PROJECT REPORT
A Deep learning approach to classify Monkeypox Skin Lesion

Introduction

1.1 Project Overview

Monkeypox is a rare viral disease that can cause a variety of skin lesions in humans. Early and accurate diagnosis is crucial for effective treatment and containment.
Objective: The goal of this project is to develop a deep learning model that can classify Monkeypox skin lesions from images, aiding in rapid and accurate diagnosis.

1.2 Purpose

purpose of developing a deep learning model for classifying Monkeypox skin lesions serves several important goals:
1) primary purpose is to facilitate early and accurate diagnosis of Monkeypox based on skin lesions.
2) Create an accessible tool that can be used by healthcare professionals, especially in regions with limited access to specialized medical expertise.
3) Promote the use of advanced technologies, such as deep learning, in the field of medical diagnostics.

2 Literature Survey

2.1 Existing problem

The difficulty in obtaining specialist dermatological knowledge is a major obstacle in the diagnosis and categorization of skin lesions associated with monkeypox, particularly in areas with poor healthcare resources. Many places may lack quick access to dermatologists or infectious disease specialists who are skilled in recognizing and treating uncommon illnesses like monkeypox within the current healthcare system.

2.2 References

- https://towardsdatascience.com/
- https://www.kaggle.com/
- https://stackoverflow.com/
- https://www.udemy.com/

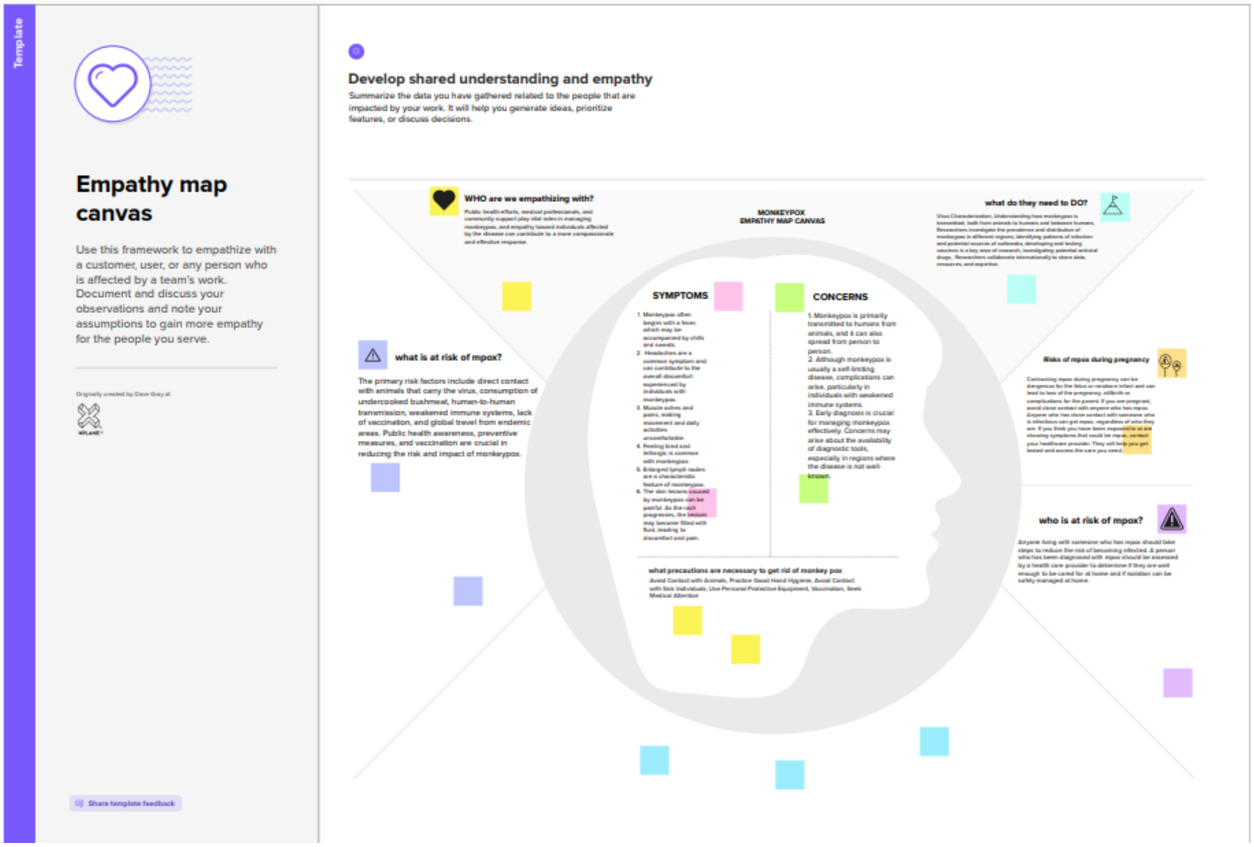
2.3 Problem Statement Definition

The difficulty in obtaining specialist dermatological knowledge is a major obstacle in the diagnosis and categorization of skin lesions associated with monkeypox, particularly in areas with poor healthcare resources. Many places may lack quick access to dermatologists or infectious disease specialists who are skilled in recognizing and treating uncommon illnesses like monkeypox within the current healthcare system.

With the use of deep learning, this research seeks to provide medical practitioners with an automated diagnostic tool that can quickly and accurately identify skin lesions associated with monkeypox. Deep learning technology is being applied to improve early detection and treatment, as well as to support public health initiatives by making advanced diagnostics more widely accessible and possibly leading to better patient outcomes. This research is in line with the more general objectives of early disease detection, technology innovation, and accessibility to healthcare.

3) Ideation and Proposed Solution

3.1 Empathy mapping



4 Requirement Analysis

4.1 Fundamental Requirement

Solution Architecture:

- Data Ingestion: Image Data is taken from the Kaggle as provided and its structure is analyzed.

- Data Preprocessing: Involves cleaning and normalization to ensure consistent pixel values and handle outliers. Techniques like rescaling and augmentation are applied for numerical stability and improved model generalization. Missing or corrupted images are addressed to maintain dataset integrity. Labeling is performed based on directory structure or external files. The dataset is often split for effective model evaluation.

- CNN: CNNs consist of convolutional layers that learn spatial hierarchies of features, pooling layers for dimensionality reduction, and fully connected layers for classification or regression tasks.

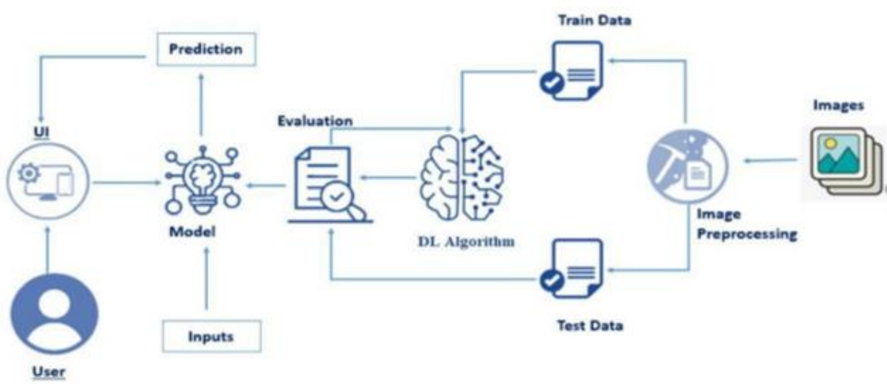
- ResNet50: ResNet-50 has an architecture based on the model depicted above, but with one important difference. The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a “bottleneck”, which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer. It uses a stack of three layers rather than two layers.

- Model Evaluation: Evaluate different model classification performance based on their accuracy score.

- Saving the Model: Select the most optimal classification technique among the four based on the evaluation metrics, and save the model that demonstrates the most favorable results.

- User Interface: Create an intuitive user interface for the web application ensuring a user-friendly experience.

Solution Architecture Diagram:



5) Project Design

User Stories:

User Type	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Data Scientist	USN-1	Identify Monkeypox Skin Lesions in Images	Develop a deep learning model for lesion detection. Model accuracy should exceed 90%.	High	-
Researcher	USN-2	Analyze and Classify Monkeypox Skin Lesions	Implement a classification system for different types of lesions. Accuracy goal: 85%.	High	-
Medical Expert	USN-3	Validate Deep Learning Results	Engage medical professionals to review and validate model predictions on real-world cases.	Medium	-
UI/UX Designer	USN-4	Design User Interface for PoxVisio	Create an intuitive interface for uploading images, viewing results, and accessing reports.	Medium	-
Developer	USN-5	Integrate PoxVisio with Existing Healthcare Systems	Ensure seamless integration with hospital databases and health information systems.	High	-

6) Project planning and Scheduling ->

Solution Architecture:

• Data Ingestion: Image Data is taken from the Kaggle as provided and its structure is analyzed.

• Data Preprocessing: Involves cleaning and normalization to ensure consistent pixel values and handle outliers. Techniques like rescaling and augmentation are applied for numerical stability and improved model generalization. Missing or corrupted images are addressed to maintain dataset integrity. Labeling is performed based on directory structure or external files. The dataset is often split for effective model evaluation.

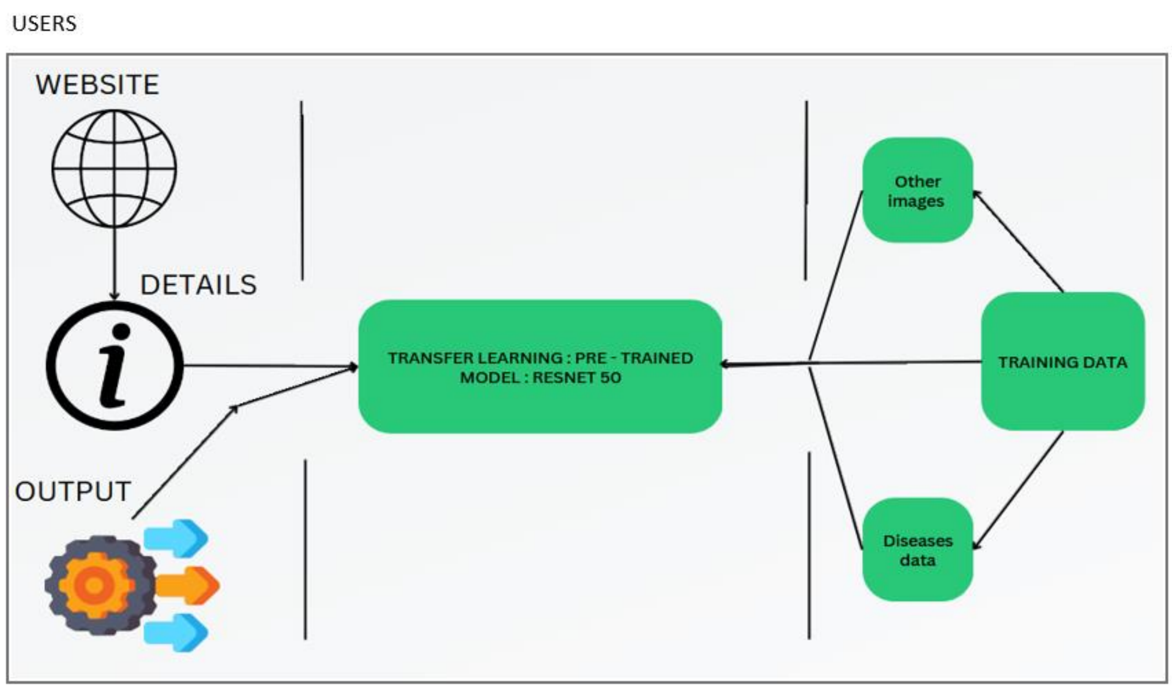
• CNN: CNNs consist of convolutional layers that learn spatial hierarchies of features, pooling layers for dimensionality reduction, and fully connected layers for classification or regression tasks.

• ResNet50: ResNet-50 has an architecture based on the model depicted above, but with one important difference. The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a “bottleneck”, which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer. It uses a stack of three layers rather than two layers.

- Model Evaluation: Evaluate different model classification performance based on their accuracy score.

- Saving the Model: Select the most optimal classification technique among the four based on the evaluation metrics, and save the model that demonstrates the most favorable results.

- User Interface: Create an intuitive user interface for the web application ensuring a user-friendly experience.



6.2 Sprint plan ->

Sprint	Functional Requirement(Epic)	User Story Number	UserStory/Task	Story Points	Priority	Team Members
Spirit 1	Collecting the data	USN-1: Collecting the dats so that the model can be trained	To train the model we will be needing datasets	11	High	AARIZ ZAFAR
Spirit 1	Segmenting the data into different classes	USN-2: Classifying the images into diseased and not diseased	The model will need 2 classes to where there will be images of a person with the disease and a person who Is not suffering from the diseases	9	High	AARIZ ZAFAR
Spirit 2	Model training	Using ResNet50, to train the model	The model has to be trained on this data so that it can classify and predict.	20	HIGH	AARIZ ZAFAR
Spirit 3	Model Prediction and evaluation	The model has to evaluated	The models evaluation will be done and the accuracy score will be calculated.	20	HIGH	AARIZ ZAFAR
Spirit 4	Web interface development and app development	USN-5: We need an interface, app where we can upload the image to classify it	The image will be uploaded locally so that it can be classified	20	Medium	AARIZ ZAFAR, PRAKALP

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2023	29 Oct 2023	20	29 Oct 2023
Sprint-2	20	6 Days	31 Oct 2023	05 Nov 2023	20	05 Nov 2023
Sprint-3	20	6 Days	07 Nov 2023	12 Nov 2023	20	12 Nov 2023
Sprint-4	20	6 Days	14 Nov 2023	19 Nov 2023	20	19 Nov 2023
Sprint-4	20	6 Days	17 Nov 2023	18 Nov 2023	20	19 Nov 2023

7 Coding and Solutioning

7.1 Feature 1 - Transfer Learning

Transfer learning is the process of using expertise from one task to improve performance on another. Using a pre-trained deep learning model that is already proficient in picture recognition, we train the Monkeypox Skin Lesion project to recognize monkeypox lesions. It's similar to having a skilled painter (pre-trained model) who gains the ability to create a particular kind of artwork (lesions from monkeypox). Considering that the model doesn't start from beginning, this expedites learning. Using modular code gives it structure. We effectively repurpose and modify components, increasing the project's effectiveness and managing it more easily. In summary, transfer learning builds on prior knowledge to assist our model in becoming an expert in identifying skin lesions associated with monkeypox.

7.2 Web application

```
<!DOCTYPE html>

<html lang="en">

<head>

   <meta charset="UTF-8">

   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <meta http-equiv="X-UA-Compatible" content="ie=edge">

   <title>cnncls</title>

        <link rel="shortcut icon"

         <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

        <style>
```

```
body{background-color: #eff2f9;}

.iupload h3{color: #1b2d6b;font-size: 30px;font-weight: 700;}

.img-part{height:300px;width:300px;margin:0px auto;}

.image-part{height:300px;width:300px;border:1px solid #1b2d6b;}

.image-part img{position:absolute;height:
300px;width:300px;display:none;padding:5px;}

.image-part #video{display:block;height: 300px;width:300px;padding:5px;}

.res-part{border:1px solid #dedede;margin-left:20px;height:
310px;width:100%;padding:5px;margin:0px auto;overflow:auto;}

.res-part2{border:1px solid #dedede;height:
310px;width:100%;padding:5px;margin:0px auto;}

.resp-img{height: 298px;width: 233px;margin:0px auto;}

.jsonRes{margin-left:30px;}

#send{cursor:pointer;}

.btn-part{width:325px;}

textarea,

select,

.form-control,

.custom-select,

button.btn,

.btn-primary,

input[type="text"],

input[type="url"],

.uneditable-input{

    border: 1px solid #363e75;
```

```
            outline: 0 !important;
            border-radius:0px;
            box-shadow: none;
    -webkit-box-shadow: none;
    -moz-box-shadow: none;
    -moz-transition: none;
    -webkit-transition: none;
}
textarea:focus,
select:focus,
.form-control:focus,
.btn:focus,
.btn-primary:focus,
.custom-select:focus,
input[type="text"]:focus,
.uneditable-input:focus{
            border: 1px solid #007bff;
            outline: 0 !important;
            border-radius:0px;
            box-shadow: none;
    -webkit-box-shadow: none;
    -moz-box-shadow: none;
    -moz-transition: none;
```

```
  -webkit-transition: none;
}
#loading {
        position: fixed;
        left: 0px;
        top: 0px;
        width: 100%;
        height: 100%;
        z-index: 9999999999;
        overflow: hidden;
        background: rgba(255, 255, 255, 0.7);
}
.loader {
        border: 8px solid #f3f3f3;
        border-top: 8px solid #363e75;
        border-radius: 50%;
        width: 60px;
        height: 60px;
        left: 50%;
        margin-left: -4em;
        display: block;
        animation: spin 2s linear infinite;
}
```

```
                .loader,
                .loader:after {display: block;position: absolute;top: 50%;margin-top: -
4.05em;}
                @keyframes spin {
                        0% {
                                transform: rotate(0deg);
                        }
                        100% {
                                transform: rotate(360deg);
                        }
                }
                .right-part{border:1px solid #dedede;padding:5px;}
                .logo{position:absolute;right:0px;bottom:0px;margin-right:30px;margin-
bottom:30px;}
        </style>
</head>
<body>
   <div class="main container">
                <section class="iupload">
                        <h3 class="text-center py-4">Object Classification</h3>
                        <div class="row">
                                <div class="img-part col-md-6">
                                        <div class="image-part">
```

```html
<video autoplay id="video"
poster="https://img.freepik.com/free-vector/group-young-people-posing-photo_52683-18824.jpg?size=338&ext=jpg"></video>

<img src="" id="photo">

<canvas style="display:none;"
id="canvas"></canvas>

</div>

<div class="btn-part">

<form id="upload-data pt-3" class="">

<div class="input-group mt-3 row">

<button type="button" class="btn
btn-primary col-md-5 col-xs-5 ml-3 mr-4" id="uload">Upload</button>

<button id="send" type="button"
class="btn btn-success col-md-5 col-xs-5">Predict</button>

</div>

<!-- change url value  -->

<input type="hidden" class="form-
control mr-2" id="url" placeholder="Enter REST Api url..." value="../predict"/>

<input name="upload" type="file"
id="fileinput" style="position:absolute;top:-500px;"/><br/>

</form>

</div>

</div>

<div class="col-md-6 col-xs-12 right-part">

<h5 class="mb-2"><center>Prediction
Results</center></h5>

<div class="row">
```

```
                                                <div class="res-part2 col-md-5 col-xs-
12"></div>

                                                <div class="res-part col-md-5 col-xs-12"><div
class="jsonRes"></div></div>

                                        </div>

                                </div>

                        </div>

                </section>

        </div>


<script>

var mybtn = document.getElementById('startbtn');

var myvideo = document.getElementById('video');

var mycanvas = document.getElementById('canvas');

var myphoto = document.getElementById('photo');

var base_data = "";


function sendRequest(base64Data){

        var type = "json";

        if(base64Data != "" || base64Data != null){

                if(type == "imgtobase"){

                        $(".res-part").html("");

                        $(".res-part").html(base64Data);

                }
```

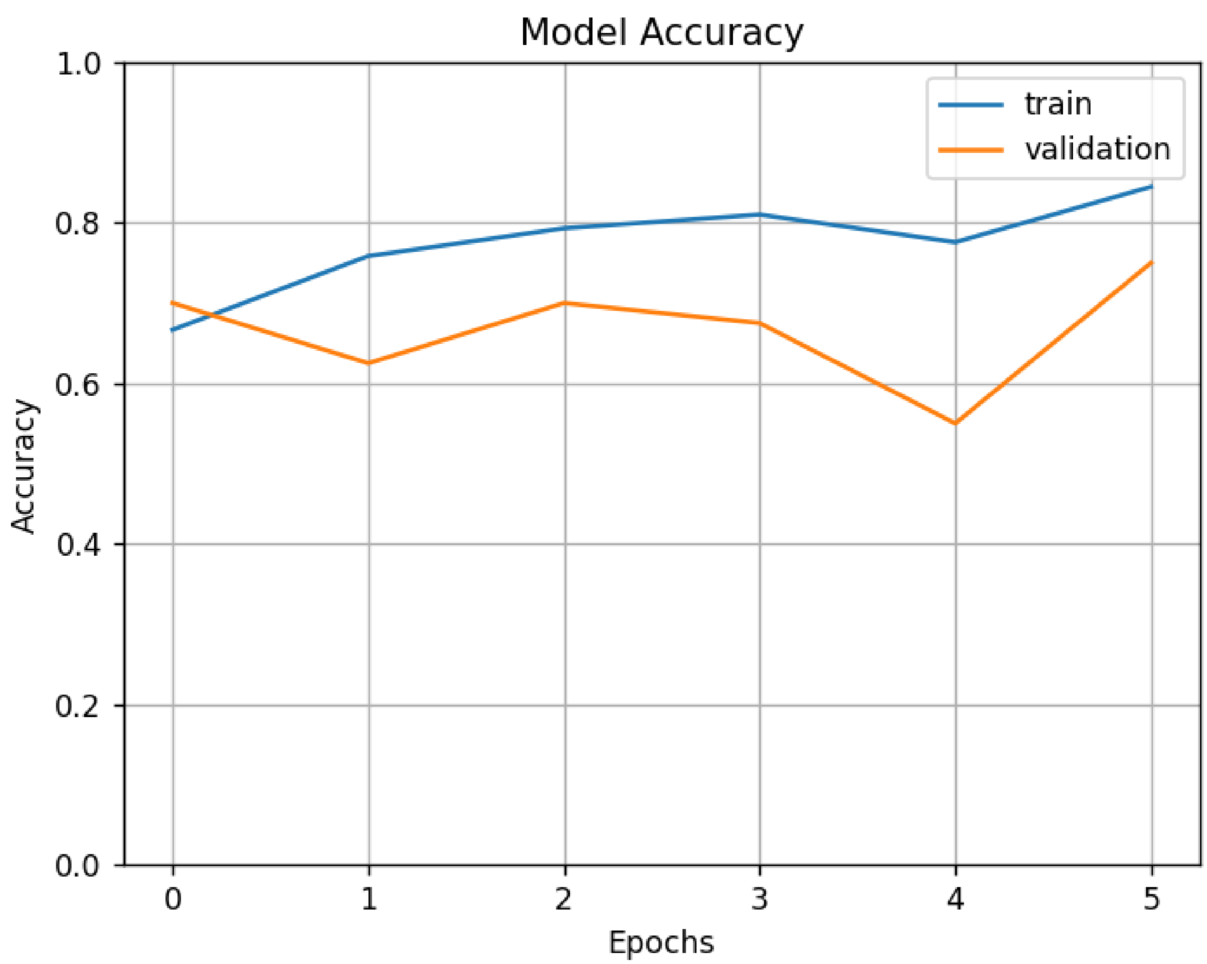
```
        else if(type == "basetoimg"){

            var imageData = $("#imgstring").val();

            $(".res-part").html("");

            $(".res-part").append("<img src='data:image/jpeg;base64," +
imageData + "' alt='' />");

        }

        else{

            var url = $("#url").val();

            $("#loading").show();

            $.ajax({

                url : url,

                type: "post",

                cache: false,

                async: true,

                crossDomain: true,

                headers: {

                    'Content-Type': 'application/json',

                    'Access-Control-Allow-Origin':'*'

                },

                data:JSON.stringify({image:base64Data}),

                success: function(res){

                    $(".res-part").html("");

                    $(".res-part2").html("");

                    try{
```

```
                                        var imageData = res[1].image;

                                        if(imageData.length > 100){

                                             if(imageData.length > 10){$(".res-
part2").append("<img class='resp-img' src='data:image/jpeg;base64," + imageData + "'
alt='' />");}

                                        }

                                   }catch(e){}

                                   $(".res-part").html("<pre>" + JSON.stringify(res[0],
undefined, 2) + "</pre>");

                                   $("#loading").hide();

                              }

                         });

                    }

               }

          }



$(document).ready(function(){

     $("#loading").hide();



     $('#send').click(function(evt){

          sendRequest(base_data);

  });



  $('#uload').click(function(evt) {

    $('#fileinput').focus().trigger('click');
```

```
    });
        $("#fileinput").change(function(){
                if (this.files && this.files[0]){
                        var reader = new FileReader();
                        reader.onload = function (e){
                                var url = e.target.result;
                                var img = new Image();
                                img.crossOrigin = 'Anonymous';
                                img.onload = function(){
                                        var canvas = document.createElement('CANVAS');
                                        var ctx = canvas.getContext('2d');
                                        canvas.height = this.height;
                                        canvas.width = this.width;
                                        ctx.drawImage(this, 0, 0);
                                        base_data = canvas.toDataURL('image/jpeg',
1.0).replace(/^data:image.+;base64,/, '');
                                        canvas = null;
                                };
                                img.src = url;
                                $('#photo').attr('src', url);
                                $('#photo').show();
                                $('#video').hide();
                        }
                        reader.readAsDataURL(this.files[0]);
```

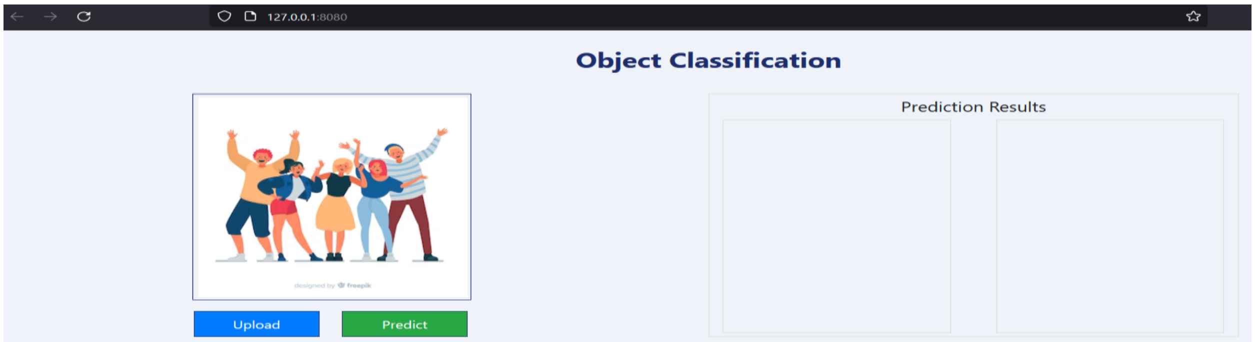
```
                    }
          });
});


</script>
</body>
</html>
```

8) **Performance testing**



9) RESULT

Output Screenshots



Adding in images

10) Advantage and Disadvantages

Advantage :

Early Detection: By assisting in the early identification of monkeypox, the study may improve patient outcomes and enable prompt medical intervention.

Resource Efficiency: By utilizing pre-trained models, transfer learning reduces the amount of data and processing power needed to provide accurate results, therefore saving resources.

Automation: The diagnosing process can be sped up by using automated skin lesion classification, giving medical personnel faster feedback.

rural Access: By utilizing a trained model, medical professionals in underserved or rural places can obtain professional diagnosis assistance, hence enhancing healthcare accessible.

Learning and Research: The project promotes additional research and advances in medical image analysis by expanding knowledge at the nexus of deep learning and healthcare.

Disadvantage

The Monkeypox Skin Lesion Classification Project's benefits include:

Early Detection: By assisting in the early identification of monkeypox, the study may improve patient outcomes and enable prompt medical intervention.

Resource Efficiency: By utilizing pre-trained models, transfer learning reduces the amount of data and processing power needed to provide accurate results, therefore saving resources.

Automation: The diagnosing process can be sped up by using automated skin lesion classification, giving medical personnel faster feedback.

rural Access: By utilizing a trained model, medical professionals in underserved or rural places can obtain professional diagnosis assistance, hence enhancing healthcare accessible.

Learning and Research: The project promotes additional research and advances in medical image analysis by expanding knowledge at the nexus of deep learning and healthcare.

11) Conclusion

In conclusion, by utilizing deep learning and transfer learning, the Monkeypox Skin Lesion Classification Project significantly advances healthcare. The project improves diagnosis speed and accuracy by providing a workable solution for early monkeypox detection through the use of pre-trained models. This may result in prompt medical intervention, which would ultimately enhance patient outcomes and support initiatives related to public health. The project's modular coding strategy guarantees effective development, which facilitates management and long-term adaptation.

Nonetheless, it's critical to recognize the obstacles the project faces. Data biases and possible over-reliance on technology are two ethical issues that need to be carefully considered. Responsible deployment of such systems in real-world medical settings requires striking a balance between the advantages of automation and the value of human expertise. The model must be updated and maintained continuously in order to reflect new developments in medicine. Essentially, the project highlights the necessity for a careful and moral integration of artificial intelligence in healthcare, stressing the cooperation between technology and human expertise for the best patient care, even though it offers promising solutions for disease detection.

12) Future Scope

The Monkeypox Skin Lesion Classification Project has the potential to significantly transform disease diagnosis and healthcare accessibility in the future. The project can develop to support a wider range of infectious diseases and skin conditions as technology advances, becoming a useful tool for medical professionals everywhere. Through continued investigation and cooperation, the model can be improved to tackle new issues in healthcare and adjust to the constantly growing body of knowledge about infectious diseases and dermatology.

Furthermore, the project establishes the foundation for the creation of intuitive user interfaces and mobile applications that enable healthcare providers to obtain prompt and precise diagnostic support, particularly in areas with limited resources. By incorporating the model into telemedicine platforms, medical practitioners in underserved areas can benefit greatly from improved remote patient care. Collaborations with governments and international health organizations can also make it easier to implement the project as a component of public health campaigns, which will help with the larger-scale early detection, containment, and management of infectious diseases.

Essentially, the project's future scope goes beyond monkeypox, positioning it as a scalable and flexible solution that could have a major impact on global healthcare by utilizing artificial intelligence to diagnose diseases accurately, quickly, and easily.

13) Appendix

Source Code

1) Data Ingestion

```
import os

import sys

from dataclasses import dataclass

from pathlib import Path

from poxVisionDetection import logging,CustomException

import urllib.request as request

import zipfile


@dataclass(frozen = True)

class DataIngestionConfig: # BELOW ARE THE RETURN TYPES

    root_dir              : Path

    source_url             : str

    local_data_file        : Path

    unzip_dir              : Path


class ConfigurationManager:

    def __init__(

            self,

            config_filepath = CONFIG_FILE_PATH,

            params_filepath = PARAMS_FILE_PATH):
```

```
        self.config = read_yaml(config_filepath)

        self.params = read_yaml(params_filepath)


        create_directory([self.config.artifacts_root])  # THIS WILL CREATE THE PARENT
DIRECTORY artifacts

                                                # WHERE ALL THE DATA RELATED FOLDERS
WILL BE PRESENT



    def get_data_ingestion_config(self) -> DataIngestionConfig:

        '''

            Will get all the data_ingestion related configuration form the config file

        '''

        config = self.config.data_ingestion


        create_directory([config.root_dir])


        data_ingestion_config = DataIngestionConfig(

            root_dir            = config.root_dir,

            source_url          = config.source_url,

            local_data_file     = config.local_data_file,

            unzip_dir           = config.unzip_dir

        )
```

```
        return data_ingestion_config


class DataIngestion:
    def __init__(self, config : DataIngestionConfig):
        self.config = config


    def download_file(self):
        '''
            will get the dataset for the remote git hub link provided


            Create local_file_folder where the file will be stored in .zip
        '''
        if not os.path.exists(self.config.local_data_file):
            filename, header = request.urlretrieve(
                url       = self.config.source_url,              # THE LINK WHERE THE FILE IS
AVAILABLE IN THE GIT HUB
                filename  = self.config.local_data_file          # THE LOCAL PATH WHERE
THE FILE WILL BE SAVED
            )
            logging.info(f'{filename} DOWNLOADED WILL THE FOLLOWING INFO :
{header}')
        else:
            logging.info(f'THE FILE ALREDY EXISTS OF SIZE :
{get_size(Path(self.config.local_data_file))}')
```

```
    def extract_zip_file(self):
        '''
            zip_file_path  : str
            Extract the zip file into the data directory
            function returns None
        '''
        unzip_path = self.config.unzip_dir
        os.makedirs(unzip_path , exist_ok = True)
        with zipfile.ZipFile(self.config.local_data_file , 'r') as zip_file:
            zip_file.extractall(unzip_path)


try:
    config                    = ConfigurationManager()
    data_ingestion_config     = config.get_data_ingestion_config()
    data_ingestion            = DataIngestion(config = data_ingestion_config)
    data_ingestion.download_file()
    data_ingestion.extract_zip_file()
except Exception as e:
    CustomException(e,sys)
```

2) Preparing base model

```
import os
import sys
```

```
from dataclasses import dataclass
from pathlib import Path
import urllib.request as request
from zipfile import ZipFile
import tensorflow as tf
from poxVisionDetection import logging,CustomException


@dataclass(frozen = True)
class PrepareBaseModelConfig:
    root_dir                : Path
    base_model_path         : Path
    updated_base_model_path : Path
    params_image_size       : list
    params_learning_rate    : float
    params_include_top      : bool
    params_weight           : str
    params_classes          : int


class ConfigurationManager:
    def __init__(
            self,
            config_filepath = CONFIG_FILE_PATH,
            params_filepath = PARAMS_FILE_PATH):
```

```
        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directory([self.config.artifacts_root])

    def get_prepare_base_model(self) -> PrepareBaseModelConfig:
        config = self.config.prepare_base_model

        create_directory([config.root_dir])

        prepare_base_model_config = PrepareBaseModelConfig(
            root_dir                 = Path(config.root_dir),
            base_model_path          = Path(config.base_model_path),
            updated_base_model_path  = Path(config.updated_base_model_path),
            params_image_size        = self.params.IMAGE_SIZE,
            params_learning_rate     = self.params.LEARNING_RATE,
            params_include_top       = self.params.INCLUDE_TOP,
            params_weight            = self.params.WEIGHTS,
            params_classes           = self.params.CLASSES
        )

        return prepare_base_model_config
```

```
class PrepareBaseModel:
    def __init__(self, config : PrepareBaseModelConfig):
        self.config = config

    def get_base_model(self):
        self.model            = tf.keras.applications.ResNet50(
            include_top        = self.config.params_include_top,
            weights            = self.config.params_weight,
            input_shape        = self.config.params_image_size,
        )

        # THE BASE MODEL WILL GET SAVED IN THE PATH PROVIDED
        self.save_model(path    = self.config.base_model_path,
                        model   = self.model)

    # THE WEIGHTS THAT ARE PRESENT IN THE ResNet50 MODEL ARE GOING TO BE USED AS SUCH ONLY THE INPUT AND OUTPUT LAYERS ARE GOING TO BE TRAINED
    @staticmethod
    def _prepare_full_model(model, classes, freeze_all, freeze_till, learning_rate):
        if freeze_all:
            for layer in model.layers:
                model.trainable = False
```

```
elif(freeze_till is not None) and (freeze_till > 0):
    for layer in model.layers[:-freeze_till]:
        model.trainable = False

flatten = model.output

Globalavgpool2D   = tf.keras.layers.GlobalAveragePooling2D()(flatten)

Dlayer1           = tf.keras.layers.Dense(
    units          = 64,
    activation     = 'relu'
)(Globalavgpool2D)

pred_layer           = tf.keras.layers.Dense(
    units          = classes,
    activation     = 'softmax'
)(Dlayer1)

full_model       = tf.keras.models.Model(
    inputs         = model.input,
    outputs        = pred_layer
)
```

```
        print(full_model)
        print('------------------------------------------')

        full_model.compile(
            optimizer          = tf.keras.optimizers.SGD(learning_rate = learning_rate),
            loss               = tf.keras.losses.CategoricalCrossentropy(),
            metrics            = ['accuracy']
        )

        full_model.summary()
        return full_model

    def updated_base_model(self):
        self.full_model       = self._prepare_full_model(
            model             = self.model,
            classes           = self.config.params_classes,
            freeze_all        = True,
            freeze_till       = None,
            learning_rate     = self.config.params_learning_rate
        )

        self.save_model(path  = self.config.updated_base_model_path,
                        model = self.full_model)
```

```python
    @staticmethod
    def save_model(path : Path, model : tf.keras.Model):
        print(model.summary)
        model.save(path)


try:
    config                  = ConfigurationManager()
    prepare_base_model_config    = config.get_prepare_base_model()
    prepare_base_model          = PrepareBaseModel(config = 
prepare_base_model_config)
    prepare_base_model.get_base_model()
    prepare_base_model.updated_base_model()
except Exception as e:
    logging.exception(CustomException(e,sys))
```

3) Preparing callbacks

```python
import os
import sys
import urllib.request as request
import tensorflow as tf
```

```
import time
from dataclasses import dataclass
from pathlib import Path
from poxVisionDetection import logging,CustomException
from poxVisionDetection.constants import *
from poxVisionDetection.utils.common import read_yaml,create_directory


@dataclass(frozen = True)
class PrepareCallbacksConfig:
    root_dir                   : Path
    tensorboard_root_log_dir   : Path
    checkpoint_model_filepath  : Path


class ConfigurationManager:
    def __init__(
            self,
            config_filepath = CONFIG_FILE_PATH,
            params_filepath = PARAMS_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directory([self.config.artifacts_root])
```

```
    def get_prepare_callback_config(self) -> PrepareCallbacksConfig:
        config          = self.config.prepare_callbacks
        model_ckpt_dir   = os.path.dirname(config.checkpoint_model_filepath)

        create_directory([
            Path(model_ckpt_dir),
            Path(config.tensorboard_root_log_dir)
        ])

        prepare_callback_config       = PrepareCallbacksConfig(
            root_dir                  = Path(config.root_dir),
            tensorboard_root_log_dir   = Path(config.tensorboard_root_log_dir),
            checkpoint_model_filepath  = Path(config.checkpoint_model_filepath)
        )

        return prepare_callback_config
class PrepareCallback:
    def __init__(self, config : PrepareCallbacksConfig):
        self.config = config

    @property
    def _create_tb_callbacks(self):
```

```
        timestamp = time.strftime('%y-%m-%d-%H-%M-%S')

        tb_running_log_dir = os.path.join(
            self.config.tensorboard_root_log_dir,
            f'tb_logs_at_{timestamp}'
        )

        return tf.keras.callbacks.TensorBoard(log_dir = tb_running_log_dir)

    @property
    def _create_ckpt_callbacks(self):
        return tf.keras.callbacks.ModelCheckpoint(
            filepath = self.config.checkpoint_model_filepath,
            save_best_only = True
        )

    # ckpt - checkpoint
    def get_tb_ckpt_callback(self):
        return [
            self._create_tb_callbacks,
            self._create_ckpt_callbacks
        ]
try:
```

```
    config                    = ConfigurationManager()
    prepare_callbacks_config  = config.get_prepare_callback_config()
    prepare_callbacks         = PrepareCallback(config = prepare_callbacks_config)
    callback_list             = prepare_callbacks.get_tb_ckpt_callback()

except Exception as e:
    CustomException(e,sys)
```

4) Model Training

```
import os
import sys
import time
from dataclasses import dataclass
import urllib.request as request
from zipfile import ZipFile
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input
from pathlib import Path
from poxVisionDetection.constants import *
from poxVisionDetection.utils.common import read_yaml,create_directory
from poxVisionDetection import CustomException,logging
```

```python
import matplotlib.pyplot as plt


@dataclass(frozen = True)
class TrainingConfig:
    root_dir                  : Path
    training_model_path       : Path
    updated_base_model_path   : Path
    training_data             : Path
    params_epochs             : int
    params_batch_size         : int
    params_is_augmentation    : bool
    params_image_size         : list


@dataclass(frozen = True)
class PrepareCallbacksConfig:
    root_dir                  : Path
    tensorboard_root_log_dir  : Path
    checkpoint_model_filepath : Path


class ConfigurationManager:
    def __init__(
        self,
        config_filepath             = CONFIG_FILE_PATH,
```

```
        params_filepath          = PARAMS_FILE_PATH):

        self.config              = read_yaml(config_filepath)
        self.params              = read_yaml(params_filepath)

        create_directory([self.config.artifacts_root])

    def get_prepare_callbacks_config(self) -> PrepareCallbacksConfig:
        config                   = self.config.prepare_callbacks
        model_ckpt_dir           = os.path.dirname(config.checkpoint_model_filepath)

        create_directory([
            Path(model_ckpt_dir),
            Path(config.tensorboard_root_log_dir)
        ])

        prepare_callback_config  = PrepareCallbacksConfig(
            root_dir                 = Path(config.root_dir),
            tensorboard_root_log_dir = Path(config.tensorboard_root_log_dir),
            checkpoint_model_filepath = Path(config.checkpoint_model_filepath)
        )

        return prepare_callback_config
```

```
def get_training_config(self) -> TrainingConfig:
    training                = self.config.training
    prepare_base_model      = self.config.prepare_base_model
    params                  = self.params

    training_data = os.path.join(self.config.data_ingestion.unzip_dir,
'poxVisionDataSet')
    create_directory([Path(training.root_dir)])

    training_config         = TrainingConfig(
        root_dir               = Path(training.root_dir),
        training_model_path    = Path(training.trained_model_path),
        updated_base_model_path   =
Path(prepare_base_model.updated_base_model_path),
        training_data          = Path(training_data),
        params_epochs          = params.EPOCHS,
        params_batch_size      = params.BATCH_SIZE,
        params_is_augmentation    = params.AUGMENTATION,
        params_image_size      = params.IMAGE_SIZE
    )

    return training_config
```

```
class PrepareCallback:
    def __init__(self, config : PrepareCallbacksConfig):
        self.config = config

    @property
    def _create_tb_callbacks(self):
        timestamp = time.strftime("%Y-%m-%d-%H-%M-%S")
        tb_running_log_dir = os.path.join(
            self.config.tensorboard_root_log_dir,
            f'tb_log_at_{timestamp}'
        )
        return tf.keras.callbacks.TensorBoard(log_dir = tb_running_log_dir)

    @property
    def _create_ckpt_callbacks(self):
        return tf.keras.callbacks.ModelCheckpoint(
            filepath      = 'artifacts\prepare_callbacks\checkpoint_dir\model.h5',
            save_best_only = True
        )

    def get_tb_ckpt_callbacks(self):
        return [
            self._create_tb_callbacks,
```

```
            self._create_ckpt_callbacks
        ]
class Training:
    def __init__(self, config : TrainingConfig):
        self.config = config

    def get_base_model(self):
        # LOADING THE UPDATED BASE MODEL
        self.model = tf.keras.models.load_model(
            self.config.updated_base_model_path
        )

    def training_valid_generator(self):
        valid_datagenerator        = tf.keras.preprocessing.image.ImageDataGenerator(
            preprocessing_function     = preprocess_input,
            shear_range                = 0.2,
            zoom_range                 = 0.2,
            validation_split           = 0.4,
        )

        # THIS GENERATOR HAS BEEN CREATED FOR THE TRAINING
        self.train_generator       = valid_datagenerator.flow_from_directory(
            directory                  = self.config.training_data,
```

```
        target_size              = self.config.params_image_size[:-1],
        batch_size               = self.config.params_batch_size,
        class_mode               = 'categorical',
        subset                   = 'training',
    )

    # THIS GENERATOR HAS BEEN CREATED FOR THE VALIDATION
    self.valid_generator       = valid_datagenerator.flow_from_directory(
        directory                = self.config.training_data,
        target_size              = self.config.params_image_size[:-1],
        batch_size               = self.config.params_batch_size,
        class_mode               = 'categorical',
        subset                   = 'validation',
    )

def train(self, callback_list : list):
    trained_model = self.model.fit(

        self.train_generator,

        epochs                   = self.config.params_epochs,
        steps_per_epoch          = 3,
```

```
        validation_data           = self.valid_generator,
        validation_steps          = 2,

        callbacks                 = callback_list
    )

    self.save_model(
        path                 = self.config.training_model_path,
        model                = self.model
    )

    return trained_model

def train_model_status(self, callback_list: list):
    trained_model = self.train(callback_list)
    plt.plot(trained_model.history['accuracy'])
    plt.plot(trained_model.history['val_accuracy'])
    plt.axis(ymin=0.0,ymax=1)
    plt.grid()
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epochs')
    plt.legend(['train','validation'])
```

```
        plt.show()


    @staticmethod
    def save_model(path : Path, model : tf.keras.Model):
        model.save(path)
try:
    config                         = ConfigurationManager()
    prepare_callbacks_config      = config.get_prepare_callbacks_config()
    prepare_callbacks             = PrepareCallback(config = prepare_callbacks_config)
    callback_list                 = prepare_callbacks.get_tb_ckpt_callbacks()
    training_config               = config.get_training_config()
    training                       = Training(config = training_config)


    training.get_base_model()
    training.training_valid_generator()


    training.train_model_status(
        callback_list=callback_list
    )


except Exception as e:
    logging.exception(CustomException(e,sys))
```

5) Model Evaluation

```
import os

import sys

import tensorflow as tf

from dataclasses import dataclass

from pathlib import Path

from poxVisionDetection.constants import *

from poxVisionDetection.utils.common import read_yaml, create_directory,
save_json

from tensorflow.keras.applications.resnet50 import preprocess_input

from poxVisionDetection import CustomException, logging

from urllib.parse import urlparse


@dataclass(frozen = True)

class EvaluationConfig:

    path_of_model         : Path

    training_data         : Path

    all_params            : dict

    params_image_size     : list

    params_batch_size     : int


class ConfigurationManager:
```

```
    def __init__(
        self,
        config_filepath   = CONFIG_FILE_PATH,
        params_filepath     = PARAMS_FILE_PATH):

      self.config = read_yaml(config_filepath)
      self.params = read_yaml(params_filepath)

      create_directory([self.config.artifacts_root])

    def get_validation_config(self) -> EvaluationConfig:
      eval_config = EvaluationConfig(
        path_of_model     = "artifacts/training/model.h5",
        training_data     = "artifacts/data_ingestion/poxVisionDataSet",
        all_params        = self.params,
        params_image_size = self.params.IMAGE_SIZE,
        params_batch_size = self.params.BATCH_SIZE
      )

      return eval_config
class Evaluation:
    def __init__(self,config : EvaluationConfig):
      self.config = config
```

```
def _valid_generator(self):

    valid_datagenerator        =
tf.keras.preprocessing.image.ImageDataGenerator(

        preprocessing_function     = preprocess_input,

        shear_range               = 0.2,

        zoom_range                = 0.2,

        validation_split          = 0.4,

    )


    self.valid_generator       = valid_datagenerator.flow_from_directory(

        directory                 = self.config.training_data,

        target_size               = self.config.params_image_size[:-1],

        batch_size                = self.config.params_batch_size,

        class_mode                = 'categorical',

        subset                    = 'validation',

    )


@staticmethod

def load_model(path : Path) -> tf.keras.Model:

    return tf.keras.models.load_model(path)


def evaluation(self):

    self.model                 = self.load_model(self.config.path_of_model)
```

```
        self._valid_generator()
        self.score          = model.evaluate(self.valid_generator)

    def save_score(self):
        score = {'loss' : self.score[0], 'accuracy' : self.score[1]}
        save_json(path = Path('score.json'), data = score)

try:
    config       = ConfigurationManager()
    val_config    = config.get_validation_config()
    evaluation    = Evaluation(val_config)
    evaluation.evaluation()
    evaluation.save_score()

except Exception as e:
    logging.exception(CustomException(e,sys))
```

6) Predict

```
import numpy as np

import os

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing import image


class PredictPipeline:

    def __init__(self, filename):

        self.filename = filename


    def predict(self):

        model = load_model(os.path.join('artifacts','training','model.h5'))


        imagename  = self.filename

        test_image = image.load_img(imagename, target_size = (224,224))

        test_image = image.img_to_array(test_image)


        test_image = np.expand_dims(test_image, axis = 0)


        result     = np.argmax(model.predict(test_image), axis=1)

        print(result)


        if result[0] == 1:
```

```
    predict = 'NOT SUFFERING FROM MONKEY POX'
    return [{'image ' : predict}]
else:
    predict = 'SUFFERING FROM MONKEY POX'
    return [{'image ' : predict}]
```

APP

```
from flask import Flask,request, jsonify, render_template
import os
from flask_cors import CORS, cross_origin
from poxVisionDetection.utils.common import decodeImage
from poxVisionDetection.pipeline.predict import PredictPipeline


os.putenv('LANG','en_US.UTF-8')
os.putenv("LC_ALL",'en_US.UTF-8')


app = Flask(__name__)
CORS(app)


class ClientApp:
    def __init__(self):
        self.filename     = 'inputImage.jpg'
        self.classifier   = PredictPipeline(self.filename)
```

```python
@app.route('/', methods = ['GET'])
@cross_origin()
def home():
    return render_template('index.html')


@app.route('/train', methods = ['GET','POST'])
@cross_origin()
def trainRoute():
    # os.system('python main.py')
    os.system('dvc repro')
    return 'training done successfully'


@app.route('/predict', methods = ['POST'])
@cross_origin()
def predictRoute():
    image = request.json['image']
    decodeImage(image,clApp.filename)
    result = clApp.classifier.predict()
    return jsonify(result)


if __name__ == '__main__':
    clApp = ClientApp()
```

```
app.run(host = '0.0.0.0', port = 8080)
```

13.1 Github Link

Gitbub link -> https://github.com/AarizZafar/poxVision_detection

Project demo link ->