate classification. Preprocessing techniques such as data augmentation, grayscale conversion, and resizing were applied to enhance the dataset and improve model generalization.

A web-based application was developed to integrate the trained model, enabling user-friendly and real-time detection. The system was rigorously tested using cross-validation and statistical analyses, ensuring reliability and scalability for practical applications.

PROJECT FLOW / FRAMEWORK OF THE PROPOSED SYSTEM

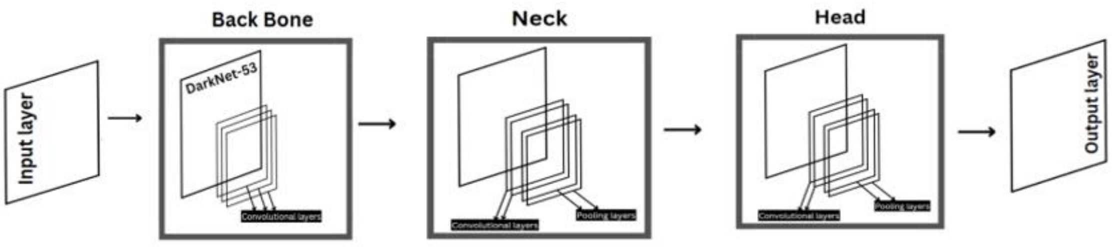


Fig. a. YOLOv8 Network Architecture

YOLOv8, an advanced version of the popular YOLO (You Only Look Once) model, excels in real-time image classification tasks due to its optimized architecture and efficient performance. The model uses a powerful backbone network, such as CSPDarknet53, to extract key features from input images. YOLOv8's detection head processes these features, allowing it to classify images with high precision. Unlike traditional object detection models, YOLOv8 uses anchor-free mechanisms, enabling it to handle objects of various shapes and sizes more effectively. Additionally, YOLOv8 incorporates multi-scale predictions and focal loss to improve accuracy in challenging conditions, where objects may appear in different scales or aspect ratios.

During training, YOLOv8 requires a large, annotated dataset with diverse images to learn robust feature representations. Preprocessing steps like resizing, normalization, and augmentation (e.g., random cropping and flipping) are crucial to ensure the model generalizes well across unseen data. The model is optimized using stochastic gradient descent (SGD) with momentum, and hyperparameters such as learning rate, batch size, and dropout rate are fine-tuned to maximize accuracy and prevent overfitting. With its capability to process images quickly and accurately, YOLOv8 is widely applicable in areas such as medical imaging, surveillance, autonomous vehicles, and industrial automation, where real-time classification is essential for decision-making.

ResNet-18 for Image Classification

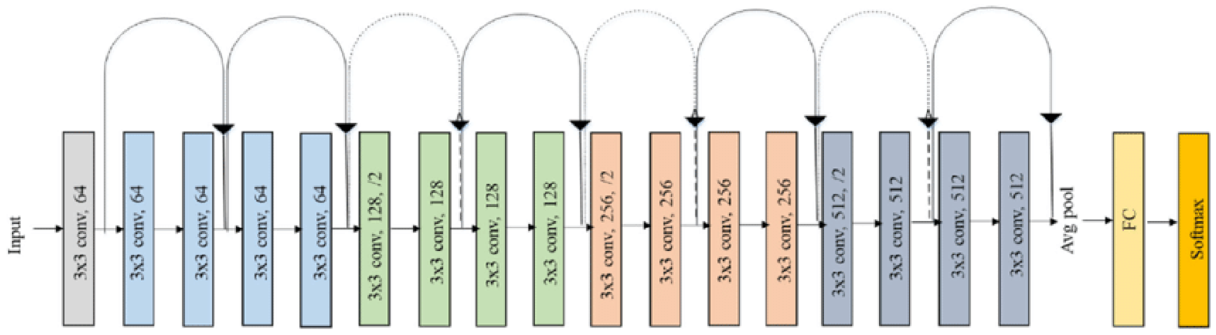


Fig. b. ResNet-18 Architecture

ResNet-18, a variant of the ResNet (Residual Networks) family, is designed to tackle the challenge of training very deep neural networks by using residual connections. These connections, which skip one or more layers, help mitigate the vanishing gradient problem, enabling the model to learn more effectively even with deeper architectures. ResNet-18 consists of 18 layers, including convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for final classification.

The key feature of ResNet-18 is its residual blocks, which allow for smoother gradient flow during backpropagation. This architecture enables the model to learn complex patterns without suffering from the degradation problem that affects deeper networks. ResNet-18 is particularly useful in image classification tasks where computational efficiency is important, as it strikes a balance between depth and performance. During training, the model is optimized using loss functions like cross-entropy, and hyperparameters are adjusted to achieve the best results.

ResNet-18 is widely used in applications such as medical image analysis, real-time video processing, and autonomous systems, offering strong performance without the computational demands of larger networks. Its ability to handle complex image classification tasks with relatively low computational cost makes it a popular choice for many real-world applications.

2.1 Model building

The model-building process for this project followed a structured workflow, beginning with research and model selection, progressing through data preparation, and training, and concluding with model evaluation and deployment. Here is an overview:

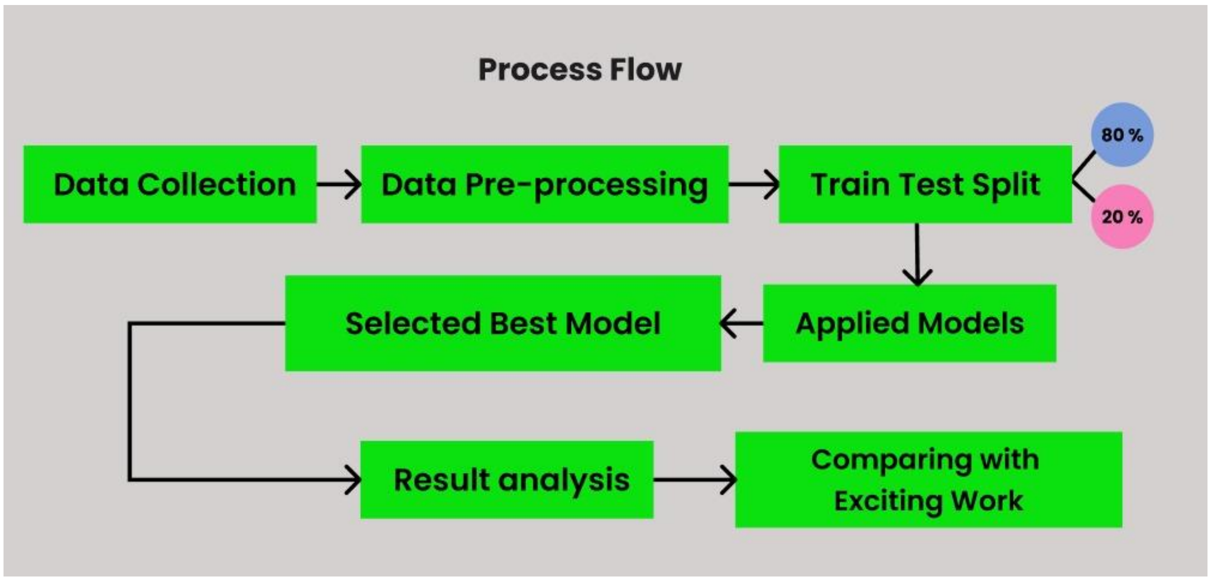


Fig. c. Process Flow

1. Research and Model Selection

- A comprehensive review of existing research papers and studies was conducted to identify the best model for the task.
- YOLOv8 was selected as the primary model for its ability to perform real-time image classification with high accuracy and low inference time.
- Other architectures like ResNet18, DenseNet121, and AlexNet were also implemented for comparison to determine the most efficient model for multi-class classification.

2. Data Collection and Preparation

Dataset Composition: The dataset consisted of over 700 images, divided into four classes: Monkeypox, Chickenpox, Measles, and normal skin conditions.

Data Splitting:

- For YOLOv8, a **70-10-10 split** (70% training, 10% validation, and 10% testing) was used to ensure a balanced evaluation.

- For other models like ResNet18, an **80-20 split** was applied for training and testing.

Preprocessing:

- Images were resized and converted to grayscale to ensure uniformity across the dataset.
- Data augmentation techniques such as flipping, rotation, and cropping were applied to increase variability and enhance generalization.

3. Training the Models

YOLOv8 Training:

- YOLOv8 utilized its anchor-free detection mechanism, optimized for multi-scale objects and diverse feature representations.
- Training included stochastic gradient descent (SGD) for optimization.

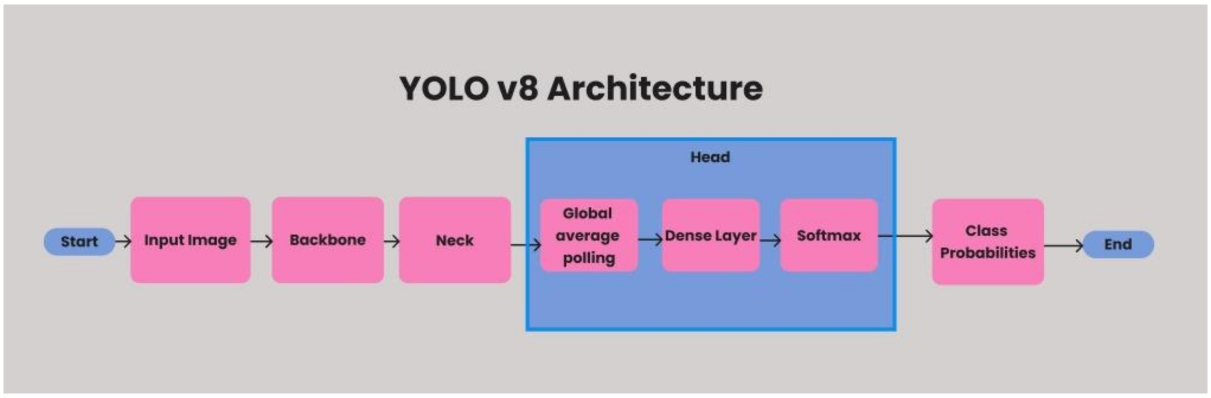


Fig. d. YOLOv8 Network Architecture

Training Other Models:

- Models like ResNet18 and DenseNet121 were trained using cross-entropy loss and standard backpropagation.
- Hyperparameters such as learning rate, batch size, and dropout rate were carefully tuned to achieve optimal performance.

4. Model Evaluation and Comparison

Metrics Used:

- Accuracy, overfitting underfitting and confusion matrix test sample verification were employed to evaluate performance.

- YOLOv8 demonstrated superior real-time classification capabilities with minimal latency, making it the ideal choice.
- ResNet18 also performed well, offering a balance of speed and accuracy, particularly for non-real-time use cases.

5. Web Application Integration

- A web-based interface was developed using Streamlit to make the model accessible and user-friendly.
- Users can upload images through the interface, and the integrated YOLOv8 model provides real-time predictions.
- This application ensures the practical usability of the project, especially for remote or underserved areas requiring efficient diagnostic tools.

2.2 Software details

Tools Used:

- Deep Learning Frameworks: PyTorch/TensorFlow (for training models).
- Web Development: Streamlit for the interactive web interface.
- Development Environment: VS Code, Google Colab

2.3 Procedure

Detecting Monkeypox using deep learning models such as YOLOv8 and ResNet18 involves a structured sequence of steps, beginning with data preparation and culminating in model deployment. Below is the step-by-step procedure tailored for Monkeypox detection:

Data Collection

- The dataset consists of over 700+ images, encompassing four classes: Monkeypox, Chickenpox, Measles, and normal skin conditions.
- Data was gathered from public sources, ensuring diversity in skin types, lighting conditions, and image resolutions.
- The dataset underwent quality checks to ensure completeness, relevance, and diversity.

Data Preprocessing

- **Normalization:** All images were resized to a consistent dimension and converted to grayscale for uniformity.

- **Augmentation:** Techniques such as flipping, rotation, cropping, and brightness adjustment were applied to enhance variability and improve model generalization.
- **Splitting:** For YOLOv8, the data was divided into 70% training, 10% validation, and 10% testing. Other models like ResNet18 used an 80-20 split for training and testing.

Feature Selection

- Feature selection was intrinsic to the deep learning model architectures used.
- YOLOv8 automatically extracted multi-scale features from images using its feature pyramid network.
- ResNet18 leveraged its hierarchical layers to identify spatial and semantic features relevant to disease classification.

Model Training

- **YOLOv8 Training:**
    - The model was optimized using stochastic gradient descent (SGD) with a suitable learning rate scheduler.
    - Loss functions such as classification loss and localization loss ensured accurate detection and classification.
- **ResNet18 Training:**
    - Cross-entropy loss and Adam optimizer were used to fine-tune the network for multi-class classification.
    - Hyperparameter tuning included adjustments to learning rate, batch size, and epochs.

Model Evaluation

- Evaluation metrics such as Accuracy, overfitting underfitting and confusion matrix test sample verification were employed to evaluate performance.,
- YOLOv8 excelled in real-time detection with high accuracy and low latency, while ResNet18 performed robustly in general classification tasks.
- Confusion matrices were analysed to assess classification performance across all four categories.

Cross-Validation

- Cross-validation ensured the robustness of the models, with validation offering insights into performance consistency across data splits.
- This step helped refine model hyperparameters and mitigate overfitting.

Comparison with Other Models

- Models like AlexNet, DenseNet121, and SqueezeNet were evaluated alongside YOLOv8 and ResNet18.

- YOLOv8 demonstrated superior real-time performance, while ResNet18 balanced accuracy with computational efficiency.

Web Application Deployment

- A Streamlit-based interface was developed for user-friendly interaction.
- Users could upload images, and the trained YOLOv8 model provided instant feedback with class labels.

This comprehensive approach ensured that the developed solution was accurate, efficient, and deployable in practical scenarios for Monkeypox detection and classification.

CHAPTER 3

RESULTS

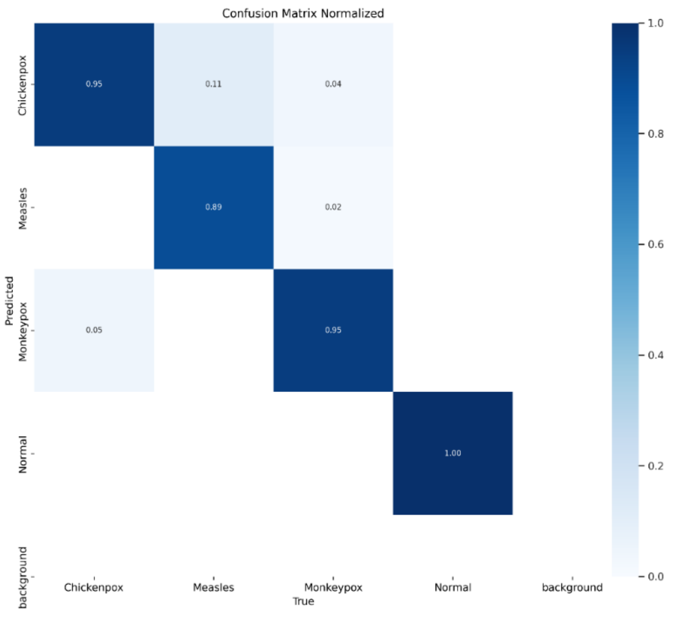
Dataset: The model was trained on a dataset containing 700+ images across four classes: Monkeypox, Chickenpox, Measles, and normal.
Performance:

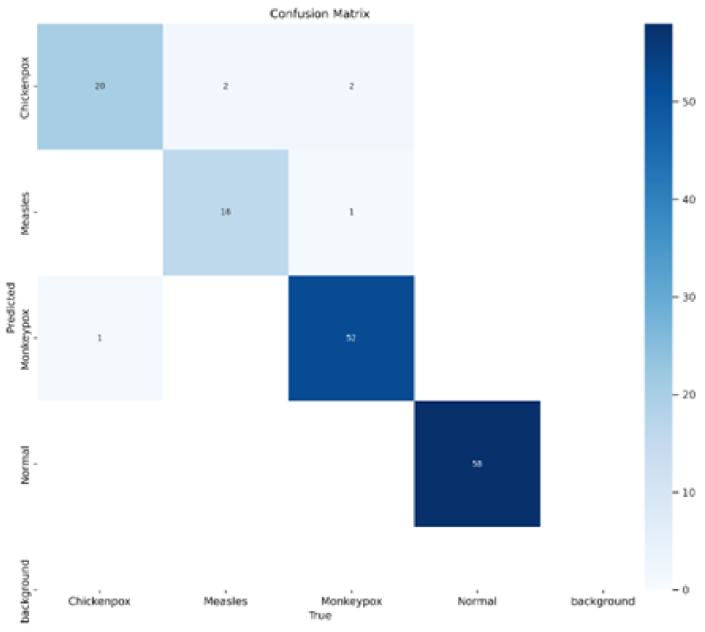
- YOLOv8 delivered the best classification accuracy among the tested models.
- Results showed reliable differentiation among the four classes, indicating the robustness of the chosen algorithms.

Real-Time Testing: The Streamlit-based interface enables efficient testing by providing instant feedback on the uploaded image, enhancing usability in remote and underserved regions.

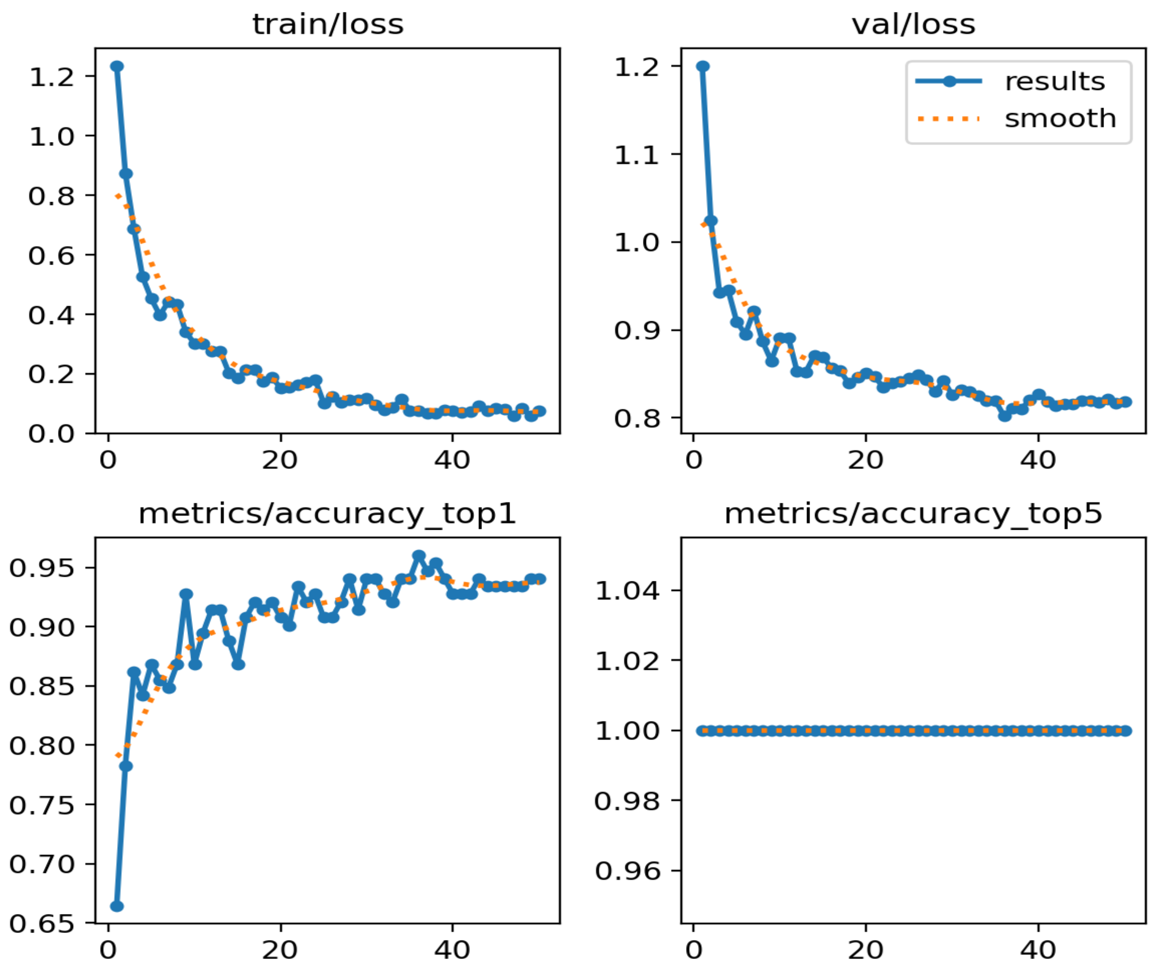
Training batch sample:

Confusion matrix:

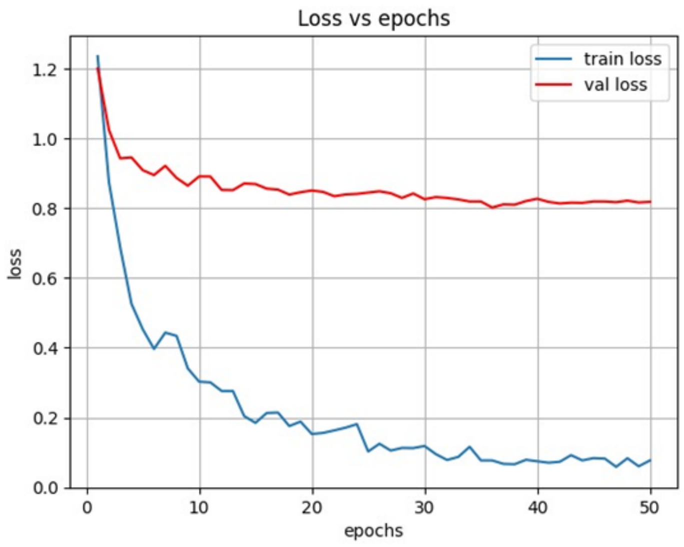


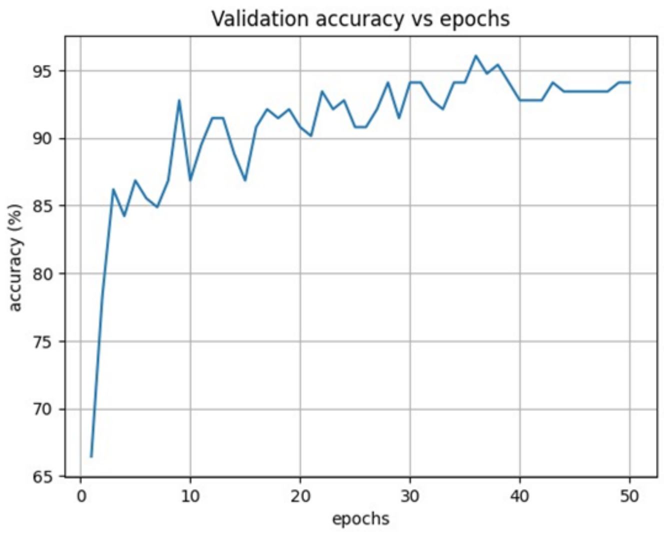


Yolov8 performs:

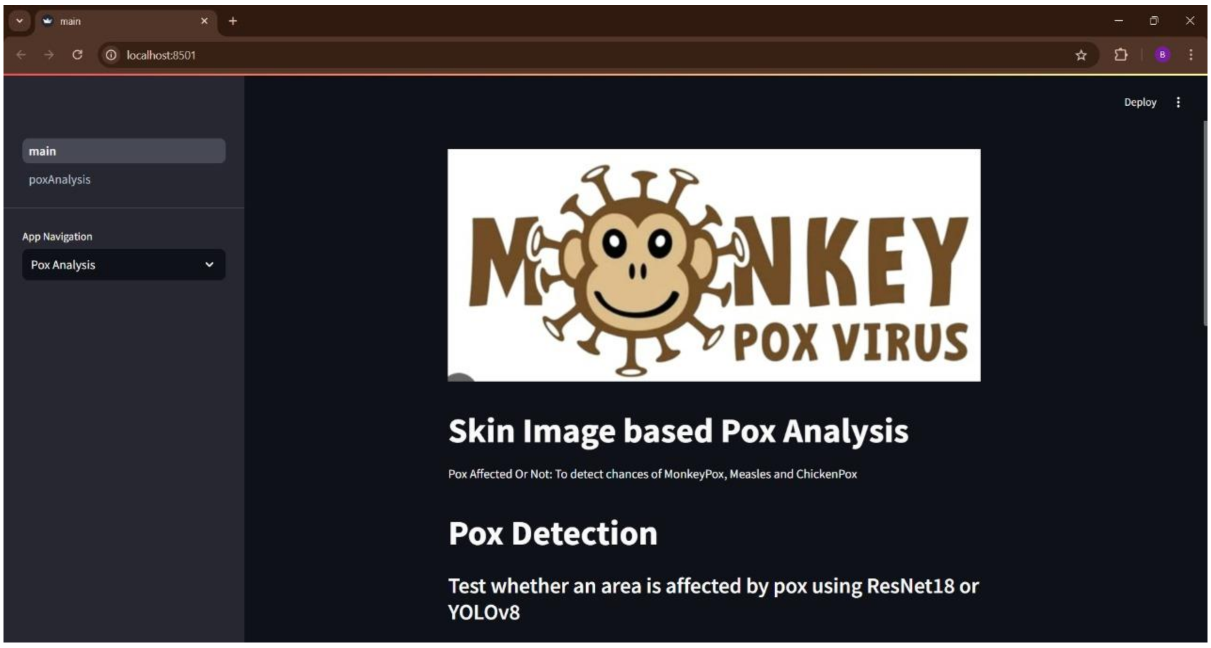


Validation Accuracy and train loss :





Web App:

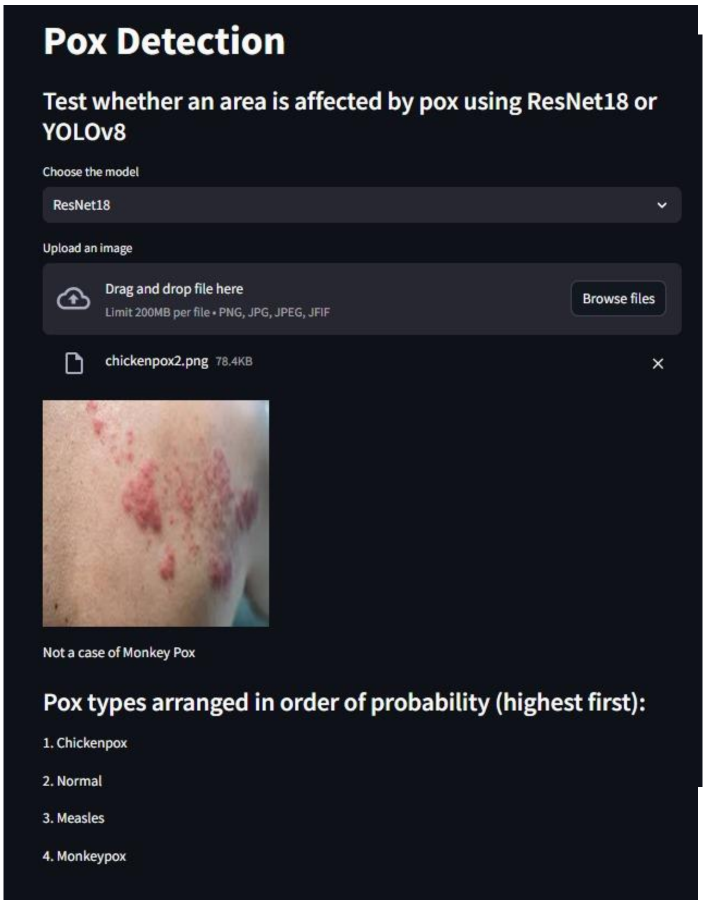


Skin Image based Pox Analysis

Pox Affected Or Not: To detect chances of MonkeyPox, Measles and ChickenPox

Pox Detection

Test whether an area is affected by pox using ResNet18 or YOLOv8



Pox Detection

Test whether an area is affected by pox using ResNet18 or YOLOv8

Choose the model

ResNet18

Upload an image

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG, JFIF

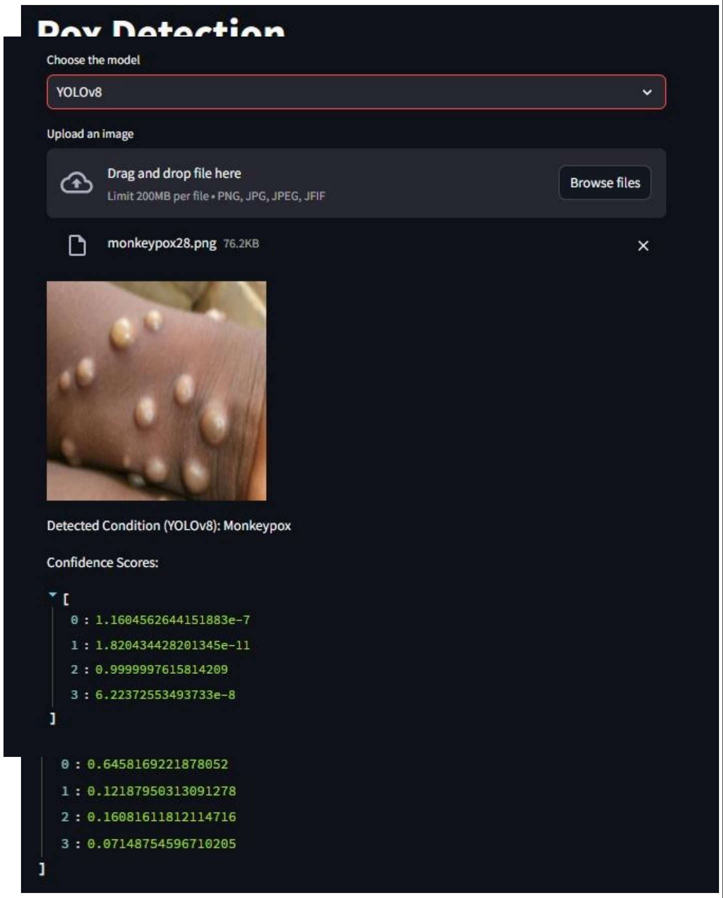
Browse files

chickenpox2.png 78.4KB

Not a case of Monkey Pox

Pox types arranged in order of probability (highest first):

1. Chickenpox
2. Normal
3. Measles
4. Monkeypox



Pox Detection

Choose the model

YOLOv8

Upload an image

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG, JFIF

Browse files

monkeypox28.png 76.2KB

Detected Condition (YOLOv8): Monkeypox

Confidence Scores:

```
[
  0 : 1.1604562644151883e-7
  1 : 1.820434428201345e-11
  2 : 0.9999997615814209
  3 : 6.22372553493733e-8
]

  0 : 0.6458169221878052
  1 : 0.12187950313091278
  2 : 0.16081611812114716
  3 : 0.07148754596710205
]
```

DISCUSSION

- The results validate the use of YOLOv8 as robust models for Monkeypox detection.
- The deployment of YOLOv8 for real-time application workflows caters to diverse use cases.
- Despite the high accuracy, future work can focus on increasing the dataset size and diversity to further enhance model generalization.
- Exploring ensemble techniques could combine the strengths of multiple models to improve overall performance.
- The integration of the web application bridges the gap between research and practical usability, making the solution accessible to a wider audience.

CHAPTER 4

Conclusion and Future works

4.1 Conclusion

This project successfully applied advanced deep learning techniques to classify Monkeypox, Chickenpox, Measles, and normal skin conditions. The following key takeaways emerged:

1. **Model Performance:**
    - YOLOv8 demonstrated superior performance for real-time image classification, achieving an accuracy of 96%, making it ideal for applications requiring immediate feedback.
    - ResNet18 proved effective for batch processing tasks, offering reliable classification accuracy in a computationally efficient manner.
2. **Web Application Deployment:**
    - The integration of a user-friendly Streamlit interface provided seamless interaction for users, allowing real-time classification through image uploads, bridging the gap between technical innovations and practical usage.
3. **Dataset Robustness:**
    - A well-curated dataset of over 700+ images enabled the models to generalize effectively, showcasing their ability to distinguish between different skin conditions accurately.

4.2 Future Works

While the project achieved its primary objectives, there is room for enhancement and expansion in the following areas:

1. **Enhanced Dataset:**

    - **Expansion:** Incorporate a larger dataset with images from diverse sources and demographics to improve model robustness.

- **Real-World Images:** Collect images directly from healthcare settings to better reflect practical use cases.

2. **Model Improvement:**

   - Experiment with advanced ensemble techniques, combining YOLOv8 and ResNet18 to leverage their strengths.
   - Explore lightweight versions of the models for efficient deployment on edge devices like mobile phones or embedded systems.

3. **Clinical Validation:**

   - Test the models in real-world healthcare settings in collaboration with medical professionals to validate their reliability and effectiveness.
   - Obtain necessary regulatory certifications to enable deployment in diagnostic workflows.

4. **Deployment and Accessibility:**

   - Scale the web application to mobile and offline platforms, enhancing accessibility in remote regions.
   - Add support for multiple languages and accessibility features to cater to a wider audience.

5. **Integration with Healthcare Systems:**

   - Develop APIs and plugins for integration into existing electronic health records (EHR) and telemedicine platforms to streamline healthcare delivery.

Chapter 5

Appendix

# CODE FOR CNN

```
from keras.models import Sequential
from keras.layers import Dense,Activation,Flatten,Dropout
from keras.layers import Conv2D,MaxPooling2D
from keras.callbacks import ModelCheckpoint

model1=Sequential()

model1.add(Conv2D(128,(3,3),input_shape=data.shape[1:]))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
#The first CNN layer followed by Relu and MaxPooling layers

model1.add(Conv2D(64,(3,3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
#The second convolution layer followed by Relu and MaxPooling layers

model1.add(Conv2D(32,(3,3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))
#The thrid convolution layer followed by Relu and MaxPooling layers

model1.add(Flatten())
#Flatten layer to stack the output convolutions from 3rd convolution
layer
model1.add(Dropout(0.2))

model1.add(Dense(128,activation='relu'))
#Dense layer of 128 neurons

model1.add(Dropout(0.1))
model1.add(Dense(64,activation='relu'))
#Dense layer of 64 neurons

model1.add(Dense(4,activation='softmax'))
#The Final layer with two outputs for two categories
```

```
model1.compile(loss='categorical_crossentropy',optimizer='adam',metrics
=['accuracy'])
```

# CODE FOR RESNET 50

```
import cv2
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
import keras
from keras.models import Sequential, Model,load_model
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping,ModelCheckpoint
from google.colab.patches import cv2_imshow
from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D,
BatchNormalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D,
GlobalMaxPooling2D,MaxPool2D
from keras.preprocessing import image
from keras.initializers import glorot_uniform
train_datagen =
ImageDataGenerator(zoom_range=0.15,width_shift_range=0.2,height_shift_r
ange=0.2,shear_range=0.15,)
test_datagen = ImageDataGenerator()
train_generator =
train_datagen.flow_from_directory("/content/dataset_train/train",target
_size=(224, 224),batch_size=32,shuffle=True)
test_generator =
test_datagen.flow_from_directory("/content/dataset_train/val",target_si
ze=(224,224),batch_size=32,shuffle=False)
def identity_block(X, f, filters, stage, block):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    F1, F2, F3 = filters

    X_shortcut = X

    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(1, 1),
padding='valid', name=conv_name_base + '2a',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)
    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1),
padding='same', name=conv_name_base + '2b',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1),
padding='valid', name=conv_name_base + '2c',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    X = Add()([X, X_shortcut])# SKIP Connection
    X = Activation('relu')(X)

    return X
def convolutional_block(X, f, filters, stage, block, s=2):

    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2, F3 = filters

    X_shortcut = X

    X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(s, s),
padding='valid', name=conv_name_base + '2a',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1),
padding='same', name=conv_name_base + '2b',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1),
padding='valid', name=conv_name_base + '2c',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)

    X_shortcut = Conv2D(filters=F3, kernel_size=(1, 1), strides=(s, s),
padding='valid', name=conv_name_base + '1',
kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
    X_shortcut = BatchNormalization(axis=3, name=bn_name_base +
'1')(X_shortcut)

    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X
def ResNet50(input_shape=(224, 224, 3)):

    X_input = Input(input_shape)

    X = ZeroPadding2D((3, 3))(X_input)

    X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1',
kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2,
block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')


    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3,
block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4,
block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    X = X = convolutional_block(X, f=3, filters=[512, 512, 2048],
stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)

    model = Model(inputs=X_input, outputs=X, name='ResNet50')

    return model
base_model = ResNet50(input_shape=(224, 224, 3))
headModel = base_model.output
headModel = Flatten()(headModel)
```

```
headModel=Dense(256, activation='relu',
name='fc1',kernel_initializer=glorot_uniform(seed=0))(headModel)
headModel=Dense(128, activation='relu',
name='fc2',kernel_initializer=glorot_uniform(seed=0))(headModel)
headModel = Dense(4, activation='softmax', name='fc3',
kernel_initializer=glorot_uniform(seed=0))(headModel)
opt = SGD(learning_rate=1e-3, momentum=0.9)
model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])

model = Model(inputs=base_model.input, outputs=headModel)
H =
model.fit(train_generator,validation_data=test_generator,epochs=100,ver
bose=1,callbacks=[mc,es])
```

# MODEL BUILDING

```
import torch
import torchvision
from torchvision import datasets, models, transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader, random_split

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn

import numpy as np
import matplotlib.pyplot as plt
import time
import os
from PIL import Image
import copy
root = '/content/drive/MyDrive/project/data/'

data_transform =
transforms.Compose([transforms.Grayscale(num_output_channels=1),
                                     transforms.Resize((64,64)),
                                     transforms.ToTensor()])
dataset = ImageFolder(root, transform=data_transform)

print(dataset.classes)
print(dataset.class_to_idx)

# Split test and train dataset
train_size = int(0.7 * len(dataset))
test_size = len(dataset) - train_size
```

```
train_data, test_data = random_split(dataset, [train_size, test_size])

# Set batch size of train data loader
batch_size_train = 20

# Set batch size of test data loader
batch_size_test = 22

# load the split train and test data into batches via DataLoader()
train_loader = DataLoader(train_data, batch_size=batch_size_train,
shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size_test,
shuffle=True)

label_map={
    0:"Chickenpox",
    1:"Measles",
    2:"Monkeypox",
    3:"Normal"
}
classes = ('Chickenpox', 'Measles', 'Monkeypox', 'Normal')

dataloaders={}
dataloaders["train"]=train_loader
dataloaders["val"]=test_loader

dataset_sizes =
{"train":len(train_loader.dataset),"val":len(test_loader.dataset)}
```

```

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
num_epochs=30

def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.repeat(1, 3, 1, 1)

                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() /
dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc:
{epoch_acc:.4f}')

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m
{time_elapsed % 60:.0f}s')
    print(f'Best val Acc: {best_acc:4f}')

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
def accuracy(model, test_loader) :
    model.eval()
    with torch.no_grad():
        running_corrects=0
        for inputs, labels in test_loader:
            inputs = inputs.repeat(1, 3, 1, 1)
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            running_corrects += torch.sum(preds == labels.data)
        acc = running_corrects.double() / dataset_sizes["val"]
    return acc

from collections import Counter

train_classes = [dataset.targets[i] for i in train_data.indices]
print("train:",Counter(train_classes)) # if doesn' work:
Counter(i.item() for i in train_classes)

test_classes = [dataset.targets[i] for i in test_data.indices]
print("Test:",Counter(test_classes)) # if doesn' work: Counter(i.item()
for i in train_classes)

print("Total:",dict(Counter(test_data.dataset.targets)))
```

#RESNET 18

```
# Path where the model is saved
PATH = '/content/drive/MyDrive/resnet18_net.pth'

# Load the pretrained model
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, len(classes))
model_ft = model_ft.to(device)

# Load the saved model state if it exists
if os.path.isfile(PATH):
    print("Loading saved model...")
    model_ft.load_state_dict(torch.load(PATH, map_location=device))
```

```
# Set the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)

# Continue training from the saved state
print("Continuing training...")
model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler, num_epochs=num_epochs)

# Save the updated model state to the same file
torch.save(model_ft.state_dict(), PATH)

# Optionally, load the updated model and evaluate (for testing
purposes)
model_ft.eval()  # Set the model to evaluation mode
print(accuracy(model_ft, test_loader))
```

# AlexNet

```
#PATH = './alex_net.pth'
PATH = '/content/drive/MyDrive/alex_net.pth'

# setup
model_ft = models.alexnet(pretrained=True,)
model_ft.classifier[6] = nn.Linear(4096,len(classes))
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)

# train
model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                       num_epochs=num_epochs)

torch.save(model_ft.state_dict(), PATH)

# load
model_ft3 = models.alexnet(pretrained=True)
model_ft3.classifier[6] = nn.Linear(4096,len(classes))
model_ft3.to(device)
model_ft3.load_state_dict(torch.load(PATH,map_location=device))
model_ft3.eval()

# test
print(accuracy(model_ft3, test_loader))
```

#Densenet121

```
model_ft = models.densenet121(pretrained=True,)
model_ft.classifier=nn.Linear(1024,len(classes))
model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)
model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                       num_epochs=num_epochs)

torch.save(model_ft.state_dict(), PATH)
model_ft3 = models.densenet121(pretrained=True)
model_ft3.classifier=nn.Linear(1024,len(classes))
model_ft3.to(device)

model_ft3.load_state_dict(torch.load(PATH,map_location=device))
model_ft3.eval()
print(accuracy(model_ft3, test_loader))
```

# Densenet161

```
model_ft = models.densenet161(pretrained=True,)
model_ft.classifier=nn.Linear(2208,len(classes))
model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
```

```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)


model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                       num_epochs=num_epochs)

torch.save(model_ft.state_dict(), PATH)
model_ft3 = models.densenet161(pretrained=True)
model_ft3.classifier=nn.Linear(2208,len(classes))
model_ft3.to(device)

model_ft3.load_state_dict(torch.load(PATH,map_location=device))
model_ft3.eval()


print(accuracy(model_ft3, test_loader))
```

#Densenet169

```
PATH = '/content/drive/MyDrive/densenet169.pth'

model_ft = models.densenet169(pretrained=True,)
print(model_ft.classifier)
model_ft.classifier=nn.Linear(1664,len(classes))
model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)

# train
model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                       num_epochs=num_epochs)

torch.save(model_ft.state_dict(), PATH)

#test
model_ft3 = models.densenet169(pretrained=True)
model_ft3.classifier=nn.Linear(1664,len(classes))
model_ft3.to(device)

model_ft3.load_state_dict(torch.load(PATH,map_location=device))
```

```
model_ft3.eval()


print(accuracy(model_ft3, test_loader))
```

# densenet201

```
model_ft = models.densenet201(pretrained=True,)
# print(model_ft.classifier)
model_ft.classifier=nn.Linear(1920,len(classes))
model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)


model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                       num_epochs=num_epochs)

torch.save(model_ft.state_dict(), PATH)
model_ft3 = models.densenet201(pretrained=True)
model_ft3.classifier=nn.Linear(1920,len(classes))
model_ft3.to(device)

model_ft3.load_state_dict(torch.load(PATH,map_location=device))
model_ft3.eval()
print(accuracy(model_ft3, test_loader))
```

#Squeezenet1_0

```
PATH = '/content/drive/MyDrive/squeezenet1_0.pth'

# setup
model_ft = models.squeezenet1_0(pretrained=True,)
model_ft.classifier[1] = nn.Conv2d(512, len(classes),
kernel_size=(1,1), stride=(1,1))
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
```

```
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7,
gamma=0.1)

# train
model_ft = train_model(model_ft, criterion, optimizer_ft,
exp_lr_scheduler,
                       num_epochs=num_epochs)

torch.save(model_ft.state_dict(), PATH)

# load
model_ft3 = models.squeezenet1_0(pretrained=True,)
model_ft3.classifier[1] = nn.Conv2d(512, len(classes),
kernel_size=(1,1), stride=(1,1))
model_ft3.to(device)

model_ft3.load_state_dict(torch.load(PATH,map_location=device))
model_ft3.eval()

# test
print(accuracy(model_ft3, test_loader))
model_ft3 = models.squeezenet1_1(pretrained=True,)
model_ft3.classifier[1] = nn.Conv2d(512, len(classes),
kernel_size=(1,1), stride=(1,1))
model_ft3.to(device)

model_ft3.load_state_dict(torch.load(PATH,map_location=device))
model_ft3.eval()
```

#YOLO V8

```
!pip install ultralytics
### Install the Split - folder package
!pip install split-folders[full]
#### Import the package
import splitfolders
###### Path of input folder
input_folder = '/content/drive/MyDrive/Monkeypox Skin Image Dataset'
######ration of split is 70%, 20% and 10%
splitfolders.ratio(input_folder, output="dataset_train2",
                   seed=42, ratio=(.7, .2, .1),
                   group_prefix=None)

import os

from ultralytics import YOLO
model = YOLO("yolov8n-cls.pt")
```

```
results = model.train(data='/content/dataset_train2', epochs=50,
imgsz=600)
model.save('/content/drive/MyDrive/yolo_v8_best_model_20_11.pt')

#Loading the Saved Model for Future Use: To use the saved model in the
future, you can load it as follows:


from ultralytics import YOLO

# Load the saved model
model = YOLO('/content/drive/MyDrive/yolo_v8_best_model_20_11.pt')

# Use the model for inference or further training
results =
model.predict('/content/drive/MyDrive/dataset_train/train/Chickenpox/ch
ickenpox10.png')
```

# WEB APP CODE
# Streamlit main.py file

```
import os
import streamlit as st
import numpy as np
from PIL import  Image

# Custom imports
from multipage import MultiPage

from pages import poxAnalysis
# Create an instance of the app
app = MultiPage()

# Title of the main page
display = Image.open('cover.jpg')
display = np.array(display)
st.image(display)
st.title("Skin Image based Pox Analysis")
st.text("Pox Affected Or Not: To detect chances of MonkeyPox, Measles
and ChickenPox")

# col1 = st.columns(1)
# col1, col2 = st.columns(2)
# col1.image(display, width = 400)
# col2.title("Data Storyteller Application")

# Add all your application here
app.add_page("Pox Analysis", poxAnalysis.app)
```

```
# app.add_page("Detect Disaster Type", detectDisaster.app)

# The main app
app.run()
```

# Model prediction code

```
import torch
from torchvision import models, transforms
from PIL import Image
import numpy as np
from ultralytics import YOLO

label_map = {
    0: "Chickenpox",
    1: "Measles",
    2: "Monkeypox",
    3: "Normal"
}
classes = ('Chickenpox', 'Measles', 'Monkeypox', 'Normal')
PATH = 'models/resnet18_net.pth'
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

data_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

def load_model():
    """Load the ResNet18 model."""
    model = models.resnet18(pretrained=True)
    num_ftrs = model.fc.in_features
    model.fc = torch.nn.Linear(num_ftrs, len(classes))
    model.to(device)
    model.load_state_dict(torch.load(PATH, map_location=device))
    model.eval()
    return model

def image_loader(image_name):
    """load image, returns cuda tensor"""

    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
```

```
    picture = Image.open(image_name)
    image = data_transform(picture)
    images=image.reshape(1,1,64,64)
    new_images = images.repeat(1, 3, 1, 1)
    return new_images

def predict(model, image_name):
    '''

    pass the model and image url to the function
    Returns: a list of pox types with decreasing probability
    '''
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    picture = Image.open(image_name)
    image = data_transform(picture)
    images=image.reshape(1,1,64,64)
    new_images = images.repeat(1, 3, 1, 1)

    outputs=model(new_images)

    _, predicted = torch.max(outputs, 1)
    ranked_labels=torch.argsort(outputs,1)[0]
    probable_classes=[]
    for label in ranked_labels:
        probable_classes.append(classes[label.numpy()])
    probable_classes.reverse()
    return probable_classes

# YOLOv8 Integration
def load_yolov8_model():
    """Load the YOLOv8 model."""
    yolov8_model = YOLO('models\last.pt')
    return yolov8_model

def predict_yolov8(model, image_file):
    """Make predictions using YOLOv8."""
    image = Image.open(image_file)
    image = np.array(image)
    results = model(image)
    detections = results[0]
    class_names = detections.names
    confidences = detections.probs.data.tolist()
    detected_class = class_names[np.argmax(confidences)]
    return detected_class, confidences
```

Reference

1. Bulletin of the World Health Organization. Available online: https://www.who.int/publications/journals/bulletin (accessed on 23 November 2023).
2. Heymann, D.L.; Szczeniowski, M.; Esteves, K. Re-emergence of monkeypox in Africa: A review of the past six years. Br. Med. Bull. 1998, 54, 693–702. [Google Scholar] [CrossRef]
3. Bragazzi, N.L.; Kong, J.D.; Mahroum, N.; Tsigalou, C.; Khamisy-Farah, R.; Converti, M.; Wu, J. Epidemiological trends and clinical features of the ongoing monkeypox epidemic: A preliminary pooled data analysis and literature review. J. Med. Virol. 2023, 95, e27931. [Google Scholar] [CrossRef]
4. Wilson, M.E.; Hughes, J.M.; McCollum, A.M.; Damon, I.K. Human Monkeypox. Clin. Infect. Dis. 2014, 58, 260–267. [Google Scholar] [CrossRef]
5. Ahsan, M.M.; Uddin, M.R.; Farjana, M.; Sakib, A.N.; Al Momin, K.; Luna, S.A. Image Data collection and implementation of deep learning-based model in detecting Monkeypox disease using modified VGG16. arXiv 2022, arXiv:2206.0186. Available online: https://arxiv.org/abs/2206.01862v1 (accessed on 23 November 2023).
6. Abdelhamid, A.A.; El-Kenawy, E.S.M.; Khodadadi, N.; Mirjalili, S.; Khafaga, D.S.; Alharbi, A.H.; Ibrahim, A.; Eid, M.M.; Saber, M. Classification of monkeypox images based on transfer learning and the Al-Biruni Earth Radius Optimization algorithm. Mathematics 2022, 10, 3614. [Google Scholar] [CrossRef]
7. Sahin, V.H.; Oztel, I.; Yolcu Oztel, G. Human monkeypox classification from skin lesion images with deep pre-trained network using mobile application. J. Med. Syst. 2022, 46, 79. [Google Scholar] [CrossRef] [PubMed]
8. Hussain, M.A.; Islam, T.; Chowdhury, F.U.H.; Islam, B.R. Can Artificial Intelligence Detect Monkeypox from Digital Skin Images? bioRxiv 2022. [Google Scholar] [CrossRef]
9. Sitaula, C.; Shahi, T.B. Monkeypox Virus Detection Using Pre-trained Deep Learning-based Approaches. J. Med. Syst. 2022, 46, 78. [Google Scholar] [CrossRef] [PubMed]
10. Ali, S.N.; Ahmed, M.T.; Paul, J.; Jahan, T.; Sani, S.M.; Noor, N.; Hasan, T. Monkeypox Skin Lesion Detection Using Deep Learning Models: A Feasibility Study. arXiv 2022, arXiv:2207.03342. Available online: https://arxiv.org/abs/2207.03342v1 (accessed on 23 November 2023).
11. M. Altun, H. Gürüler, O. Özkaraca, F. Khan, J. Khan and Y. Lee, "Monkeypox detection using CNN with transfer learning", Sensors, vol. 23, no. 4, pp. 1783, Feb. 2023 Show in Context CrossRef Google Scholar
12. R. Pramanik, B. Banerjee, G. Efimenko, D. Kaplun and R. Sarkar, "Monkeypox detection from skin lesion images using an amalgamation of CNN models aided with beta function-based normalization scheme", PLoS ONE, vol. 18, no. 4, Apr. 2023 Show in Context CrossRef Google Scholar

13. C. Sitaula and T. B. Shahi, "Monkeypox virus detection using pre-trained deep learning-based approaches", J. Med. Syst., vol. 46, no. 11, pp. 78, Oct. 2022. Show in Context CrossRef Google Scholar
14. A. Sorayaie Azar, A. Naemi, S. Babaei Rikan, J. Bagherzadeh Mohasefi, H. Pirnejad and U. K. Wiil, "Monkeypox detection using deep neural

networks", BMC Infectious Diseases, vol. 23, no. 1, pp. 438, Jun. 2023. Show in Context CrossRef Google Scholar
15. M. M. Ahsan, M. S. Ali, M. M. Hassan, T. A. Abdullah, K. D. Gupta, U. Bagci, et al., "Monkeypox diagnosis with interpretable deep learning", IEEE Access, vol. 11, pp. 81965-81980, 2023.
Show in Context View Article Google Scholar
16. A. S. Jaradat, R. E. Al Mamlook, N. Almakayeel, N. Alharbe, A. S. Almuflih, A. Nasayreh, et al., "Automated monkeypox skin lesion detection using deep learning and transfer learning techniques", Int. J. Environ. Res. Public Health, vol. 20, no. 5, pp. 4422, Mar. 2023.
Show in Context CrossRef Google Scholar
17. S. Nafisa Ali, M. Tazuddin Ahmed, T. Jahan, J. Paul, S. M. Sakeef Sani, N. Noor, et al., "A web-based mpox skin lesion detection system using state-of-the-art deep learning models considering racial diversity", arXiv:2306.14169, 2023.
Show in Context Google Scholar
18. M. M. Ahsan, M. R. Uddin, M. S. Ali, M. K. Islam, M. Farjana, A. N. Sakib, et al., "Deep transfer learning approaches for monkeypox disease diagnosis", Exp. Syst. Appl., vol. 216, Apr. 2023.
Show in Context CrossRef Google Scholar
19. A. I. Saleh and A. H. Rabie, "Human monkeypox diagnose (HMD) strategy based on data mining and artificial intelligence techniques", Comput. Biol. Med., vol. 152, Jan. 2023.
Show in Context CrossRef Google Scholar
20. A. Gupta, M. Bhagat and V. Jain, "Blockchain-enabled healthcare monitoring system for early monkeypox detection", J. Supercomput., vol. 79, no. 14, pp. 15675-15699, Sep. 2023.
Show in Context CrossRef Google Scholar
21. H. Rashid, M. A. Tanveer and H. Aqeel Khan, "Skin lesion classification using GAN based data augmentation", Proc. 41st Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC), pp. 916-919, Jul. 2019.
Show in Context View Article Google Scholar
22. Z. Qin, Z. Liu, P. Zhu and Y. Xue, "A GAN-based image synthesis method for skin lesion classification", Comput. Methods Programs Biomed., vol. 195, Oct. 2020.
Show in Context CrossRef Google Scholar
23. C. Zhao, R. Shuai, L. Ma, W. Liu, D. Hu and M. Wu, "Dermoscopy image classification based on StyleGAN and DenseNet201", IEEE Access, vol. 9, pp. 8659-8679, 2021.
Show in Context View Article Google Scholar
24. P. Sedigh, R. Sadeghian and M. T. Masouleh, "Generating synthetic medical images by using GAN to improve CNN performance in skin

cancer classification", Proc. 7th Int. Conf. Robot. Mechatronics (ICRoM), pp. 497-502, Nov. 2019.
Show in Context View Article Google Scholar

25. A. Bissoto, E. Valle and S. Avila, "GAN-based data augmentation and anonymization for skin-lesion analysis: A critical review", Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), pp. 1847-1856, Jun. 2021.
Show in Context View Article Google Scholar

*****THE END*****