Automatic Classification of Road Damage

Name and surname : Assyl Kazhiakhmetova
Group : BDA - 2202

Final Model:
https://road-damage-dngomvd2weytavjugm9mvp.streamlit.app/

GitHub:
https://github.com/assylkzh/Road-Damage.git

Project overview

The aim of the "**Automatic Classification of Road Damage"** project is to classify images of road damages into four categories based on the extent of the damage. The categories are:
Good: No visible damage.
Poor: Noticeable damage, but not severe.
Satisfactory: Moderate damage, requiring attention.
Very Poor: Severe damage, very likely to be hazardous.
The dataset used is the "Road Damage Classification and Assessment" dataset from Kaggle platform, which contains images of roads with different levels of damage. The baseline model used is - **ResNet50**

Dataset Details

Road damage classification on Kaggle
https://www.kaggle.com/datasets/prudhvignv/road-damage-classification-and-assessment
This dataset contains images of roads with damages. The damages of roads are classified into 4 categories such as good, poor, satisfactory, very poor according to their extent of damage.
Overall, the dataset contains 2074 images across all 4 categories.
the dataset folder structure is as follows:

```
data/
│       ├── good/
│       ├── poor/
│       ├── satisfactory/
│       └── very_poor/
```

Classification model

Firstly, the dataset was downloaded from the Kaggle platform.

Automatic Detection of Road Damage - Classification Project

Classify road damage into four categories: good, poor, satisfactory, and very poor using images. By that this classification model can help in prioritizing road maintenance.

```
import kagglehub

# Download dataset from the kaggle
path = kagglehub.dataset_download("prudhvignv/road-damage-classification-and-assessment")

print("Path to dataset files:", path)
```

```
Path to dataset files: /Users/asylkaziahmetova/.cache/kagglehub/datasets/prudhvignv/road-damage-classification-and-assessment/versions/1
```

```
# path to the dataset
DATASET_PATH = "/Users/asylkaziahmetova/.cache/kagglehub/datasets/prudhvignv/road-damage-classification-and-assessment/versions/1"
```

Then, the required packages were installed and libraries were imported.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader
import seaborn as sns
```

Explanation of libraries used:

- os for interaction with the operating system
- numpy for numerical computing
- matplotlib.pyplot for plotting
- scikit-learn is used for splitting a dataset into training and testing sets, evaluation of the models using precision, recall, F1-score, and support
- torch for Deep learning, the enhancement of the CNN model architecture
- seaborn for data visualization

The parameters of the classification models

```
# Parameters of the models
BATCH_SIZE = 32
IMAGE_SIZE = 224
NUM_CLASSES = 4
EPOCHS = 10
LEARNING_RATE = 0.001
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Batch size of 32
Image size of 224x224 pixels
Number of classes is 4 (good, poor, satisfactory, very_poor)
Number of epochs are 10
Learning rate of 0.001
Devise is for selecting the GPU or CPU in case of non-existence of GPU

Splitting the dataset into train, test, validation sets (by 70,15,15%)

```
from torch.utils.data import random_split
from torchvision.datasets import ImageFolder

# Load the full dataset with transformations
dataset = ImageFolder(root=os.path.join(DATASET_PATH, "sih_road_dataset"), transform=train_transforms)
```

The dataset was splitted into train, test, validation sets by 70%,15%,15%, respectively. for that the dataset were loaded from the folder and the specific transformation(data augmentation) were applied to generalize and preprocess the data images.

```
# Data transformations
train_transforms = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Explanation:
Resize - to make the images fixed size , they were resized into 224x224 pixels size
RandomHorizontalFlip() - randomly fl~ipping the images horizontally with a probability of 50%.
RandomRotation(15) - randomly rotating the images within a range of ±15 degrees.
ColorJitter(brightness=0.2, contrast=0.2) - adjusting the brightness and contrast of the image.
ToTensor() - converting to Tensors.
Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) - normalizing the image's pixel values using the mean and standard deviation for each channel (RGB) (Means: [0.485, 0.456, 0.406] and Standard Deviations: [0.229, 0.224, 0.225]).
The sizes of train, test, validation sets were set as 70, 15, 15 percents. Then, using the random_split function of torch.utils library the dataset was splitted.

```
# Get the size of the dataset
dataset_size = len(dataset)
train_size = int(0.7 * dataset_size)  # 70% for training
val_size = int(0.15 * dataset_size)  # 15% for validation
test_size = dataset_size - train_size - val_size  # 15% for testing

# Split dataset into train, validation, and test sets
train_data, val_data, test_data = random_split(dataset, [train_size, val_size, test_size])

print(f"Dataset sizes - Train: {len(train_data)}, Validation: {len(val_data)}, Test: {len(test_data)}")
```

The sizes of the sets:

```
Dataset sizes - Train: 1451, Validation: 311, Test: 312
```

After, the similar specific transformations were applied into test and validation sets of the data.

```
# Apply specific transforms for validation and test data
val_data.dataset.transform = val_transforms
test_data.dataset.transform = val_transforms
```

Smilarly, all the images were resized into 224x224 pixel images, converted to tensors and normalized using mean and standard deviation.

```
val_transforms = transforms.Compose([
    transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

The classes of the datasets:

```
# get the classes' names
class_names = dataset.classes
print("Classes:", class_names)
```

```
Classes: ['good', 'poor', 'satisfactory', 'very_poor']
```

Baseline Model - ResNet50

ResNet50 is a powerful convolutional neural network (CNN) architecture which was pre-trained on the ImageNet dataset. The dataset contains millions of labeled images across 1,000 classes. ResNet50 has a deep network with 50 layers, and introduces residual connections (input of a layer is added to its output), and mainly, ResNet50 is designed for multi-class classification tasks.

For that, the model ResNet50 was loaded then, the Fully Connected (FC) Layer of it was updated for 4 classes of road damages as the original model has 1,000-class Fully Connected layer.

```
# Load a pretrained model resnet50
model = models.resnet50(pretrained=True)

model.fc = nn.Linear(model.fc.in_features, NUM_CLASSES)

model = model.to(DEVICE)
```

The loss function and optimizer initialization:

```
# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

The loss function is simple Cross-Entropy Loss, and the optimizer id Adam with a learning rate of 0.001. The cross entropy loss function measures the difference between the model's predicted probability and the true values. and Adam optimizer updates the model's parameters (weights and biases) during training to minimize the loss function.

ResNet50 Model training

```python
def train_one_epoch(model, dataloader, optimizer, criterion):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in dataloader:
        images, labels = images.to(DEVICE), labels.to(DEVICE)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        correct += predicted.eq(labels).sum().item()
        total += labels.size(0)

    accuracy = 100 * correct / total
    return running_loss / len(dataloader), accuracy

def validate_one_epoch(model, dataloader, criterion):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in dataloader:
            images, labels = images.to(DEVICE), labels.to(DEVICE)
            outputs = model(images)
            loss = criterion(outputs, labels)

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            correct += predicted.eq(labels).sum().item()
            total += labels.size(0)

    accuracy = 100 * correct / total
    return running_loss / len(dataloader), accuracy
```

The train, validation_one_epoch function are training and evaluation functions. Firstly, set the model into train()/eval() modes, respectively.
Train function performs the forward pass, computes the loss, backpropagates to calculate gradients, and updates the model's parameters. Computes and returns average training loss and accuracy for the epoch.
Evaluation(val) function performs only the forward pass (without gradient computation), calculates the validation loss and accuracy.
Both of them, as mentioned, compute loss and accuracy of each epoch and return average loss and accuracy.

```
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(EPOCHS):
    train_loss, train_acc = train_one_epoch(model, train_loader, optimizer, criterion)
    val_loss, val_acc = validate_one_epoch(model, val_loader, criterion)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)

    print(f"Epoch {epoch+1}/{EPOCHS}")
    print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_acc:.2f}%")
    print(f"Val Loss: {val_loss:.4f}, Val Accuracy: {val_acc:.2f}%")
```

```
Epoch 1/10
Train Loss: 0.3312, Train Accuracy: 88.77%
Val Loss: 6.1680, Val Accuracy: 65.92%
Epoch 2/10
Train Loss: 0.2090, Train Accuracy: 93.11%
Val Loss: 0.1773, Val Accuracy: 93.57%
Epoch 3/10
Train Loss: 0.2114, Train Accuracy: 92.42%
Val Loss: 0.3126, Val Accuracy: 86.50%
```

Then, the model is trained for 10 epochs and however, stopped in 3d epoch when the results were as follows:

Epoch 1/10
Train Loss: 0.3312, Train Accuracy: 88.77%
Val Loss: 6.1680, Val Accuracy: 65.92%
Epoch 2/10
Train Loss: 0.2090, Train Accuracy: 93.11%
Val Loss: 0.1773, Val Accuracy: 93.57%
Epoch 3/10
Train Loss: 0.2114, Train Accuracy: 92.42%
Val Loss: 0.3126, Val Accuracy: 86.50%

Base Model evaluation

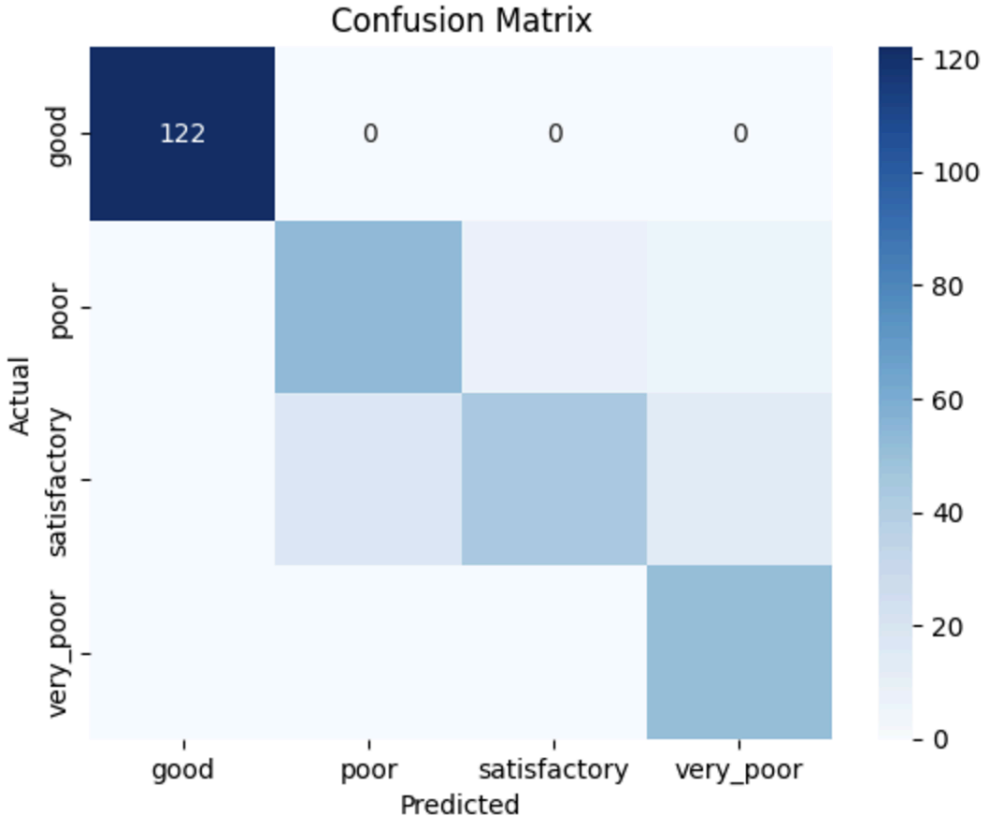
The testing of the model's performance on the test dataset, generating a detailed classification report, and visualizing the results using a confusion matrix heatmap.

```
def evaluate_model(model, dataloader):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in dataloader:
            images, labels = images.to(DEVICE), labels.to(DEVICE)
            outputs = model(images)
            _, preds = outputs.max(1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_preds), np.array(all_labels)

# Evaluate
test_preds, test_labels = evaluate_model(model, test_loader)

# Classification Report
print(classification_report(test_labels, test_preds, target_names=class_names))

# Confusion Matrix
cm = confusion_matrix(test_labels, test_preds)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

The results:

	precision	recall	f1-score	support
good	1.00	1.00	1.00	122
poor	0.77	0.80	0.79	66
satisfactory	0.86	0.60	0.71	73
very_poor	0.73	1.00	0.84	51
accuracy			0.87	312
macro avg	0.84	0.85	0.83	312
weighted avg	0.87	0.87	0.86	312

- 87% of all samples were classified correctly.
- The model performs well overall but has room for improvement, especially in classes like "Satisfactory" and "Poor."
- Recall for "Satisfactory" is notably low, suggesting the model might need better representation or augmentation for this class.



In the plot, it can be seen that the poor and satisfactory classes are misclassified in some cases.
Then, the baseline model was saved as model1

```
:  torch.save(model.state_dict(), "road_damage_model1.pth")
   print("Model saved as road_damage_model1.pth")
```

```
Model saved as road_damage_model1.pth
```

Enhanced Model

Firstly, the pretrained layers were frozen to avoid updating their weights during training.
To enhance The baseline ResNet50 model the Fully Connected Layer was modified:

- Dimensionality Reduction: The fc layer output is reduced from its original size to 512 neurons.
- ReLU Activation function were added
- Batch Normalization were added
- Dropout (40%): to prevent overfitting , the random dropout of 40% of neurons were added.

```
class EnhancedResNet50(nn.Module):
    def __init__(self, num_classes):
        super(EnhancedResNet50, self).__init__()
        self.base_model = models.resnet50(pretrained=True)

        # Freeze the base model layers if required
        for param in self.base_model.parameters():
            param.requires_grad = False

        # Modify the final layers
        self.base_model.fc = nn.Sequential(
            nn.Linear(self.base_model.fc.in_features, 512),  # Reduce dimensionality
            nn.ReLU(),
            nn.BatchNorm1d(512),  # Batch Normalization
            nn.Dropout(0.4),      # Dropout with 40% probability
            nn.Linear(512, num_classes)  # Output layer
        )

    def forward(self, x):
        return self.base_model(x)
```

Training and Validation of the model

The loss function and optimizer were set as in the baseline model: Cross entropy and Adam optimizer of learning rate 0.001. Then the model was trained and evaluated using above train and validation functions.

```
enhanced_model = EnhancedResNet50(num_classes=NUM_CLASSES).to(DEVICE)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(enhanced_model.parameters(), lr=LEARNING_RATE)

# Training and Evaluation
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(EPOCHS):
    train_loss, train_acc = train_one_epoch(enhanced_model, train_loader, optimizer, criterion)
    val_loss, val_acc = validate_one_epoch(enhanced_model, val_loader, criterion)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)

    print(f"Epoch {epoch+1}/{EPOCHS}")
    print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_acc:.2f}%")
    print(f"Val Loss: {val_loss:.4f}, Val Accuracy: {val_acc:.2f}%")
```

Training was stopped at 4th epoch at this results:

Epoch 1/10
Train Loss: 0.2561, Train Accuracy: 90.70%
Val Loss: 0.1725, Val Accuracy: 92.60%
Epoch 2/10
Train Loss: 0.1243, Train Accuracy: 95.86%
Val Loss: 0.1195, Val Accuracy: 96.14%
Epoch 3/10
Train Loss: 0.1039, Train Accuracy: 96.49%
Val Loss: 0.1191, Val Accuracy: 97.11%
Epoch 4/10
Train Loss: 0.0971, Train Accuracy: 96.76%
Val Loss: 0.1160, Val Accuracy: 96.46%

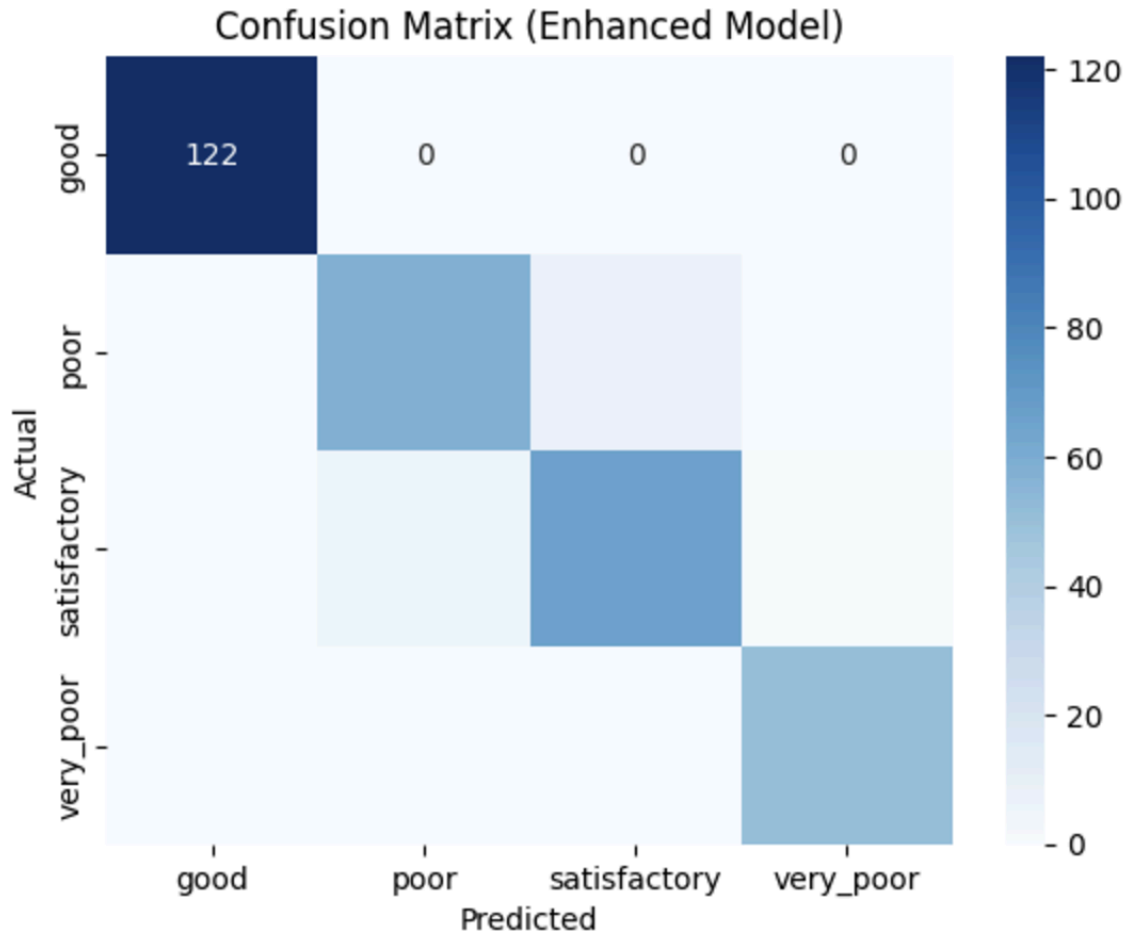
Evaluation
The enhanced model ,then, was evaluated on a test set, using similar metrics and visualizations.

```
# Evaluate the Enhanced Model
test_preds, test_labels = evaluate_model(enhanced_model, test_loader)

# Classification Report
print(classification_report(test_labels, test_preds, target_names=class_names))

# Confusion Matrix
cm = confusion_matrix(test_labels, test_preds)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (Enhanced Model)")
plt.show()
```

The results:

	precision	recall	f1-score	support
good	1.00	1.00	1.00	122
poor	0.91	0.88	0.89	66
satisfactory	0.89	0.90	0.90	73
very_poor	0.98	1.00	0.99	51
accuracy			0.95	312
macro avg	0.94	0.95	0.95	312
weighted avg	0.95	0.95	0.95	312

- The model correctly predicted the class labels for 95% of the test samples.
- The "Good" and "Very Poor" classes perform almost perfectly.



Finally, the enhanced model was saved model2:

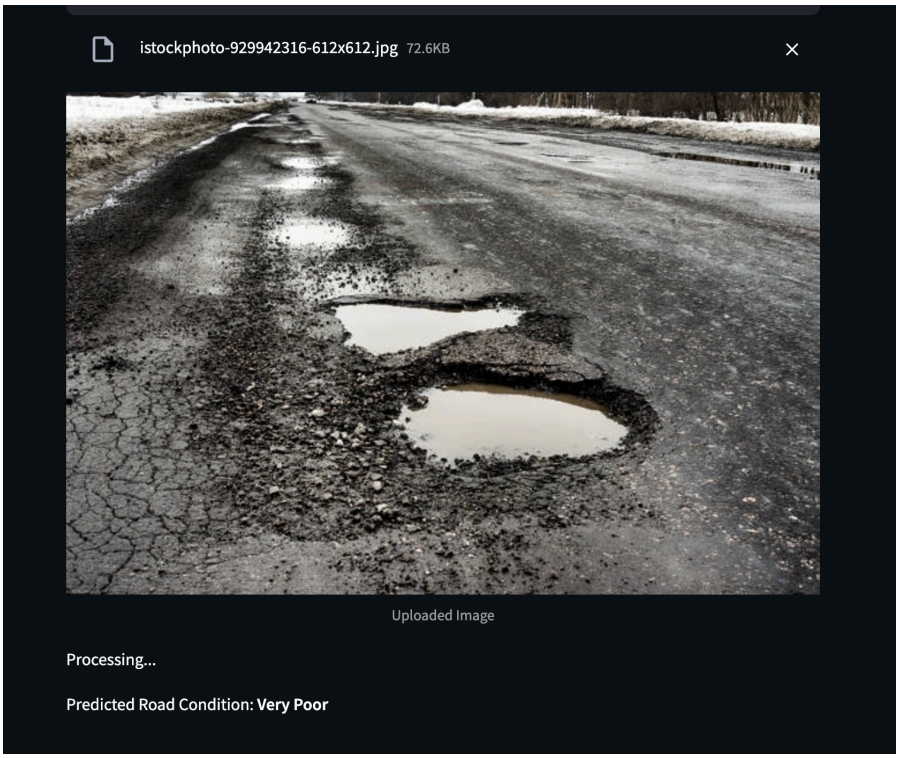
```
torch.save(model.state_dict(), "road_damage_model2.pth")
print("Model saved as road_damage_model2.pth")
```

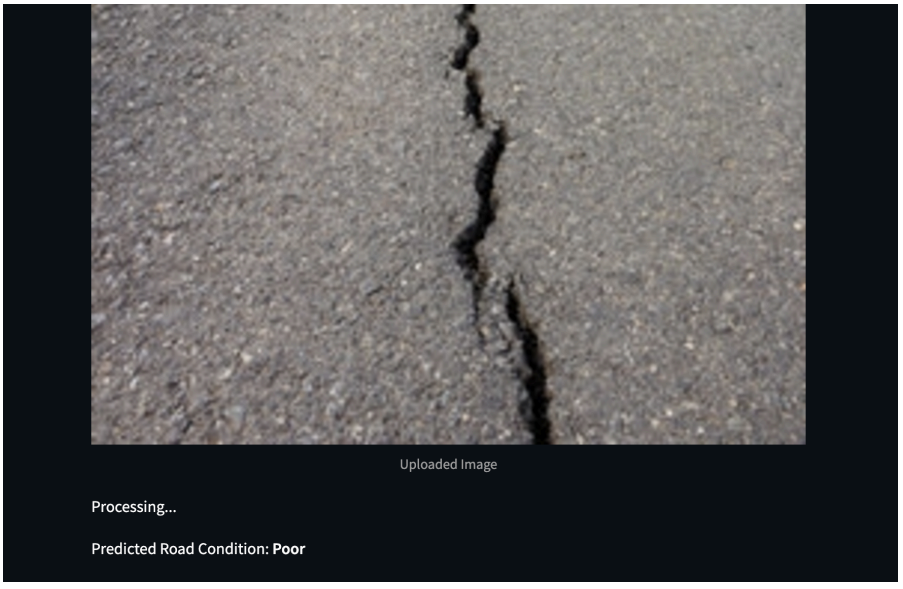
```
Model saved as road_damage_model2.pth
```

Comparison of the models:

- The enhanced model's accuracy of 95% is a notable improvement over the baseline's 87%.
- "Poor" class' Precision improves by 14% in enhanced model
- In "Satisfactory" class recall improves from 60% to 90%, which indicates a reduction in misclassification.
- For "Very Poor" class Precision improves by 25%, and F1-score reaches 99%.

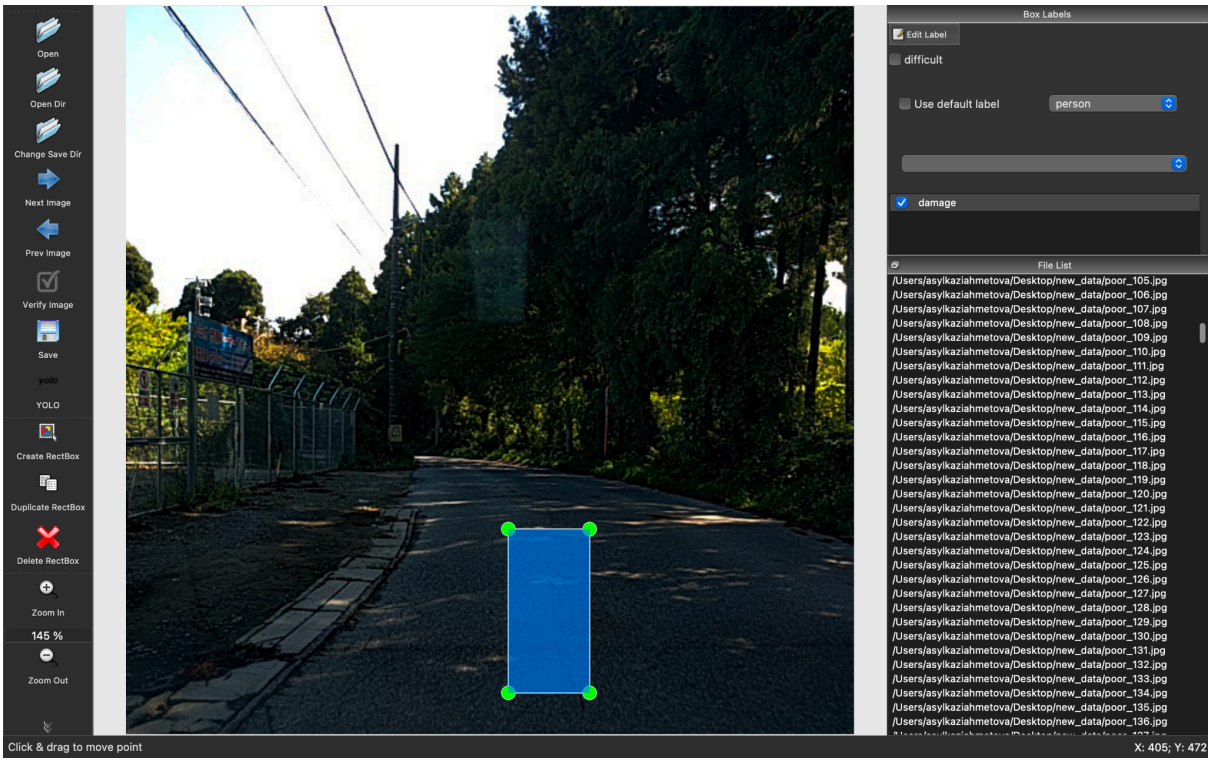
Final Model:
https://road-damage-dngomvd2weytavjugm9mvp.streamlit.app/
Upload an image and got the classification result:

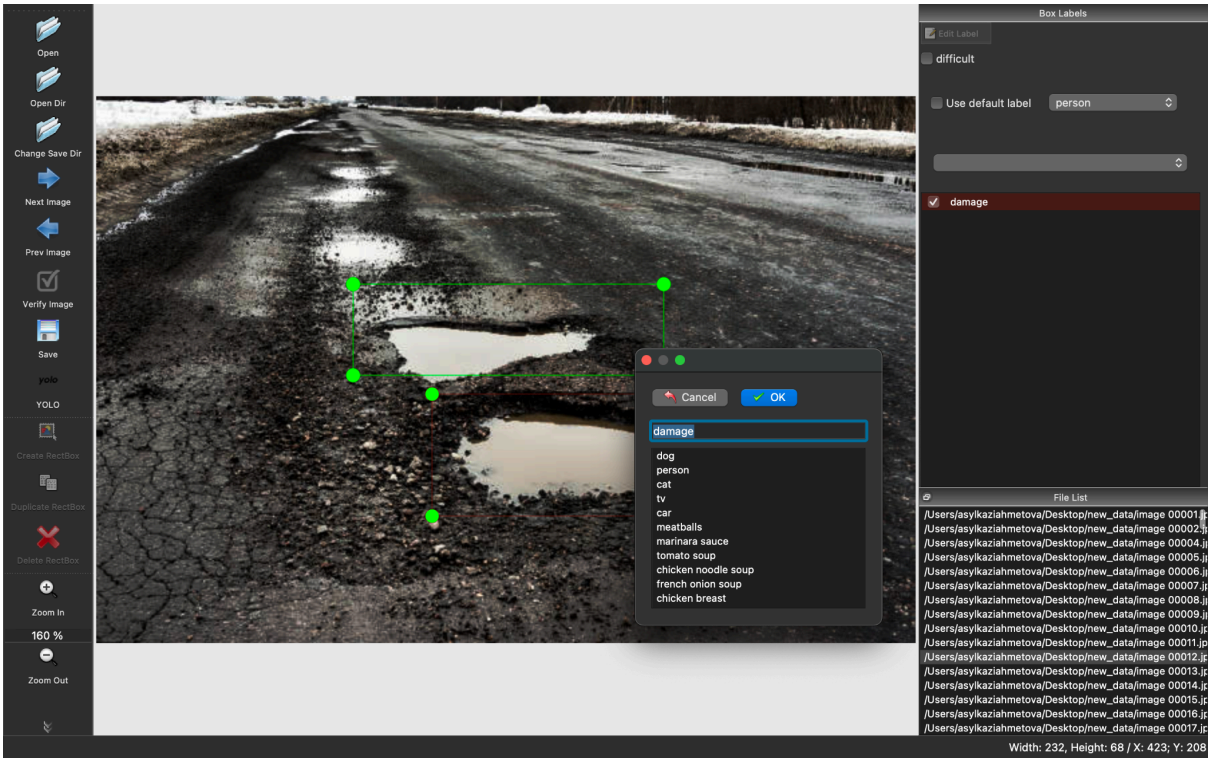


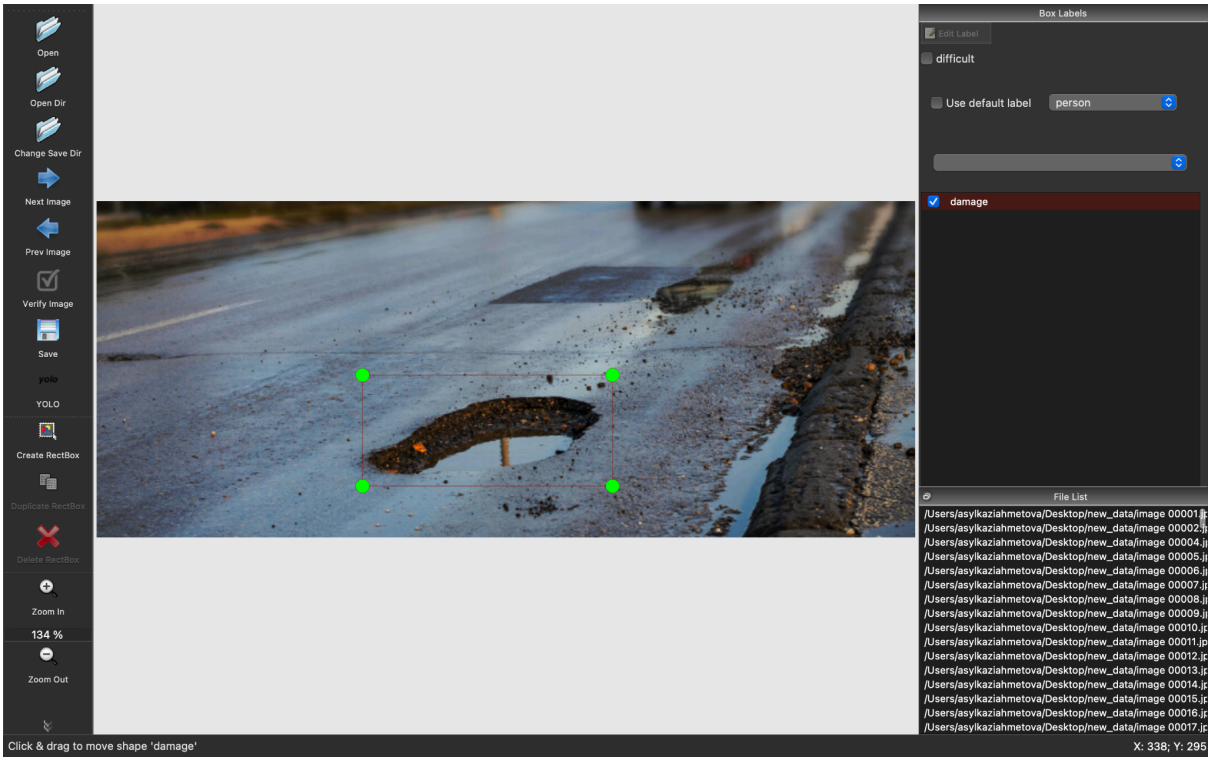


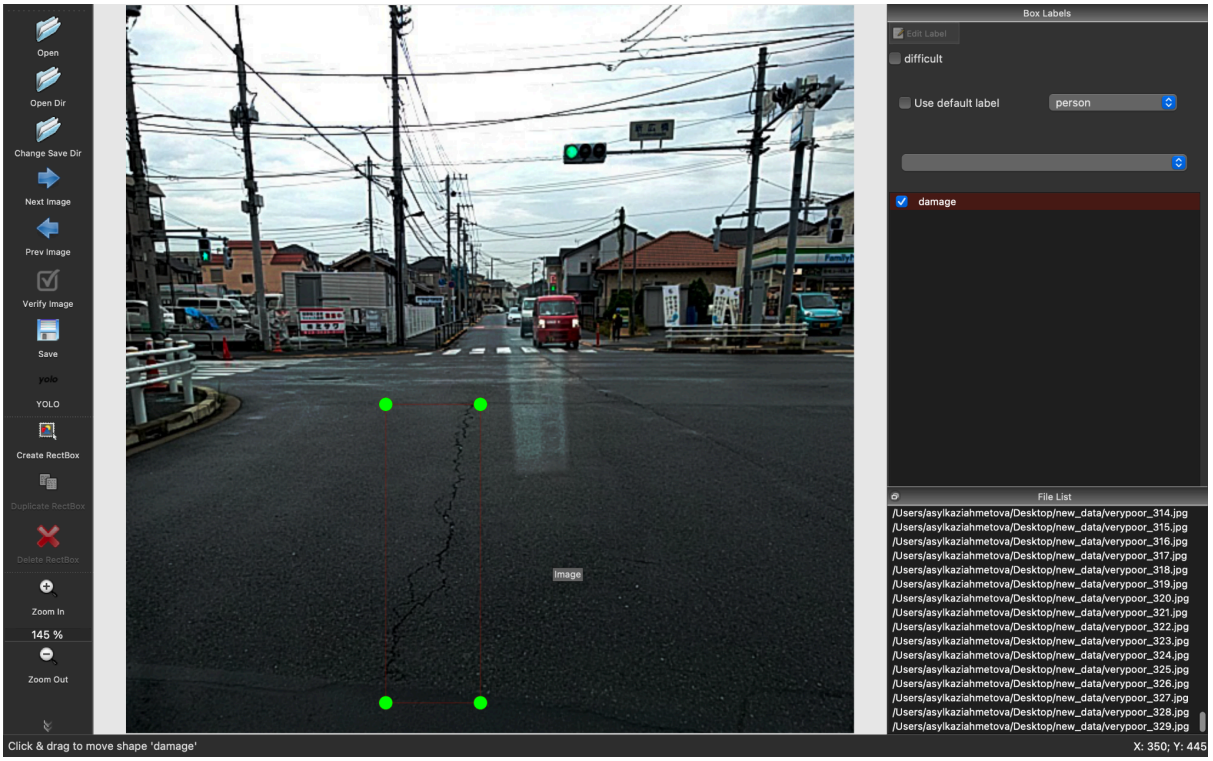
Data labelling using Labelimg

Additionally, all the data were labelled using Labelimg tool. overall, 891 images were labelled (labelling the images where the roads have a damage such as hole or a crack).






Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
YOLO
Create RectBox
Duplicate RectBox
Delete RectBox
Zoom In
134 %
Zoom Out
Box Labels
Edit Label
difficult
Use default label
person
damage
File List
/Users/asylkaziahmetova/Desktop/new_data/image 00001.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00002.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00004.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00005.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00006.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00007.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00008.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00009.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00010.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00011.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00012.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00013.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00014.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00015.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00016.jp
/Users/asylkaziahmetova/Desktop/new_data/image 00017.jp
Click & drag to move shape 'damage'
X: 338; Y: 295


Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
YOLO
Create RectBox
Duplicate RectBox
Delete RectBox
Zoom In
145 %
Zoom Out
Box Labels
Edit Label
difficult
Use default label
person
damage
File List
/Users/asylkaziahmetova/Desktop/new_data/verypoor_314.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_315.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_316.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_317.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_318.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_319.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_320.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_321.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_322.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_323.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_324.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_325.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_326.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_327.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_328.jpg
/Users/asylkaziahmetova/Desktop/new_data/verypoor_329.jpg
Image
Click & drag to move shape 'damage'
X: 350; Y: 445