In [ ]:
```
import cv2
import torch
import torch.nn as nn
from torchvision import transforms
import torch
import cv2
import torch.nn as nn
from torchvision import models, transforms
from PIL import Image

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define class names and create a class-to-index mapping dictionary
class_names = ['Wheel Mark on road', 'Longitudinal crack', 'Lateral Equal Interval
class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}

# Load the model
model = models.resnet18(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model.load_state_dict(torch.load('F:/venkatesh/Resnet/model_8.pth', map_location=t
model.eval()

# Define the transform for the input image
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
# Load the input image
image = cv2.imread('F:/venkatesh/images/longitudinal construction joint.jpeg')

# Convert the image to PyTorch tensor
tensor = transform(image)
tensor = tensor.unsqueeze(0)
tensor = tensor.to(device)

# Pass the tensor through the model
with torch.no_grad():
    output = model(tensor)
    _, preds = torch.topk(output, k=5, dim=1)  # Get top 3 predictions
    class_indices = preds.squeeze().tolist()
    class_names = [class_names[idx] for idx in class_indices]

# Draw the predicted class names on the frame
y_offset = 50
for class_name in class_names:
    print(class_name)
    cv2.putText(image, class_name, (50, y_offset), cv2.FONT_HERSHEY_SIMPLEX, 1, (0
    y_offset += 50

# Display the image
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [ ]:
```
#!pip install geopy
!pip uninstall geocoder
!pip install geocoder
import geocoder
```

```
# Get current location
location = geocoder.ip('me')

print("Latitude:", location.lat)
print("Longitude:", location.lng)
print("City:", location.city)
print("State:", location.state)
print("Country:", location.country)
print("Postal code:", location.postal)
```

In [ ]:

```
# detection on video and ipwebcam with  storing geo location
import cv2
import torch
import requests
import json
import csv
import torch.nn as nn
from torchvision import transforms
import torch
import cv2
import torch.nn as nn
from torchvision import models, transforms
from PIL import Image
import geocoder

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define class names and create a class-to-index mapping dictionary
class_names = ['Wheel Mark on road', 'Longitudinal crack', 'Lateral Equal Interval
class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}

# Load the model
model = models.resnet18(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model.load_state_dict(torch.load('F:/venkatesh/Resnet/model_8.pth', map_location=d
model.eval()

# Define the transform for the input image
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Open the video file
cap = cv2.VideoCapture('F:/venkatesh/videos/pothole3.mp4')
#cap = cv2.VideoCapture('http://192.168.71.20:8080/video')
#cap = cv2.VideoCapture(0)


# Get the video properties
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

# Create a video writer
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('F:/venkatesh/videos/output.mp4', fourcc, fps, (width, heigh
# Initialize frame counter
```

```
frame_count = 0

# Initialize the CSV writer
with open('F:/venkatesh/output_location_data.csv', mode='w', newline='') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINI

    # Write header row
    writer.writerow(['Damage', 'Latitude', 'Longitude','City','state','country','p
    detected_class_text=""
    # Loop through the frames
    while True:
        # Read the frame
        ret, frame = cap.read()

        # Check if the frame was successfully read
        if not ret:
            break
        # Only process every other frame
        y_offset = 50
        if frame_count % 5 == 0:


            # Convert the frame to PyTorch tensor
            tensor = transform(frame)
            tensor = tensor.unsqueeze(0)
            tensor = tensor.to(device)
            # Pass the tensor through the model
            with torch.no_grad():
                output = model(tensor)
                _, preds = torch.topk(output, k=3, dim=1)  # Get top 3 predictions
                class_indices = preds.squeeze().tolist()
                class_names_1 = [class_names[idx] for idx in class_indices]



            for class_idx, class_name in zip(class_indices, class_names_1):
                confidence = torch.nn.functional.softmax(output, dim=1)[0][class_i
                text = f"{class_name}: {confidence:.2f}"
                # Get current location
                location = geocoder.ip('me')
                # Print latitude and longitude

                print("type of damage : {:35}".format(class_name),end=" ")
                if location is not None :
                    # Print latitude and longitude
                    latitude=location.lat
                    longitude=location.lng
                    print("Latitude : ", latitude,end=" ")
                    print("Longitude : ", longitude,end=" ")
                    # Write the row to the CSV file
                    writer.writerow([class_name, latitude, longitude,location.city

                else:
                    print("Latitude : ", "can't fetch",end=" ")
                    print("Longitude : ", "can't fetch",end=" ")
                    # Write the row to the CSV file
                    writer.writerow([class_name,  "can't fetch",  "can't fetch", "
                print()
                #text = class_name
                cv2.putText(frame, text, (50, y_offset), cv2.FONT_HERSHEY_SIMPLEX,
                y_offset += 50
                # Write the frame to the output video
                out.write(frame)
```

```
        # Display the frame
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the resources
    cap.release()
    out.release()
    cv2.destroyAllWindows()
```

In [ ]:

```
# detection on video and ipwebcam with out storing geo location
import cv2
import torch
import requests
import json
import csv
import torch.nn as nn
from torchvision import transforms
import torch
import cv2
import torch.nn as nn
from torchvision import models, transforms
from PIL import Image
import geocoder

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define class names and create a class-to-index mapping dictionary
class_names = ['Wheel Mark on road', 'Longitudinal crack', 'Lateral Equal Interval
class_to_idx = {class_name: i for i, class_name in enumerate(class_names)}

# Load the model
model = models.resnet18(pretrained=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model.load_state_dict(torch.load('F:/venkatesh/Resnet/model_8.pth', map_location=d
model.eval()

# Define the transform for the input image
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Open the video file
cap = cv2.VideoCapture('F:/venkatesh/videos/pothole8.mp4')
#cap = cv2.VideoCapture('http://192.168.71.20:8080/video')
#cap = cv2.VideoCapture(0)


# Get the video properties
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

# Create a video writer
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('F:/venkatesh/videos/output.mp4', fourcc, fps, (width, heigh
# Initialize frame counter
frame_count = 0
```

```
# Initialize the CSV writer
with open('F:/venkatesh/output_location_data.csv', mode='w', newline='') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINI

    # Write header row
    writer.writerow(['Damage', 'Latitude', 'Longitude'])
    detected_class_text=""
    # Loop through the frames
    while True:
        # Read the frame
        ret, frame = cap.read()

        # Check if the frame was successfully read
        if not ret:
            break
        # Only process every other frame
        y_offset = 50
        if frame_count % 5 == 0:


            # Convert the frame to PyTorch tensor
            tensor = transform(frame)
            tensor = tensor.unsqueeze(0)
            tensor = tensor.to(device)

            # Pass the tensor through the model
            with torch.no_grad():
                output = model(tensor)
                _, preds = torch.topk(output, k=3, dim=1)  # Get top 3 predictions
                class_indices = preds.squeeze().tolist()
                class_names_1 = [class_names[idx] for idx in class_indices]



            detected_class_list=[]
            for class_idx, class_name in zip(class_indices, class_names_1):
                confidence = torch.nn.functional.softmax(output, dim=1)[0][class_i
                text = f"{class_name}: {confidence:.2f}"
                #text = class_name


                if confidence>0.02:
                    detected_class_list.append(text)
                    cv2.putText(frame, text, (50, y_offset), cv2.FONT_HERSHEY_SIMP
                    y_offset += 50


        else:
            for detected_class_text in detected_class_list:
                cv2.putText(frame, detected_class_text, (50, y_offset), cv2.FONT_H
                y_offset += 50

        # Increment the frame counter
        frame_count += 1
        #text = class_name
        #y_offset += 50
        # Write the frame to the output video
        out.write(frame)

        # Display the frame
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
    # Release the resources
    cap.release()
    out.release()
    cv2.destroyAllWindows()
```