Imports

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as patches
from sklearn.metrics import precision_recall_curve,
average_precision_score
from sklearn.preprocessing import LabelEncoder
from torchvision import transforms
from PIL import Image
from tqdm import tqdm
import shutil
from sklearn.metrics import roc_curve, auc
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, models
from torch.utils.data import Dataset, DataLoader, random_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import pandas as pd

# Load the provided CSV file
input_csv_path = '/kaggle/input/gtsrb-german-traffic-sign/Test.csv'
output_csv_path = '/kaggle/working/Brightness_Test.csv'

# Read the input CSV file
df = pd.read_csv(input_csv_path)

# Replace the path in the relevant column
# Assuming the column containing paths is the last column in the
DataFrame
df.iloc[:, -1] = df.iloc[:, -1].str.replace('Test/',
'brightness_images/', regex=False)

# Save the updated DataFrame to a new CSV file
df.to_csv(output_csv_path, index=False)

output_csv_path
```

```
'/kaggle/working/Brightness_Test.csv'
```

```
import pandas as pd

# Load the provided CSV file
input_csv_path = '/kaggle/input/gtsrb-german-traffic-sign/Test.csv'
```

```
output_csv_path = '/kaggle/working/MotionBlur_Test.csv'

# Read the input CSV file
df = pd.read_csv(input_csv_path)

# Replace the path in the relevant column
# Assuming the column containing paths is the last column in the
DataFrame
df.iloc[:, -1] = df.iloc[:, -1].str.replace('Test/',
'motion_blur_images/', regex=False)

# Save the updated DataFrame to a new CSV file
df.to_csv(output_csv_path, index=False)

output_csv_path
```

```
'/kaggle/working/MotionBlur_Test.csv'
```

```
import pandas as pd

# Load the provided CSV file
input_csv_path = '/kaggle/input/gtsrb-german-traffic-sign/Test.csv'
output_csv_path = '/kaggle/working/Rain_Test.csv'

# Read the input CSV file
df = pd.read_csv(input_csv_path)

# Replace the path in the relevant column
# Assuming the column containing paths is the last column in the
DataFrame
df.iloc[:, -1] = df.iloc[:, -1].str.replace('Test/', 'rain/',
regex=False)

# Save the updated DataFrame to a new CSV file
df.to_csv(output_csv_path, index=False)

output_csv_path
```

```
'/kaggle/working/Rain_Test.csv'
```

```
import pandas as pd

# Load the provided CSV file
input_csv_path = '/kaggle/input/gtsrb-german-traffic-sign/Test.csv'
output_csv_path = '/kaggle/working/Snow_Test.csv'

# Read the input CSV file
df = pd.read_csv(input_csv_path)

# Replace the path in the relevant column
# Assuming the column containing paths is the last column in the
```

```
DataFrame
df.iloc[:, -1] = df.iloc[:, -1].str.replace('Test/', 'snow/',
regex=False)

# Save the updated DataFrame to a new CSV file
df.to_csv(output_csv_path, index=False)

output_csv_path
```

```
'/kaggle/working/Snow_Test.csv'
```

```
import pandas as pd

# Load the provided CSV file
input_csv_path = '/kaggle/input/gtsrb-german-traffic-sign/Test.csv'
output_csv_path = '/kaggle/working/Rotate_Test.csv'

# Read the input CSV file
df = pd.read_csv(input_csv_path)

# Replace the path in the relevant column
# Assuming the column containing paths is the last column in the
DataFrame
df.iloc[:, -1] = df.iloc[:, -1].str.replace('Test/', 'rotate/',
regex=False)

# Save the updated DataFrame to a new CSV file
df.to_csv(output_csv_path, index=False)

output_csv_path
```

```
'/kaggle/working/Rotate_Test.csv'
```

Paths

```
test_base_dir ="/kaggle/input/gtsrb-german-traffic-sign"
test_csv_path = '/kaggle/input/gtsrb-german-traffic-sign/Test.csv'

new_dataset_base_dir ="/kaggle/input/gtsrb-brightness"

brightness_csv_path = '/kaggle/working/Brightness_Test.csv'
motion_blur_csv_path = '/kaggle/working/MotionBlur_Test.csv'
rain_csv_path = '/kaggle/working/Rain_Test.csv'
snow_csv_path = '/kaggle/working/Snow_Test.csv'
rotate_csv_path = '/kaggle/working/Rotate_Test.csv'
```

Custom Data class

```
class CustomDataset(Dataset):
    def __init__(self, dataframe, base_dir, transform=None):
        self.dataframe = dataframe
        self.base_dir = base_dir
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_path = os.path.join(self.base_dir,
str(self.dataframe.iloc[idx, -1]))
        label = int(self.dataframe.iloc[idx, -2])
        try:
            image = Image.open(img_path).convert("RGB")
        except FileNotFoundError:
            raise ValueError(f"Image not found: {img_path}")
        if self.transform:
            image = self.transform(image)
        return image, label


test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]),
])
```

Data Loaders

```
test_df = pd.read_csv(test_csv_path)
brightness_test_df = pd.read_csv(brightness_csv_path)
motion_blur_test_df = pd.read_csv(motion_blur_csv_path)
rain_test_df = pd.read_csv(rain_csv_path)
snow_test_df = pd.read_csv(snow_csv_path)
rotate_test_df = pd.read_csv(rotate_csv_path)

test_dataset = CustomDataset(test_df, test_base_dir,
transform=test_transforms)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)

brightness_test_dataset = CustomDataset(brightness_test_df,
new_dataset_base_dir, transform=test_transforms)
brightness_test_loader = DataLoader(brightness_test_dataset,
batch_size=1, shuffle=False)
```

```
motion_blur_test_dataset = CustomDataset(motion_blur_test_df,
new_dataset_base_dir, transform=test_transforms)
motion_blur_test_loader = DataLoader(motion_blur_test_dataset,
batch_size=1, shuffle=False)

rain_test_dataset = CustomDataset(rain_test_df, new_dataset_base_dir,
transform=test_transforms)
rain_test_loader = DataLoader(rain_test_dataset, batch_size=1,
shuffle=False)

snow_test_dataset = CustomDataset(snow_test_df, new_dataset_base_dir,
transform=test_transforms)
snow_test_loader = DataLoader(snow_test_dataset, batch_size=1,
shuffle=False)

rotate_test_dataset = CustomDataset(rotate_test_df,
new_dataset_base_dir, transform=test_transforms)
rotate_test_loader = DataLoader(rotate_test_dataset, batch_size=1,
shuffle=False)
```

Load Model

```
model_path =
"/kaggle/input/theiss_model/pytorch/default/1/mobilenet_v2_traffic_sig
ns.pth"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model =
models.mobilenet_v2(weights=models.MobileNet_V2_Weights.IMAGENET1K_V1)
model.classifier[1] = nn.Linear(model.last_channel, 43)
model.load_state_dict(torch.load(model_path))
model = model.to(device)
model.eval()
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-
b0353104.pth" to /root/.cache/torch/hub/checkpoints/mobilenet_v2-
b0353104.pth
100%|██████████| 13.6M/13.6M [00:00<00:00, 79.4MB/s]
<ipython-input-16-e723f4d4784d>:6: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
```

```
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  model.load_state_dict(torch.load(model_path))

MobileNetV2(
  (features): Sequential(
    (0): Conv2dNormActivation(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU6(inplace=True)
    )
    (1): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=32, bias=False)
          (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (2): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=96, bias=False)
          (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
        (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (3): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=144, bias=False)
          (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(144, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (4): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=144, bias=False)
          (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(144, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): InvertedResidual(
```

```
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=192, bias=False)
          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=192, bias=False)
          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(192, 192, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=192, bias=False)
          (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (8): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=384, bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (9): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1), groups=384, bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (10): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=384, bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (11): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=384, bias=False)
          (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
```

```
        (2): Conv2d(384, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (12): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=576, bias=False)
          (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (13): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=576, bias=False)
          (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
```

```
    )
    (14): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(576, 576, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=576, bias=False)
          (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(576, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (15): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
          (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (16): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
          (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
          (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (17): InvertedResidual(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=960, bias=False)
          (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(960, 320, kernel_size=(1, 1), stride=(1, 1),
bias=False)
        (3): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (18): Conv2dNormActivation(
      (0): Conv2d(320, 1280, kernel_size=(1, 1), stride=(1, 1),
bias=False)
      (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU6(inplace=True)
    )
  )
  (classifier): Sequential(
```

```
    (0): Dropout(p=0.2, inplace=False)
    (1): Linear(in_features=1280, out_features=43, bias=True)
  )
)
```

Test Function

```
def test_model(model, test_loader, device):
    """
    Test the model and return accuracy, labels, and predictions.

    Args:
        model (torch.nn.Module): The model to test.
        test_loader (torch.utils.data.DataLoader): DataLoader for the
test dataset.
        device (torch.device): The device to perform testing on (CPU
or GPU).

    Returns:
        tuple: A tuple containing accuracy, all true labels, and all
predicted labels.
    """
    model.eval()
    correct = 0
    total = 0
    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for images, labels in tqdm(test_loader, desc="Testing"):
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

            # Collect all labels and predictions for confusion matrix
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = 100. * correct / total
    print(f"Test Accuracy: {accuracy:.2f}%")

    return accuracy, all_labels, all_predictions

accuracy1, all_labels1, all_predictions1 = test_model(model,
test_loader, device)
```

```
Testing: 100%|██████████| 12630/12630 [03:01<00:00, 69.64it/s]
```

```
Test Accuracy: 95.61%


accuracy2, all_labels2, all_predictions2 = test_model(model,
brightness_test_loader, device)

Testing: 100%|██████████| 12630/12630 [02:16<00:00, 92.31it/s]

Test Accuracy: 95.51%
```

then your model is resistant to lighting changes. you gotta mention why.

```
accuracy3, all_labels3, all_predictions3 = test_model(model,
motion_blur_test_loader, device)

Testing: 100%|██████████| 12630/12630 [02:19<00:00, 90.32it/s]

Test Accuracy: 64.08%


accuracy4, all_labels4, all_predictions4 = test_model(model,
rain_test_loader, device)

Testing: 100%|██████████| 12630/12630 [02:19<00:00, 90.70it/s]

Test Accuracy: 65.62%


accuracy5, all_labels5, all_predictions5 = test_model(model,
snow_test_loader, device)

Testing: 100%|██████████| 12630/12630 [02:20<00:00, 89.81it/s]

Test Accuracy: 16.21%


accuracy6, all_labels6, all_predictions6 = test_model(model,
rotate_test_loader, device)

Testing: 100%|██████████| 12630/12630 [02:22<00:00, 88.79it/s]

Test Accuracy: 62.15%
```

That's significantly less than the V2V system.