```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
from openpyxl.styles.builtins import output
from torch.nn import CrossEntropyLoss
from torch.onnx.symbolic_opset9 import permute
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from torchvision import datasets
from torchvision.transforms import ToTensor
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import torchvision

import torchvision.transforms as transforms
```

Check device

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

```
device(type='cuda')
```

Extract file from resources

```
# prompt: write the code to extract the Rescources file

import zipfile

# Assuming 'Resources.zip' is in the current working directory
# Replace with the actual path if it's different
with zipfile.ZipFile('/content/Resources_ModelTraining_CNN.zip', 'r') as zip_ref:
    zip_ref.extractall('extracted_resources') # Extracts to a folder called extracted_resources
```

Create transform with many parameters which help to serve for many cases

```
transformer = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

dataset = datasets.ImageFolder(root='/content/extracted_resources/dataset', transform=transformer)
```

Get split data

```
train_size = int(0.75*len(dataset))
test_size = len(dataset) - train_size
```

```
train_dataset, val_dataset = random_split(dataset, [train_size, test_size])
```

Create train loader

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

Expolore dataset - train loader

```
for images, labels in train_loader:
    print(images.shape)
    print(labels.shape)
    break
```

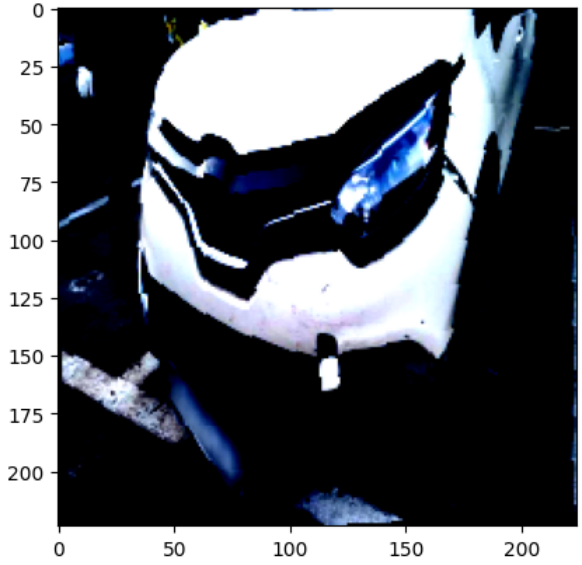
```
torch.Size([32, 3, 224, 224])
torch.Size([32])
```

```
class_dataset =dataset.classes
class_dataset
```

```
['F_Breakage', 'F_Crushed', 'F_Normal', 'R_Breakage', 'R_Crushed', 'R_Normal']
```

```
single_image = images[0]
single_image = single_image.permute(1,2,0)
plt.imshow(single_image)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0.
```



Explore images and labels from dataset

```
fig, axes = plt.subplots(2,4, figsize=(17,8))
axes = axes.flatten()
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]
for i in range(8):
    single_image = images[i]  # Clone image to avoid modifying the original data
```

```
    for j in range(3):  # Assuming 3 channels (RGB)
        single_image[j, :, :] = single_image[j, :, :] * std[j] + mean[j]
    single_image = single_image.permute(1,2,0)
    axes[i].imshow(single_image)  #
    axes[i].set_title(class_dataset[labels[i]])
    axes[i].axis('off')
plt.tight_layout()
plt.show()
```

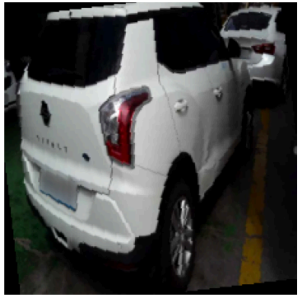
F_Breakage

F_Breakage

R_Crushed

R_Breakage

R_Crushed



F_Breakage

F_Crushed

F_Crushed

Get num classes or output values

```
num_classes = len(class_dataset)
num_classes
```

```
6
```

Create class function for CNN Regular type

```
class CNNCarDamageDetection(nn.Module):
    def __init__(self):
        super().__init__()
        self.classifier = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels = 16, kernel_size=3,stride=1, padding=1), # 16,224,224,224
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0), # 16, 112,112,112
            nn.ReLU(),
            nn.Conv2d(in_channels=16, out_channels= 32, kernel_size=3, padding=1), # 32,112,112,112
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0), # 32, 56,56,56
            nn.Conv2d(in_channels=32, out_channels= 64, kernel_size=3, stride=1, padding=1), # 64,56,56,56
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0), #64,28,28,28
            nn.Flatten(),
            nn.Linear(64*28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 6)
        )
    def forward(self, x):
        x = self.classifier(x)
        return x
```

```
model = CNNCarDamageDetection().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)


print(model)
```

```
CNNCarDamageDetection(
  (classifier): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ReLU()
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=50176, out_features=512, bias=True)
    (10): ReLU()
    (11): Linear(in_features=512, out_features=6, bias=True)
  )
)
```

Define train model with training and evaluation for whole processing of training --> Then get the total loss and accuracy

```
def train_model(model, criterion, optimizer, epochs):
    model.train()
    for epoch in range(epochs):
        total_loss = 0
        for batch , (images, labels) in enumerate(train_loader):
            images = images.to(device)
```

```
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            if batch % 10 == 0:
                print(f" Batch {batch}, epoch :{epoch}, Loss: {loss.item():.4f}")
            total_loss += loss.item()
        print(f"Epoch {epoch+1}/{epochs}, Loss: {total_loss/len(train_loader):.4f}")

        model.eval()
        correct = 0
        total = 0
        total_loss = 0
        with torch.no_grad():
            for images, labels in val_loader:
                images = images.to(device)
                labels = labels.to(device)
                outputs = model(images)

                loss = criterion(outputs, labels)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
                total_loss += loss.item()
            accuracy = 100 * correct / total
            print(f"Total loss : {total_loss/len(val_dataset):.4f}, Validation Accuracy: {accuracy:.2f}%")

train_model(model, criterion, optimizer, epochs=5)
```

```
 Batch 0, epoch :0, Loss: 1.7915
 Batch 10, epoch :0, Loss: 1.8150
 Batch 20, epoch :0, Loss: 1.5238
 Batch 30, epoch :0, Loss: 1.2426
```

```
 Batch 40, epoch :0, Loss: 1.3491
 Batch 50, epoch :0, Loss: 0.9913
Epoch 1/5, Loss: 1.7327
Total loss : 0.0422, Validation Accuracy: 47.48%
 Batch 0, epoch :1, Loss: 1.2057
 Batch 10, epoch :1, Loss: 1.1848
 Batch 20, epoch :1, Loss: 1.1801
 Batch 30, epoch :1, Loss: 1.3338
 Batch 40, epoch :1, Loss: 1.1626
 Batch 50, epoch :1, Loss: 1.0333
Epoch 2/5, Loss: 1.1616
Total loss : 0.0365, Validation Accuracy: 50.09%
 Batch 0, epoch :2, Loss: 1.0040
 Batch 10, epoch :2, Loss: 1.4386
 Batch 20, epoch :2, Loss: 1.1489
 Batch 30, epoch :2, Loss: 0.9798
 Batch 40, epoch :2, Loss: 0.9009
 Batch 50, epoch :2, Loss: 0.9686
Epoch 3/5, Loss: 1.0360
Total loss : 0.0336, Validation Accuracy: 53.39%
 Batch 0, epoch :3, Loss: 0.9200
 Batch 10, epoch :3, Loss: 0.6951
 Batch 20, epoch :3, Loss: 1.0537
 Batch 30, epoch :3, Loss: 1.0142
 Batch 40, epoch :3, Loss: 1.0924
 Batch 50, epoch :3, Loss: 0.9153
Epoch 4/5, Loss: 0.9572
Total loss : 0.0331, Validation Accuracy: 55.48%
 Batch 0, epoch :4, Loss: 0.9700
 Batch 10, epoch :4, Loss: 0.6362
 Batch 20, epoch :4, Loss: 0.7400
 Batch 30, epoch :4, Loss: 0.7664
 Batch 40, epoch :4, Loss: 0.6705
 Batch 50, epoch :4, Loss: 0.7906
Epoch 5/5, Loss: 0.8781
Total loss : 0.0336, Validation Accuracy: 53.57%
```

Try with new class - Adjust with Regularization methods : Add BatchNormalization, Dropout and weight_decay

```
class CNNCarDamageDetectionWithRegularization(nn.Module):
    def __init__(self):
        super().__init__()
        self.classifier = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels = 16, kernel_size=3,stride=1, padding=1), # 16,224,224,224
            nn.BatchNorm2d(16),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0), # 16, 112,112,112
            nn.ReLU(),
            nn.Conv2d(in_channels=16, out_channels= 32, kernel_size=3, padding=1), # 32,112,112,112
            nn.BatchNorm2d(32),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0), # 32, 56,56,56
            nn.Conv2d(in_channels=32, out_channels= 64, kernel_size=3, stride=1, padding=1), # 64,56,56,56
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0), #64,28,28,28
            nn.Flatten(),
            nn.Linear(64*28*28, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 6)
        )

    def forward(self, x):
        x = self.classifier(x)
        return x
```

```
model_reg = CNNCarDamageDetectionWithRegularization().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model_reg.parameters(), lr=0.0001, weight_decay= 1e-4)


train_model(model_reg, criterion, optimizer, epochs=5)
```

```
 Batch 0, epoch :0, Loss: 1.9098
 Batch 10, epoch :0, Loss: 2.6957
 Batch 20, epoch :0, Loss: 1.7979
 Batch 30, epoch :0, Loss: 1.4585
 Batch 40, epoch :0, Loss: 1.3640
 Batch 50, epoch :0, Loss: 1.2843
Epoch 1/5, Loss: 1.8405
Total loss : 0.0390, Validation Accuracy: 48.17%
 Batch 0, epoch :1, Loss: 1.1576
 Batch 10, epoch :1, Loss: 1.3397
 Batch 20, epoch :1, Loss: 1.2216
 Batch 30, epoch :1, Loss: 1.0653
 Batch 40, epoch :1, Loss: 1.0545
 Batch 50, epoch :1, Loss: 0.8547
Epoch 2/5, Loss: 1.1593
Total loss : 0.0354, Validation Accuracy: 54.78%
 Batch 0, epoch :2, Loss: 0.8359
 Batch 10, epoch :2, Loss: 1.1201
 Batch 20, epoch :2, Loss: 1.1091
 Batch 30, epoch :2, Loss: 1.2565
 Batch 40, epoch :2, Loss: 1.2556
 Batch 50, epoch :2, Loss: 1.1074
Epoch 3/5, Loss: 1.0749
Total loss : 0.0367, Validation Accuracy: 46.09%
 Batch 0, epoch :3, Loss: 1.1370
 Batch 10, epoch :3, Loss: 1.1934
 Batch 20, epoch :3, Loss: 0.9096
 Batch 30, epoch :3, Loss: 1.1594
 Batch 40, epoch :3, Loss: 0.9951
 Batch 50, epoch :3, Loss: 0.8407
Epoch 4/5, Loss: 1.0199
Total loss : 0.0335, Validation Accuracy: 55.65%
```

```
 Batch 0, epoch :4, Loss: 0.9767
 Batch 10, epoch :4, Loss: 1.0922
 Batch 20, epoch :4, Loss: 1.1101
 Batch 30, epoch :4, Loss: 1.3670
 Batch 40, epoch :4, Loss: 0.8108
 Batch 50, epoch :4, Loss: 0.8644
Epoch 5/5, Loss: 0.9571
Total loss : 0.0337, Validation Accuracy: 51.48%
```

Try with pretrained transfer model Efficient_b0 : Using dropout 0.5, optimizer get optimize only the unfrozen parameters.

```
class CarClassifierEfficientNet(nn.Module):
    def __init__(self, num_classes, dropout_rate=0.5):
        super().__init__()
        self.model = models.efficientnet_b0(weights='DEFAULT')

        # Freeze all layers except the classifier
        for param in self.model.parameters():
            param.requires_grad = False

        # Unfreeze the last layers of EfficientNet (adjust as needed)
        for param in self.model.features[5:].parameters(): # Example: unfreeze from the 6th block onwards
          param.requires_grad = True

        # Replace the final fully connected layer with a custom classifier
        num_ftrs = self.model.classifier[1].in_features
        self.model.classifier = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(num_ftrs, num_classes)
        )
```

```
    def forward(self, x):
        x = self.model(x)
        return x

model = CarClassifierEfficientNet(num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=0.001) # Optimize only the unfrozen p

train_model(model, criterion, optimizer, epochs=8)
```

```
Epoch 1/8, Loss: 0.9469
Total loss : 0.0183, Validation Accuracy: 76.17%
 Batch 0, epoch :1, Loss: 0.2808
 Batch 10, epoch :1, Loss: 1.1727
 Batch 20, epoch :1, Loss: 0.6214
 Batch 30, epoch :1, Loss: 0.8631
 Batch 40, epoch :1, Loss: 0.7632
 Batch 50, epoch :1, Loss: 0.8972
Epoch 2/8, Loss: 0.7548
Total loss : 0.0214, Validation Accuracy: 68.87%
 Batch 0, epoch :2, Loss: 0.5035
 Batch 10, epoch :2, Loss: 0.6732
 Batch 20, epoch :2, Loss: 0.4856
 Batch 30, epoch :2, Loss: 0.9077
 Batch 40, epoch :2, Loss: 0.5971
 Batch 50, epoch :2, Loss: 0.6357
Epoch 3/8, Loss: 0.5694
Total loss : 0.0215, Validation Accuracy: 69.57%
 Batch 0, epoch :3, Loss: 0.4457
 Batch 10, epoch :3, Loss: 0.3111
 Batch 20, epoch :3, Loss: 0.7232
 Batch 30, epoch :3, Loss: 0.6661
 Batch 40, epoch :3, Loss: 0.9284
```

```
 Batch 30, epoch :4, Loss: 0.4412
 Batch 40, epoch :4, Loss: 0.3876
 Batch 50, epoch :4, Loss: 0.4754
Epoch 5/8, Loss: 0.4471
Total loss : 0.0182, Validation Accuracy: 77.39%
 Batch 0, epoch :5, Loss: 0.2819
 Batch 10, epoch :5, Loss: 0.2796
 Batch 20, epoch :5, Loss: 0.1733
 Batch 30, epoch :5, Loss: 0.3456
 Batch 40, epoch :5, Loss: 0.4338
 Batch 50, epoch :5, Loss: 0.5376
Epoch 6/8, Loss: 0.3632
Total loss : 0.0186, Validation Accuracy: 75.30%
 Batch 0, epoch :6, Loss: 0.3549
 Batch 10, epoch :6, Loss: 0.2958
 Batch 20, epoch :6, Loss: 0.3091
 Batch 30, epoch :6, Loss: 0.2527
 Batch 40, epoch :6, Loss: 0.2024
 Batch 50, epoch :6, Loss: 0.1969
Epoch 7/8, Loss: 0.2841
Total loss : 0.0181, Validation Accuracy: 80.87%
 Batch 0, epoch :7, Loss: 0.1964
 Batch 10, epoch :7, Loss: 0.1795
 Batch 20, epoch :7, Loss: 0.3300
 Batch 30, epoch :7, Loss: 0.1777
 Batch 40, epoch :7, Loss: 0.4229
 Batch 50, epoch :7, Loss: 0.5495
Epoch 8/8, Loss: 0.2557
Total loss : 0.0194, Validation Accuracy: 79.13%
```

Try with Pretrained ResNet 50 : Dropout 0.5

```
# Load the pre-trained ResNet model
import torchvision.models as models

class CarClassifierResNet(nn.Module):
```

```
    def __init__(self, num_classes, dropout_rate=0.5):
        super().__init__()
        self.model = models.resnet50(weights='DEFAULT')
        # Freeze all layers except the final fully connected layer
        for param in self.model.parameters():
            param.requires_grad = False

        # Unfreeze layer4 and fc layers
        for param in self.model.layer4.parameters():
            param.requires_grad = True

        # Replace the final fully connected layer
        self.model.fc = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(self.model.fc.in_features, num_classes)
        )

    def forward(self, x):
        x = self.model(x)
        return x

model = CarClassifierResNet(num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=0.001)

train_model(model, criterion, optimizer, epochs=10)
```

```
 Batch 0, epoch :0, Loss: 1.7692
 Batch 10, epoch :0, Loss: 0.9476
 Batch 20, epoch :0, Loss: 0.6264
 Batch 30, epoch :0, Loss: 0.7514
 Batch 40, epoch :0, Loss: 0.8097
 Batch 50, epoch :0, Loss: 0.7794
Epoch 1/10, Loss: 0.9128
Total loss : 0.0308, Validation Accuracy: 66.09%
 Batch 0, epoch :1, Loss: 1.2675
 Batch 10, epoch :1, Loss: 0.8614
```

```
 Batch 20, epoch :1, Loss: 1.0086
 Batch 30, epoch :1, Loss: 1.1123
 Batch 40, epoch :1, Loss: 0.6762
 Batch 50, epoch :1, Loss: 0.7631
Epoch 2/10, Loss: 0.8955
Total loss : 0.0608, Validation Accuracy: 37.22%
 Batch 0, epoch :2, Loss: 1.6019
 Batch 10, epoch :2, Loss: 0.7985
 Batch 20, epoch :2, Loss: 1.0732
 Batch 30, epoch :2, Loss: 0.8293
 Batch 40, epoch :2, Loss: 0.9241
 Batch 50, epoch :2, Loss: 0.4874
Epoch 3/10, Loss: 0.7502
Total loss : 0.0212, Validation Accuracy: 72.17%
 Batch 0, epoch :3, Loss: 0.4665
 Batch 10, epoch :3, Loss: 0.6960
 Batch 20, epoch :3, Loss: 0.4533
 Batch 30, epoch :3, Loss: 0.5170
 Batch 40, epoch :3, Loss: 0.6698
 Batch 50, epoch :3, Loss: 0.4675
Epoch 4/10, Loss: 0.5282
Total loss : 0.0175, Validation Accuracy: 78.26%
 Batch 0, epoch :4, Loss: 0.3341
 Batch 10, epoch :4, Loss: 0.4795
 Batch 20, epoch :4, Loss: 0.4112
 Batch 30, epoch :4, Loss: 0.3586
 Batch 40, epoch :4, Loss: 0.5767
 Batch 50, epoch :4, Loss: 0.4267
Epoch 5/10, Loss: 0.4229
Total loss : 0.0191, Validation Accuracy: 77.74%
 Batch 0, epoch :5, Loss: 0.3165
 Batch 10, epoch :5, Loss: 0.7581
 Batch 20, epoch :5, Loss: 0.3775
 Batch 30, epoch :5, Loss: 0.4053
 Batch 40, epoch :5, Loss: 0.2278
 Batch 50, epoch :5, Loss: 0.3828
Epoch 6/10, Loss: 0.3721
Total loss : 0.0194, Validation Accuracy: 76.00%
 Batch 0, epoch :6, Loss: 0.3947
```

```
 Batch 10, epoch :6, Loss: 0.2291
 Batch 20, epoch :6, Loss: 0.2080
 Batch 30, epoch :6, Loss: 0.1134
 Batch 40, epoch :6, Loss: 0.2695
 Batch 50, epoch :6, Loss: 0.2331
Epoch 7/10, Loss: 0.3176
Total loss : 0.0163, Validation Accuracy: 79.30%
 Batch 0, epoch :7, Loss: 0.1433
```

```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.1.0-py3-none-any.whl.metadata (16 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.14.0-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from optuna) (24.2)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.10/dist-packages (from optuna) (2.0.36)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna) (6.0.2)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.8-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.4.2
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1
Downloading optuna-4.1.0-py3-none-any.whl (364 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 364.4/364.4 kB 23.5 MB/s eta 0:00:00
Downloading alembic-1.14.0-py3-none-any.whl (233 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.5/233.5 kB 18.4 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Downloading Mako-1.3.8-py3-none-any.whl (78 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.6/78.6 kB 7.2 MB/s eta 0:00:00
Installing collected packages: Mako, colorlog, alembic, optuna
Successfully installed Mako-1.3.8 alembic-1.14.0 colorlog-6.9.0 optuna-4.1.0
```

Try Optuna for Pretrained ResNet 50 --> Then get the best params

```
import optuna

def objective(trial):
    # Hyperparameter to tune
    dropout_rate = trial.suggest_float("dropout_rate", 0.2, 0.7)
    lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)

    # Model creation
    model = CarClassifierResNet(num_classes=num_classes, dropout_rate=dropout_rate).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=lr)

    # Training loop (1 epoch only to minimize runtime for the search)
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

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

    # Validation accuracy
```

```
    model.eval()
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (predicted == labels).sum().item()

    val_accuracy = val_correct / val_total
    return val_accuracy

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=20)

print("Best trial:")
print(study.best_trial)
```

```
[I 2025-01-09 11:43:55,437] A new study created in memory with name: no-name-8c0a9b55-0e50-4877-9048-67d91ab576be
<ipython-input-46-0489bbd37719>:6: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature w
  lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)
[I 2025-01-09 11:44:35,616] Trial 0 finished with value: 0.7043478260869566 and parameters: {'dropout_rate': 0.272
[I 2025-01-09 11:45:14,767] Trial 1 finished with value: 0.3217391304347826 and parameters: {'dropout_rate': 0.536
[I 2025-01-09 11:45:53,909] Trial 2 finished with value: 0.2817391304347826 and parameters: {'dropout_rate': 0.661
[I 2025-01-09 11:46:33,298] Trial 3 finished with value: 0.7008695652173913 and parameters: {'dropout_rate': 0.645
[I 2025-01-09 11:47:12,920] Trial 4 finished with value: 0.6504347826086957 and parameters: {'dropout_rate': 0.566
[I 2025-01-09 11:47:52,426] Trial 5 finished with value: 0.7356521739130435 and parameters: {'dropout_rate': 0.223
[I 2025-01-09 11:48:31,538] Trial 6 finished with value: 0.4782608695652174 and parameters: {'dropout_rate': 0.653
[I 2025-01-09 11:49:11,360] Trial 7 finished with value: 0.7373913043478261 and parameters: {'dropout_rate': 0.597
[I 2025-01-09 11:49:50,597] Trial 8 finished with value: 0.6 and parameters: {'dropout_rate': 0.5834303354124549,
[I 2025-01-09 11:50:29,171] Trial 9 finished with value: 0.5182608695652174 and parameters: {'dropout_rate': 0.489
[I 2025-01-09 11:51:08,155] Trial 10 finished with value: 0.7495652173913043 and parameters: {'dropout_rate': 0.37
[I 2025-01-09 11:51:46,961] Trial 11 finished with value: 0.711304347826087 and parameters: {'dropout_rate': 0.385
[I 2025-01-09 11:52:25,938] Trial 12 finished with value: 0.7008695652173913 and parameters: {'dropout_rate': 0.37
[I 2025-01-09 11:53:04,569] Trial 13 finished with value: 0.688695652173913 and parameters: {'dropout_rate': 0.393
[I 2025-01-09 11:53:43,453] Trial 14 finished with value: 0.7252173913043478 and parameters: {'dropout_rate': 0.31
```

```
[I 2025-01-09 11:54:22,223] Trial 15 finished with value: 0.6678260869565218 and parameters: {'dropout_rate': 0.47
[I 2025-01-09 11:55:00,860] Trial 16 finished with value: 0.697391304347826 and parameters: {'dropout_rate': 0.446
[I 2025-01-09 11:55:39,791] Trial 17 finished with value: 0.7095652173913043 and parameters: {'dropout_rate': 0.32
[I 2025-01-09 11:56:18,893] Trial 18 finished with value: 0.44869565217391305 and parameters: {'dropout_rate': 0.6
[I 2025-01-09 11:56:57,745] Trial 19 finished with value: 0.6173913043478261 and parameters: {'dropout_rate': 0.51
Best trial:
FrozenTrial(number=10, state=TrialState.COMPLETE, values=[0.7495652173913043], datetime_start=datetime.datetime(20
```

```
best_params={'dropout_rate': 0.37112580272910733, 'lr': 0.0010867605115576732}
```

Try Pretrained transder ResNet 50 with the best params from optuna

```
# Load the pre-trained ResNet model
import torchvision.models as models

class CarClassifierResNet(nn.Module):
    def __init__(self, num_classes, dropout_rate=best_params['dropout_rate']):
        super().__init__()
        self.model = models.resnet50(weights='DEFAULT')
        # Freeze all layers except the final fully connected layer
        for param in self.model.parameters():
            param.requires_grad = False

        # Unfreeze layer4 and fc layers
        for param in self.model.layer4.parameters():
            param.requires_grad = True

        # Replace the final fully connected layer
        self.model.fc = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(self.model.fc.in_features, num_classes)
```

```
        )

    def forward(self, x):
        x = self.model(x)
        return x

model = CarClassifierResNet(num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=best_params['lr'])

train_model(model, criterion, optimizer, epochs=8)
```

```
Epoch 1/8, Loss: 0.9093
Total loss : 0.0321, Validation Accuracy: 65.39%
 Batch 0, epoch :1, Loss: 1.1587
 Batch 10, epoch :1, Loss: 0.9225
 Batch 20, epoch :1, Loss: 1.2516
 Batch 30, epoch :1, Loss: 1.3139
 Batch 40, epoch :1, Loss: 0.9367
 Batch 50, epoch :1, Loss: 0.6715
Epoch 2/8, Loss: 1.0242
Total loss : 0.0301, Validation Accuracy: 57.57%
 Batch 0, epoch :2, Loss: 0.8892
 Batch 10, epoch :2, Loss: 0.5044
 Batch 20, epoch :2, Loss: 0.5705
 Batch 30, epoch :2, Loss: 0.5139
 Batch 40, epoch :2, Loss: 0.7839
 Batch 50, epoch :2, Loss: 0.8427
Epoch 3/8, Loss: 0.6980
Total loss : 0.0232, Validation Accuracy: 65.22%
 Batch 0, epoch :3, Loss: 0.5846
 Batch 10, epoch :3, Loss: 0.9578
```

```
 Batch 0, epoch :4, Loss: 0.4459
 Batch 10, epoch :4, Loss: 0.3609
 Batch 20, epoch :4, Loss: 0.3675
 Batch 30, epoch :4, Loss: 0.5202
 Batch 40, epoch :4, Loss: 0.6305
 Batch 50, epoch :4, Loss: 0.4732
Epoch 5/8, Loss: 0.4521
Total loss : 0.0165, Validation Accuracy: 76.52%
 Batch 0, epoch :5, Loss: 0.2320
 Batch 10, epoch :5, Loss: 0.3211
 Batch 20, epoch :5, Loss: 0.4068
 Batch 30, epoch :5, Loss: 0.4581
 Batch 40, epoch :5, Loss: 0.3769
 Batch 50, epoch :5, Loss: 0.3862
Epoch 6/8, Loss: 0.4271
Total loss : 0.0181, Validation Accuracy: 76.87%
 Batch 0, epoch :6, Loss: 0.2396
 Batch 10, epoch :6, Loss: 0.6497
 Batch 20, epoch :6, Loss: 0.3915
 Batch 30, epoch :6, Loss: 0.3626
 Batch 40, epoch :6, Loss: 0.5431
 Batch 50, epoch :6, Loss: 0.2204
Epoch 7/8, Loss: 0.3690
Total loss : 0.0158, Validation Accuracy: 78.78%
 Batch 0, epoch :7, Loss: 0.2836
 Batch 10, epoch :7, Loss: 0.2700
 Batch 20, epoch :7, Loss: 0.1793
 Batch 30, epoch :7, Loss: 0.3212
 Batch 40, epoch :7, Loss: 0.2214
 Batch 50, epoch :7, Loss: 0.4015
Epoch 8/8, Loss: 0.2871
Total loss : 0.0186, Validation Accuracy: 78.09%
```

Try optuna with Efficient pretrained model

```
# prompt: write optuna for my class CarClassifierEfficientNet
```

```
def objective(trial):
    # Hyperparameter to tune
    dropout_rate = trial.suggest_float("dropout_rate", 0.2, 0.7)
    lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)

    # Model creation
    model = CarClassifierEfficientNet(num_classes=num_classes, dropout_rate=dropout_rate).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=lr)

    # Training loop (1 epoch only to minimize runtime for the search)
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

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

    # Validation accuracy
    model.eval()
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
```

```
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            val_total += labels.size(0)
            val_correct += (predicted == labels).sum().item()

    val_accuracy = val_correct / val_total
    return val_accuracy

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=20)

print("Best trial:")
print(study.best_trial)

best_params = study.best_params
print(best_params)
```

```
[I 2025-01-09 12:38:04,674] A new study created in memory with name: no-name-0699d08d-6be0-42fb-8f63-6809f4acd63a
<ipython-input-49-0519d32e9161>:6: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature w
  lr = trial.suggest_loguniform("lr", 1e-5, 1e-2)
[I 2025-01-09 12:38:42,329] Trial 0 finished with value: 0.7617391304347826 and parameters: {'dropout_rate': 0.560
[I 2025-01-09 12:39:19,808] Trial 1 finished with value: 0.4260869565217391 and parameters: {'dropout_rate': 0.325
[I 2025-01-09 12:39:55,927] Trial 2 finished with value: 0.6608695652173913 and parameters: {'dropout_rate': 0.479
[I 2025-01-09 12:40:32,177] Trial 3 finished with value: 0.568695652173913 and parameters: {'dropout_rate': 0.6841
[I 2025-01-09 12:41:08,120] Trial 4 finished with value: 0.49043478260869566 and parameters: {'dropout_rate': 0.50
[I 2025-01-09 12:41:43,648] Trial 5 finished with value: 0.7217391304347827 and parameters: {'dropout_rate': 0.481
[I 2025-01-09 12:42:19,450] Trial 6 finished with value: 0.3773913043478261 and parameters: {'dropout_rate': 0.295
[I 2025-01-09 12:42:56,163] Trial 7 finished with value: 0.5582608695652174 and parameters: {'dropout_rate': 0.281
[I 2025-01-09 12:43:32,026] Trial 8 finished with value: 0.7008695652173913 and parameters: {'dropout_rate': 0.621
[I 2025-01-09 12:44:07,987] Trial 9 finished with value: 0.3026086956521739 and parameters: {'dropout_rate': 0.650
[I 2025-01-09 12:44:43,729] Trial 10 finished with value: 0.6608695652173913 and parameters: {'dropout_rate': 0.38
[I 2025-01-09 12:45:19,758] Trial 11 finished with value: 0.711304347826087 and parameters: {'dropout_rate': 0.564
[I 2025-01-09 12:45:55,944] Trial 12 finished with value: 0.7391304347826086 and parameters: {'dropout_rate': 0.41
[I 2025-01-09 12:46:31,638] Trial 13 finished with value: 0.6608695652173913 and parameters: {'dropout_rate': 0.37
[I 2025-01-09 12:47:08,601] Trial 14 finished with value: 0.5947826086956521 and parameters: {'dropout_rate': 0.55
[I 2025-01-09 12:47:44,878] Trial 15 finished with value: 0.7321739130434782 and parameters: {'dropout_rate': 0.21
[I 2025-01-09 12:48:20,529] Trial 16 finished with value: 0.6991304347826087 and parameters: {'dropout_rate': 0.40
[I 2025-01-09 12:48:56,403] Trial 17 finished with value: 0.7339130434782609 and parameters: {'dropout_rate': 0.55
```

```
[I 2025-01-09 12:49:34,325] Trial 18 finished with value: 0.6730434782608695 and parameters: {'dropout_rate': 0.43
[I 2025-01-09 12:50:13,760] Trial 19 finished with value: 0.648695652173913 and parameters: {'dropout_rate': 0.606
Best trial:
FrozenTrial(number=0, state=TrialState.COMPLETE, values=[0.7617391304347826], datetime_start=datetime.datetime(202
{'dropout_rate': 0.560122966606635, 'lr': 0.0005842324132540079}
```

```
best_params_efficient=best_params
best_params_efficient
```

```
{'dropout_rate': 0.560122966606635, 'lr': 0.0005842324132540079}
```

Setup train model again to get predict and labels for report accuracy and confusion matrics

```
def train_model(model, criterion, optimizer, epochs):
    total_val = []
    total_prediction = []
    model.train()
    for epoch in range(epochs):
        total_loss = 0
        for batch , (images, labels) in enumerate(train_loader):
            images = images.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            if batch % 10 == 0:
```

```
            print(f" Batch {batch}, epoch :{epoch}, Loss: {loss.item():.4f}")
        total_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {total_loss/len(train_loader):.4f}")

    model.eval()
    correct = 0
    total = 0
    total_loss = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)

            loss = criterion(outputs, labels)
            _, predicted = torch.max(outputs.data, 1)


            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            total_prediction.extend(predicted.cpu().numpy())
            total_val.extend(labels.cpu().numpy())

            total_loss += loss.item()
        accuracy = 100 * correct / total
        print(f"Total loss : {total_loss/len(val_dataset):.4f}, Validation Accuracy: {accuracy:.2f}%")

return total_prediction, total_val
```

Set up class with the best params from Pretrained optuna then get the reults of accuracy, predict and labels

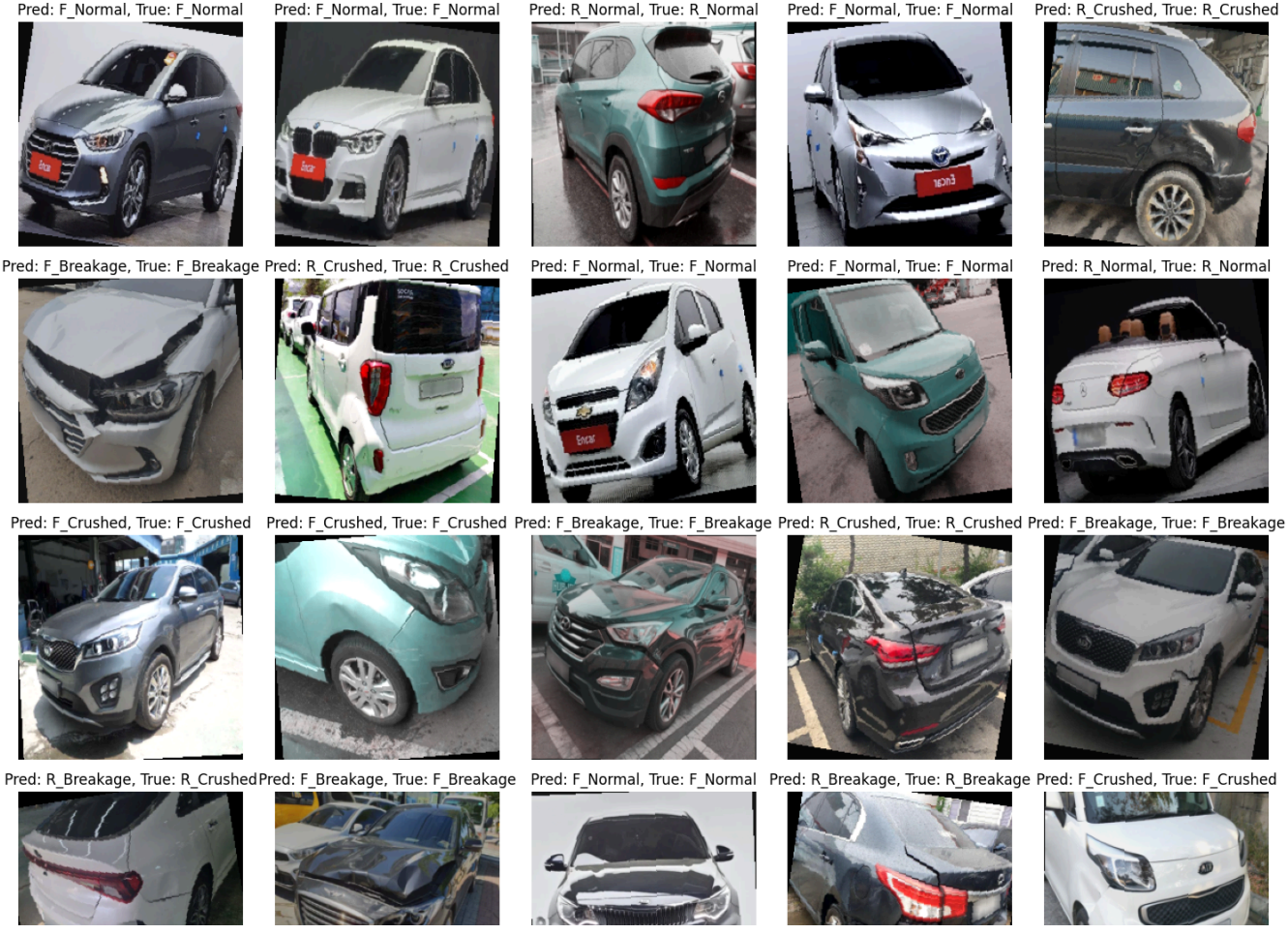
```
class CarClassifierEfficientNet(nn.Module):
    def __init__(self, num_classes, dropout_rate=best_params_efficient['dropout_rate']):
        super().__init__()
        self.model = models.efficientnet_b0(weights='DEFAULT')

        # Freeze all layers except the classifier
        for param in self.model.parameters():
            param.requires_grad = False

        # Unfreeze the last layers of EfficientNet (adjust as needed)
        for param in self.model.features[5:].parameters(): # Example: unfreeze from the 6th block onwards
          param.requires_grad = True

        # Replace the final fully connected layer with a custom classifier
        num_ftrs = self.model.classifier[1].in_features
        self.model.classifier = nn.Sequential(
            nn.Dropout(dropout_rate),
            nn.Linear(num_ftrs, num_classes)
        )

    def forward(self, x):
        x = self.model(x)
        return x

model = CarClassifierEfficientNet(num_classes=num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=best_params_efficient['lr']) # Optimiz

prediction, labels = train_model(model, criterion, optimizer, epochs=5)
```

```
Batch 0, epoch :0, Loss: 1.8084
Batch 10, epoch :0, Loss: 1.1827
Batch 20, epoch :0, Loss: 1.1280
Batch 30, epoch :0, Loss: 0.7849
Batch 40, epoch :0, Loss: 0.9460
```

```
 Batch 50, epoch :0, Loss: 0.6985
Epoch 1/5, Loss: 1.0341
Total loss : 0.0188, Validation Accuracy: 73.74%
 Batch 0, epoch :1, Loss: 0.3271
 Batch 10, epoch :1, Loss: 0.6702
 Batch 20, epoch :1, Loss: 1.0155
 Batch 30, epoch :1, Loss: 0.7959
 Batch 40, epoch :1, Loss: 0.7414
 Batch 50, epoch :1, Loss: 0.6368
Epoch 2/5, Loss: 0.6365
Total loss : 0.0193, Validation Accuracy: 73.57%
 Batch 0, epoch :2, Loss: 0.4305
 Batch 10, epoch :2, Loss: 0.7049
 Batch 20, epoch :2, Loss: 0.5458
 Batch 30, epoch :2, Loss: 0.3283
 Batch 40, epoch :2, Loss: 0.5231
 Batch 50, epoch :2, Loss: 0.5072
Epoch 3/5, Loss: 0.4933
Total loss : 0.0162, Validation Accuracy: 78.26%
 Batch 0, epoch :3, Loss: 0.3765
 Batch 10, epoch :3, Loss: 0.5095
 Batch 20, epoch :3, Loss: 0.5158
 Batch 30, epoch :3, Loss: 0.1580
 Batch 40, epoch :3, Loss: 0.2428
 Batch 50, epoch :3, Loss: 0.3626
Epoch 4/5, Loss: 0.4476
Total loss : 0.0244, Validation Accuracy: 68.70%
 Batch 0, epoch :4, Loss: 0.6091
 Batch 10, epoch :4, Loss: 0.3809
 Batch 20, epoch :4, Loss: 0.2155
 Batch 30, epoch :4, Loss: 0.3771
 Batch 40, epoch :4, Loss: 0.3879
 Batch 50, epoch :4, Loss: 0.4812
Epoch 5/5, Loss: 0.3720
Total loss : 0.0146, Validation Accuracy: 80.70%
```

```
import random
```

```
def predict_random_images(model, dataset, class_dataset, num_images=25):
    model.eval()
    random_indices = random.sample(range(len(dataset)), num_images)
    images, labels = zip(*[dataset[i] for i in random_indices])

    images = torch.stack(images).to(device)
    labels = torch.Tensor(labels).long().to(device)

    with torch.no_grad():
        outputs = model(images)
        _, predictions = torch.max(outputs, 1)

    fig, axes = plt.subplots(5, 5, figsize=(15, 15))
    axes = axes.flatten()

    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]

    for i, (img, pred, label, ax) in enumerate(zip(images, predictions, labels, axes)):
        img = img.cpu()
        for j in range(3):
            img[j] = img[j] * std[j] + mean[j]
        img = img.permute(1, 2, 0).numpy()

        ax.imshow(img)
        ax.set_title(f"Pred: {class_dataset[pred]}, True: {class_dataset[label]}")
        ax.axis('off')

    plt.tight_layout()
    plt.show()

predict_random_images(model, dataset, class_dataset)
```


Pred: F_Normal, True: F_Normal
Pred: F_Normal, True: F_Normal
Pred: R_Normal, True: R_Normal
Pred: F_Normal, True: F_Normal
Pred: R_Crushed, True: R_Crushed
Pred: F_Breakage, True: F_Breakage
Pred: R_Crushed, True: R_Crushed
Pred: F_Normal, True: F_Normal
Pred: F_Normal, True: F_Normal
Pred: R_Normal, True: R_Normal
Pred: F_Crushed, True: F_Crushed
Pred: F_Crushed, True: F_Crushed
Pred: F_Breakage, True: F_Breakage
Pred: R_Crushed, True: R_Crushed
Pred: F_Breakage, True: F_Breakage
Pred: R_Breakage, True: R_Crushed
Pred: F_Breakage, True: F_Breakage
Pred: F_Normal, True: F_Normal
Pred: R_Breakage, True: R_Breakage
Pred: F_Crushed, True: F_Crushed

Pred: R_Normal, True: R_Normal

Pred: R_Crushed, True: R_Crushed

Pred: R_Normal, True: R_Normal

Pred: R_Crushed, True: R_Crushed

Pred: F_Breakage, True: F_Breakage

The accuracy is around 80% and with the best param from optuna for pretrained model transfer Efficient.

Check report accuracy

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns


report = classification_report(labels, prediction, target_names=class_dataset)
print(report)
```

```
                precision    recall  f1-score   support

      F_Breakage       0.78      0.88      0.83       660
       F_Crushed       0.71      0.68      0.70       545
        F_Normal       0.86      0.83      0.85       595
      R_Breakage       0.72      0.64      0.68       335
       R_Crushed       0.56      0.69      0.62       370
        R_Normal       0.83      0.65      0.73       370

        accuracy                           0.75      2875
       macro avg       0.75      0.73      0.73      2875
    weighted avg       0.76      0.75      0.75      2875
```

```
cm = confusion_matrix(labels, prediction)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_dataset, yticklabels=class_dataset)
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
plt.show()
```