**Mobile based Surveillance System:**

Dataset Conversion in the form of motion segmented Image:

import cv2

import numpy as np

from matplotlib import pyplot as plt

import os

import imghdr

if \_\_name\_\_ == '\_\_main\_\_':

    data\_dir = 'E:\comsat\Comsats\Semester7\Image processing\ProjectData'

    # image\_exts = ['jpeg', 'jpg', 'bmp', 'png']

    counter=0

    for image\_class in os.listdir(data\_dir):

        for image in os.listdir(os.path.join(data\_dir, image\_class)):

            image\_path = os.path.join(data\_dir, image\_class, image)

            original\_image = cv2.imread(image\_path)

            original\_image = cv2.cvtColor(original\_image, cv2.COLOR\_BGR2GRAY)

            # Apply median filtering

            original\_image = cv2.medianBlur(original\_image, 3)

            # Apply Canny edge detection

            edges\_canny = cv2.Canny(original\_image, 50, 150)

            # Apply morphological closing

            kernel = np.ones((7, 7), np.uint8)

            edges\_canny = cv2.morphologyEx(edges\_canny, cv2.MORPH\_CLOSE, kernel)

            # Fill holes in the image

            edgesCanny\_uint8 = edges\_canny.astype(np.uint8)

            contours, \_ = cv2.findContours(edgesCanny\_uint8, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

            filled\_image = np.zeros\_like(edgesCanny\_uint8)

            cv2.drawContours(filled\_image, contours, -1, 255, thickness=cv2.FILLED)

            path = "E:\comsat\Comsats\Semester7\Image processing\ProjectBlackandwhite"

            # Save the image using OpenCV

            cv2.imwrite(f"{path}/filename{counter}.png", filled\_image)

            counter+=1

            # Display the result

            # plt.imshow(filled\_image, cmap='gray')

            # plt.title('Image')

            # plt.show()

**Results:**

A close-up of a fire

Description automatically generatedInput Images:



Non-Fire:

A red and white cartoon

Description automatically generated

A white silhouette of a cartoon character

Description automatically generated

Trained Dataset Using CNN:

import os

import torch

import glob

import torch.nn as nn

from torchvision.transforms import transforms

from torch.utils.data import DataLoader

from torch.optim import Adam

from torch.optim import SGD

import torchvision

import matplotlib.pyplot as plt

from PIL import Image

def main():

    # Transforms

    transformer = transforms.Compose([

        transforms.Resize((200, 200)),

        transforms.ToTensor(),  # 0-255 to 0-1, numpy to tensors

        transforms.Normalize([0.5, 0.5, 0.5],  # 0-1 to [-1,1] , formula (x-mean)/std

                             [0.5, 0.5, 0.5])

    ])

    # Path for training and testing directory

    train\_path = 'E:\comsat\Comsats\Semester7\Image processing\ProjectBlackandwhite'

    train\_loader = DataLoader(

        torchvision.datasets.ImageFolder(train\_path, transform=transformer),

        batch\_size=16, shuffle=True

    )

    # train\_count = len(glob.glob(train\_path + '/\*\*/\*.jpg'))

    # learning\_rate\_arr = [1, 0.1, 0.01, 1, 0.1, 0.01]

    # optimizer\_arr = ['SGD', 'SGD', 'SGD', 'Adam', 'Adam', 'Adam']

    # batching\_arr = [False, True, False, False, True, False]

    learning\_rate\_arr = [0.0001]

    optimizer\_arr = ['Adam']

    batching\_arr = [ True]

    for i in range(len(learning\_rate\_arr)):

        model = ConvNet(num\_classes=2, batch=batching\_arr[i])

        if optimizer\_arr[i] == 'Adam':

            optimizer = Adam(model.parameters(), lr=learning\_rate\_arr[i])

        else:

            optimizer = SGD(model.parameters(), lr=learning\_rate\_arr[i])

        optimizer.zero\_grad()

        loss\_function = nn.CrossEntropyLoss()

        num\_epochs = 5

        epoch\_arr = [i for i in range(num\_epochs)]

        train\_acc\_arr = []

        loss\_arr = []

        for epoch in range(num\_epochs):

            train\_accuracy = 0.0

            train\_loss = 0.0

            model.train()

            for images, labels in train\_loader:

                optimizer.zero\_grad()

                outputs = model(images)

                loss = loss\_function(outputs, labels)

                loss.backward()

                optimizer.step()

                train\_loss += loss.item()

                per\_error, prediction = torch.max(outputs.data, 1)

                # print(labels.data)

                # print(prediction)

                # print((prediction == labels.data))

                # print(torch.sum(prediction == labels.data))

                train\_accuracy += int(torch.sum(prediction == labels.data))

                # print(train\_accuracy)

            # print(train\_count)

            train\_accuracy = train\_accuracy / 326

            train\_acc\_arr.append(train\_accuracy)

            loss\_arr.append(train\_loss)

            print('Epoch: ' + str(epoch) + ' Train Loss: ' + str(train\_loss) + ' Train Accuracy: ' + str(

                train\_accuracy))

        torch.save(model, train\_path+'\modelfirebatch16.pth')

        plt.plot(epoch\_arr, train\_acc\_arr, label='Train Accuracy')

        plt.xlabel('Epoch')

        plt.ylabel('Accuracy')

        plt.legend()

        plt.show()

        plt.plot(epoch\_arr, loss\_arr, label='Train error')

        plt.xlabel('Epoch')

        plt.ylabel('Loss')

        plt.legend()

        plt.show()

class ConvNet(nn.Module):

    def \_\_init\_\_(self, num\_classes=2, batch=False):

        super(ConvNet, self).\_\_init\_\_()

        self.isBatch = batch

        # Input shape= (32,3,200,200)

        self.conv1 = nn.Conv2d(in\_channels=3, out\_channels=12, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,12,200,200)

        self.bn1 = nn.BatchNorm2d(num\_features=12)

        # Shape= (32,12,200,200)

        self.relu1 = nn.ReLU()

        # Shape= (32,12,200,200)

        self.conv2 = nn.Conv2d(in\_channels=12, out\_channels=23, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,23,200,200)

        self.relu2 = nn.ReLU()

        # Shape= (32,23,200,200)

        self.pool1 = nn.MaxPool2d(kernel\_size=2)

        # Shape= (32,23,100,100)

        self.conv3 = nn.Conv2d(in\_channels=23, out\_channels=32, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,32,100,100)

        self.bn2 = nn.BatchNorm2d(num\_features=32)

        # Shape= (32,32,100,100)

        self.relu3 = nn.ReLU()

        # Shape= (32,32,100,100)

        self.conv4 = nn.Conv2d(in\_channels=32, out\_channels=20, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,20,100,100)

        self.bn3 = nn.BatchNorm2d(num\_features=20)

        # Shape= (32,20,100,100)

        self.relu4 = nn.ReLU()

        # Shape= (32,20,100,100)

        self.pool2 = nn.MaxPool2d(kernel\_size=2)

        # Shape= (32,20,50,50)

        self.fl = nn.Flatten()

        self.fc1 = nn.Linear(in\_features=50 \* 50 \* 20, out\_features=500)

        # self.dropout1 = nn.Dropout(0.5)

        self.fc2 = nn.Linear(in\_features=500, out\_features=100)

        # self.dropout2 = nn.Dropout(0.5)

        self.fc3 = nn.Linear(in\_features=100, out\_features=num\_classes)

        # Feed forwad function

    def forward(self, input):

        output = self.conv1(input)

        if self.isBatch:

            output = self.bn1(output)

        output = self.relu1(output)

        output = self.conv2(output)

        output = self.relu2(output)

        output = self.pool1(output)

        output = self.conv3(output)

        if self.isBatch:

            output = self.bn2(output)

        output = self.relu3(output)

        output = self.conv4(output)

        output = self.relu4(output)

        output = self.pool2(output)

        output = self.fl(output)

        output = self.fc1(output)

        # output = self.dropout1(output)

        output = self.fc2(output)

        # output = self.dropout2(output)

        output = self.fc3(output)

        return output

main()

Test Data Set for Prediction:

import torch

import os

import torch.nn as nn

from PIL import Image

from torch.optim import Adam

from torchvision.transforms import transforms

import torchvision

import matplotlib.pyplot as plt

def main():

    prediction=['fire','notfire']

    model = ConvNet(num\_classes=2, batch=True)

    train\_path = 'E:\comsat\Comsats\Semester7\Image processing\ProjectBlackandwhite'

    model = torch.load(train\_path+'\modelfire.pth')

    # with open(train\_path+'\modelfire.pth','rb') as f:

    #     model.load\_state\_dict(torch.load(f))

    img = Image.open('E:\comsat\Comsats\Semester7\Image processing\ProjectBlackandwhite\ilenametest0.png').resize((200,200))

    img = img.convert('RGB')

    img.show()

    image\_tensor = transforms.ToTensor()(img).unsqueeze(0)

        # Apply the transformation to convert the image to a tensor

    print(image\_tensor.shape)

    print(prediction[torch.argmax(model(image\_tensor))],'prediction')

class ConvNet(nn.Module):

    def \_\_init\_\_(self, num\_classes=2, batch=False):

        super(ConvNet, self).\_\_init\_\_()

        self.isBatch = batch

        # Input shape= (32,3,200,200)

        self.conv1 = nn.Conv2d(in\_channels=3, out\_channels=12, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,12,200,200)

        self.bn1 = nn.BatchNorm2d(num\_features=12)

        # Shape= (32,12,200,200)

        self.relu1 = nn.ReLU()

        # Shape= (32,12,200,200)

        self.conv2 = nn.Conv2d(in\_channels=12, out\_channels=23, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,23,200,200)

        self.relu2 = nn.ReLU()

        # Shape= (32,23,200,200)

        self.pool1 = nn.MaxPool2d(kernel\_size=2)

        # Shape= (32,23,100,100)

        self.conv3 = nn.Conv2d(in\_channels=23, out\_channels=32, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,32,100,100)

        self.bn2 = nn.BatchNorm2d(num\_features=32)

        # Shape= (32,32,100,100)

        self.relu3 = nn.ReLU()

        # Shape= (32,32,100,100)

        self.conv4 = nn.Conv2d(in\_channels=32, out\_channels=20, kernel\_size=3, stride=1, padding=1)

        # Shape= (32,20,100,100)

        self.bn3 = nn.BatchNorm2d(num\_features=20)

        # Shape= (32,20,100,100)

        self.relu4 = nn.ReLU()

        # Shape= (32,20,100,100)

        self.pool2 = nn.MaxPool2d(kernel\_size=2)

        # Shape= (32,20,50,50)

        self.fl = nn.Flatten()

        self.fc1 = nn.Linear(in\_features=50 \* 50 \* 20, out\_features=500)

        # self.dropout1 = nn.Dropout(0.5)

        self.fc2 = nn.Linear(in\_features=500, out\_features=100)

        # self.dropout2 = nn.Dropout(0.5)

        self.fc3 = nn.Linear(in\_features=100, out\_features=num\_classes)

        # Feed forwad function

    def forward(self, input):

        output = self.conv1(input)

        if self.isBatch:

            output = self.bn1(output)

        output = self.relu1(output)

        output = self.conv2(output)

        output = self.relu2(output)

        output = self.pool1(output)

        output = self.conv3(output)

        if self.isBatch:

            output = self.bn2(output)

        output = self.relu3(output)

        output = self.conv4(output)

        output = self.relu4(output)

        output = self.pool2(output)

        output = self.fl(output)

        output = self.fc1(output)

        # output = self.dropout1(output)

        output = self.fc2(output)

        # output = self.dropout2(output)

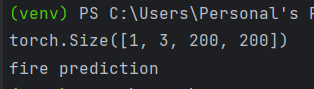
        output = self.fc3(output)

        return output

main()

Test Output:





React Native Camera Code:

import React, { useState, useEffect, useRef } from 'react';

import { StyleSheet, Text, View, TouchableOpacity } from 'react-native';

import { Camera } from 'expo-camera';

import { Image } from 'expo-image';

import \* as MediaLibrary from 'expo-media-library';

import \* as ImageManipulator from 'expo-image-manipulator';

export default function CameraCheck() {

  const [uri,setUri]=useState('')

  const blurhash =

  '|rF?hV%2WCj[ayj[a|j[az\_NaeWBj@ayfRayfQfQM{M|azj[azf6fQfQfQIpWXofj[ayj[j[fQayWCoeoeaya}j[ayfQa{oLj?j[WVj[ayayj[fQoff7azayj[ayj[j[ayofayayayj[fQj[ayayj[ayfjj[j[ayjuayj[';

  const [hasPermission, setHasPermission] = useState(null);

  const [type, setType] = useState(Camera.Constants.Type.back);

  const [isRecording, setIsRecording] = useState(false);

  const cameraRef = useRef(null);

  useEffect(() => {

    (async () => {

      const cameraStatus = await Camera.requestCameraPermissionsAsync();

      const audioStatus = await Camera.requestMicrophonePermissionsAsync();

      const mediaLibraryStatus = await MediaLibrary.requestPermissionsAsync();

      setHasPermission(

        cameraStatus.status === 'granted' &&

        audioStatus.status === 'granted' &&

        mediaLibraryStatus.status === 'granted'

      );

    })();

  }, []);

  if (hasPermission === null) {

    return <View />;

  }

  if (hasPermission === false) {

    return <Text>No access to camera</Text>;

  }

  const toggleCameraType = () => {

    setType(current => (current === Camera.Constants.Type.back ? Camera.Constants.Type.front : Camera.Constants.Type.back));

  };

  const saveMedia = async (uri) => {

    try {

      const asset = await MediaLibrary.createAssetAsync(uri);

      await MediaLibrary.createAlbumAsync('CameraApp', asset, false);

    } catch (error) {

      console.error('Error saving media to gallery:', error);

    }

  };

  const takePicture = async () => {

    if (cameraRef.current) {

      // const photo = await cameraRef.current.takePictureAsync();

      // console.log('Original URI:', photo.uri);

      //  console.log('1')

      // const manipulatedImage = await ImageManipulator.manipulateAsync(

      // photo.uri,

      // [{ resize: { width: 200, height: 200 } }],

      // { format: 'png', base64: true });

      //  console.log(manipulatedImage.uri)

      //  setUri(manipulatedImage.uri)

      // console.log('ok')

      console.log('http://192.168.233.13//getResponse.php')

      const response=await fetch('http://192.168.233.13/getResponse.php')

      const res=await response.json()

      console.log(res)

console.log('ok')

// let result = await ImagePicker.launchImageLibraryAsync({

//   mediaTypes: ImagePicker.MediaTypeOptions.All,

//   allowsEditing: true,

//   base64: true, //<-- boolean base64

//   aspect: [4, 3],

//   quality: 1,

// });

// console.log(result);

      // const image = Asset.fromModule(require(manipulatedImage.uri));

      //  console.log('1')

      // await image.downloadAsync();

      //  console.log('1')

      // console.log(image)

      // console.log(photo)

      // saveMedia(photo.uri);

      // console.log('1')

    //   const manipulatedImage = await ImageManipulator.manipulateAsync(

    //   photo.uri,

    //   [],

    //   { format: 'png', base64: true } // Set the format and base64 to true

    // );

console.log('2')

    // The manipulatedImage.uri now contains the matrix representation of the image

    // You can access the base64 data directly or convert it to a Uint8Array

    // const base64Data = manipulatedImage.base64;

    // const uint8Array = new Uint8Array(Buffer.from(base64Data, 'base64'));

console.log('3')

    // Now you can use the uint8Array as needed

    // Optionally, you can log or process the image data

    // console.log('Base64 Data:', base64Data);

    // console.log('Uint8Array:', uint8Array);

    // If you need the original URI for any reason, you can still access it from photo.uri

    // console.log('Original URI:', photo.uri);

    }

  };

  const handleRecord = async () => {

    if (isRecording) {

      cameraRef.current.stopRecording();

    } else {

      if (cameraRef.current) {

        try {

          setIsRecording(true);

          const video = await cameraRef.current.recordAsync();

          setIsRecording(false);

          saveMedia(video.uri);

        } catch (error) {

          console.error("Error during video recording:", error);

          setIsRecording(false);

        }

      }

    }

  };

  return (

    <View style={styles.container}>

      <Camera style={styles.camera} type={type} ref={cameraRef}>

        <View style={styles.buttonContainer}>

          <TouchableOpacity style={styles.button} onPress={toggleCameraType}>

            <Text style={styles.text}>Flip</Text>

          </TouchableOpacity>

          <TouchableOpacity style={styles.button} onPress={takePicture}>

            <Text style={styles.text}>Take Photo</Text>

          </TouchableOpacity>

          <TouchableOpacity style={styles.button} onPress={handleRecord}>

            <Text style={styles.text}>{isRecording ? 'Stop Recording' : 'Start Recording'}</Text>

          </TouchableOpacity>

        </View>

      </Camera>

      <Image

        style={{flex:2}}

        source={uri}

        placeholder={blurhash}

        contentFit="cover"

        transition={1000}

      />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    justifyContent: 'center',

  },

  camera: {

    flex: 1,

  },

  buttonContainer: {

    flex: 1,

    flexDirection: 'row',

    margin: 20,

    backgroundColor: 'transparent',

    justifyContent: 'center',

    alignItems: 'flex-end',

  },

  button: {

    padding: 13,

    backgroundColor: 'gray',

    margin: 10,

  },

  text: {

    fontSize: 18,

    color: 'white',

  },

});

React Native Code for Ringtone:

import React,{useState,useEffect} from 'react';

import { View, Button } from 'react-native';

import { Audio } from 'expo-av';

const RingtonePlayer = () => {

  const [ss,Setss]=useState(true)

    useEffect(() => {

      if(ss){

    const intervalId = setInterval(playRingtone, 4000);

    return () => clearInterval(intervalId);

    }

  }, [ss]);

  const playRingtone = async () => {

    const { sound } = await Audio.Sound.createAsync(

      require('./assets/ring1.mp3')

    );

    await sound.playAsync();

  };

  return (

    <View style={{marginTop:200}}>

      <Button title="Play Ringtone" onPress={()=>{

        Setss(true);

        playRingtone()}} />

      <Button title="Stop" onPress={()=>{

        console.log('false')

        Setss(false)}} />

    </View>

  );

};

export default RingtonePlayer;